
Electronic Theses and Dissertations, 2020-

2022

Efficient Graph-based Computation and Analytics

Bingbing Rao
University of Central Florida



Part of the [Computer Sciences Commons](#)

Find similar works at: <https://stars.library.ucf.edu/etd2020>

University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2020- by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Rao, Bingbing, "Efficient Graph-based Computation and Analytics" (2022). *Electronic Theses and Dissertations, 2020-*. 1273.

<https://stars.library.ucf.edu/etd2020/1273>



EFFICIENT GRAPH-BASED COMPUTING AND ANALYTIC

by

BINGBING RAO
M.S. University of Central Florida, 2017

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2022

Major Professor: Liqiang Wang

© 2022 Bingbing Rao

ABSTRACT

With data explosion in many domains, such as social media, big code repository, Internet of Things (IoT), and inertial sensors, only 32% of data available to academic and industry is put to work, and the remaining 68% goes unleveraged. Moreover, people are facing an increasing number of obstacles concerning complex analytics on the sheer size of data, which include 1) how to perform *dynamic graph* analytics in a parallel and robust manner within a reasonable time? 2) How to conduct performance optimizations on a *property graph* representing and consisting of the semantics of code, data, and runtime systems for big data applications? 3) How to innovate *neural graph* approaches (*i.e.*, Transformer) to solve realistic research problems, such as automated program repair and inertial navigation? To tackle these problems, I present two efforts along this road: efficient graph-based computation and intelligent graph analytics. Specifically, I firstly propose two theory-based dynamic graph models to characterize temporal trends in large social media networks, then implement and optimize them atop Apache Spark GraphX to improve their performances. In addition, I investigate a semantics-aware optimization framework consisting of offline static analysis and online dynamic analysis on a property graph representing the skeleton of a data-intensive application, to interactively and semi-automatically assist programmers to scrutinize the performance problems camouflaged in the source code. In the design of intelligent graph-based algorithms, I innovate novel neural graph-based approaches with multi-task learning techniques to repair a broad range of programming bugs automatically, and also improve the accuracy of pedestrian navigation systems in only consideration of sensor data of Inertial Measurement Units (IMU, *i.e.* accelerometer, gyroscope, and magnetometer). In this dissertation, I elaborate on the definitions of these research problems and leverage the knowledge of graph computation, program analysis, and deep learning techniques to seek solutions to them, followed by comprehensive comparisons with the state-of-the-art baselines and discussions on future research.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor Professor Liqiang Wang, for providing significant support without which this research study would not have been possible. He not only inspires me in research but also gives me valuable suggestions on life and my future career.

I would also like to show gratitude to my committee, including Associate Professor Damian Dechev, Dr. Paul Gazzillo, and Dr. Zhishan Guo. I am very grateful for their invaluable advice and patience with me. Genuine thanks to each of them for the extraordinary amount of time and knowledge they were willing to provide to my dissertation. I am deeply grateful to my former committee Professor Wingyan Chung at Western Carolina University. His enthusiasm and suggestions for the topic made a strong impression on me and I have always carried positive memories of the collaboration using dynamic graph techniques to detect users' activities.

Getting through my dissertation required more than academic support, and I have many, many people to thank for listening to and, at times, having to tolerate me over the past six years. I cannot begin to express my gratitude and appreciation for their friendship. Zixia Liu, Ehsan Kazemy, Yifan Ding, Dongdong Wang, Zihang Zou and Shengyang Liu have been unwavering in their personal and professional support during the time I spent at the University of Central Florida. For many memorable evenings out and in, I must thank everyone above as well as Dr.Devu M Shila who provided me an internship opportunity at Unknot.id.

Most importantly, none of this could have happened without my family. I dedicate this thesis to them: my parents Zuojiang Rao and Tianfeng Chen, as well as my wife Ruiyao Chen, for their endless love, selfless care, and support. Thanks to my lovely daughter Sophia who inspires me to keep going and brings me full of happiness every day.

TABLE OF CONTENTS

LIST OF FIGURES	xi
LIST OF TABLES	xiv
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: BACKGROUND AND LITERATURE REVIEW	6
2.1 Detecting Trends in Dynamic Social Networks	6
2.1.1 Theoretical Aspects of Social Network Analysis	7
2.1.2 Technical Aspects of Social Network Analysis	8
2.2 Semantics-aware Optimizations for Big Data Applications	11
2.3 Graph Transformer for Automated Programming Repair	14
2.4 Contextual Transformer For Inertial Navigation	15
CHAPTER 3: DETECTING TRENDS IN DYNAMIC SOCIAL MEDIA NETWORKS . .	18
3.1 Introduction and Motivation	18
3.2 Interaction Models for Dynamic Trend Detection	19
3.2.1 Model Notation	21

3.2.2	Benchmark Models	22
3.2.3	Random Interaction Model	25
3.2.4	Preferential Interaction Model	25
3.2.5	Measurement of Agent Activity	27
3.3	Experiment Design	28
3.3.1	Data Description	28
3.3.2	Datasets and Experimental Setup	30
3.3.3	Experimental Setup	30
3.3.4	Evaluation Metrics	32
3.4	Experimental Finding	34
3.4.0.1	Model Comparison Across Different Dates	34
3.4.0.2	Model Comparison Across Different Window Sizes	38
3.4.1	Model Comparison Using Frobenius Loss	40
3.4.2	Structural Analysis of Social Networks	42
3.4.3	Case Studies of User Interaction Patterns	43
3.4.4	Implications	45

CHAPTER 4: SEMANTICS-AWARE OPTIMIZATIONS FOR BIG DATA APPLICATIONS

4.1	Introduction and Motivation	48
4.2	System Overview	49
4.2.1	Architecture	49
4.2.2	Performance Problems	52
4.3	Semantics-Aware Data Model	54
4.3.1	Attribute-Based Data Abstraction	54
4.3.2	Primitive Operations	55
4.3.3	Data Operational Graph	56
4.4	Optimization Strategies	58
4.4.1	Cache Management	58
4.4.2	Operation Reordering	64
4.4.3	Element Pruning	67
4.5	Experiment and Evaluation	69
4.5.1	Benchmarks	69
4.5.2	Effectiveness Assessment	70
4.5.3	Performance Behavior	71

4.5.4	System Overhead	73
CHAPTER 5: GRAPH TRANSFORMER FOR AUTOMATED PROGRAM REPAIR . . .		75
5.1	Introduction and Motivation	75
5.2	System Overview	77
5.3	Pre-Processing: Context Abstraction	79
5.3.1	Token Pair Encoding	79
5.3.2	Taxonomy, Lexical Scope, and Idioms	80
5.3.3	Semantics-preserved and Scope-oriented Rename	82
5.3.4	Context Path	83
5.4	Code Translation and Context-aware Alignment	84
5.4.1	Token and Context Embedding	84
5.4.2	Context-aware Attention	85
5.4.3	Multi-task Learning: Code Translation and Context Alignment	86
5.4.4	Patch Generation via Beam Search	88
5.5	Research Questions	88
5.5.1	Quality of Context Abstraction	88
5.5.2	Overall Model Performance	91

5.5.3	Semantic Bug Repair	92
5.6	Experiment and Evaluation	92
5.6.1	Dataset	92
5.6.2	Model and Training Setting	93
5.6.3	Results of RQ1	94
5.6.4	Results of RQ2	95
5.6.5	Results of RQ3	100
CHAPTER 6: CONTEXTUAL TRANSFORMER FOR INERTIAL NAVIGATION		101
6.1	Introduction and Motivation	101
6.2	System Overall	103
6.3	Attention In Inertial Navigation	106
6.4	Jointly Learning Velocity and Covariance	108
6.5	Experiment and Evaluation	109
6.5.1	Dataset	110
6.5.2	Baseline	112
6.5.3	Evaluation Metrics	113
6.5.4	Overall Performance	115

6.5.5	Ablation Study	115
CHAPTER 7: CONCLUSION AND FUTURE WORK		129
7.1	Detecting Trends in Dynamic Social Networks	129
7.1.1	Summary of Findings	129
7.1.2	Contributions and Limitations	131
7.1.3	Future Directions	132
7.2	Semantics-aware Optimizations for Big Data Applications	132
7.3	Graph Transformer for Automated Programming Repair	133
7.4	Contextual Transformer For Inertial Navigation	134
LIST OF REFERENCES		135

LIST OF FIGURES

3.1	Data collection and experimental setup	28
3.2	Model comparison across different dates for Dataset 2013	34
3.3	Model comparison across different dates for Dataset 2015	35
3.4	Model comparison across different window sizes for Dataset 2013	38
3.5	Model comparison across different window sizes for Dataset 2015	38
4.1	The full life cycle of Semantics-Aware Optimization Approach for Data-Intensive Applications (SODA).	50
4.2	Data Operational Graph of Customer Reviews Analysis benchmark.	58
4.3	A simplified example of data dependency tree	68
4.4	The performance of individual optimization over the baseline.	72
5.1	Overview of the proposed Bug2Fix for APR tasks	77
5.2	The Context Abstraction of buggy (top panel) and fixed (bottom panel) code .	79
5.3	An AST example of Java program along with an example of one of the paths .	83
5.4	Statistics of actions performed to fix buggy code.	99

6.1	Overall workflow of the proposed contextual transformer model for inertial navigation.	104
6.2	Performance Comparison of CTIN and RoNIN variant models on CTIN dataset	117
6.3	Performance Comparison of CTIN and RoNIN variant models on RIDI dataset	117
6.4	Performance Comparison of CTIN and RoNIN variant models on OXIOD dataset	118
6.5	Performance Comparison of CTIN and RoNIN variant models on RoNIN dataset	118
6.6	Performance Comparison of CTIN and RoNIN variant models on IDOL dataset	118
6.7	The effectiveness of proposed attention layers on CTIN dataset.	119
6.8	The effectiveness of proposed attention layers on RIDI dataset.	120
6.9	The effectiveness of proposed attention layers on OxIOD dataset.	120
6.10	The effectiveness of proposed attention layers on RoNIN dataset.	120
6.11	The effectiveness of proposed attention layers on IDOL dataset.	121
6.12	The performance of CTIN network with different loss functions evaluated on CTIN dataset.	121
6.13	The performance of CTIN network with different loss functions evaluated on RIDI dataset.	122

6.14	The performance of CTIN network with different loss functions evaluated on OxIOD dataset.	122
6.15	The performance of CTIN network with different loss functions evaluated on RoNIN dataset.	123
6.16	The performance of CTIN network with different loss functions evaluated on IDOL dataset.	123
6.17	Selected visualizations of trajectories from CTIN and RoNIN variants models on the RIDI dataset.	124
6.18	Selected visualizations of trajectories from CTIN and RoNIN variants models on OxIOD dataset.	125
6.19	Selected visualizations of trajectories from CTIN and RoNIN variants models on the RoNIN dataset.	126
6.20	Selected visualizations of trajectories from CTIN and RoNIN variants models on the IDOL dataset.	127
6.21	Selected visualizations of trajectories from CTIN and RoNIN variants models on CTIN seen dataset.	128

LIST OF TABLES

2.1	Theoretical and Technical Aspects of Social Media Network Trend Detection	6
3.1	Notation and Its Meaning of Terms Used in Algorithms 1 and 2	20
3.2	Performance and P-values of Model Comparison by Date	36
3.3	Performance and P-values of Model Comparison by w	39
3.4	Performance and P-values of Model Comparison by Frobenius Loss	41
3.5	Pearson Correlations (and P-values) between Structural Properties and Model Performances for Datasets 2013 and 2015	42
3.6	Percentage of Overlap between ACTUAL Users and Users Predicted by Model	44
3.7	Interaction Patterns of Users Predicted by All Four Models to be among Top20	45
3.8	Interaction Patterns of Users Predicted by Only PIM and RIM to be among Top20	46
3.9	Structural Information of Social Media Networks on Selected Dates	46
4.1	The Definition of Primitive Operations	55
4.2	The statistics information and corresponding notations needed by SODA	57
4.3	The cache allocation policy based on Execution Distance for the workload in Figure 4.2	63

4.4	The results of running SODA on Spark Applications	70
4.5	System speed up of individual optimization over the baseline implementation in RDD.	72
4.6	Overall comparison about System Overheads incurred by SODA	73
5.1	Three datasets and their statistical information in Plaintext, BFP, and Bug2Fix, receptively.	93
5.2	<i>Success Ratio</i> achieved by models on <i>small</i> dataset.	95
5.3	Performance metrics on <i>small</i> dataset.	96
5.4	<i>Success Ratio</i> achieved by models on <i>median</i> dataset.	96
5.5	Performance metrics on <i>median</i> dataset.	97
5.6	<i>Success Ratio</i> achieved by models on <i>big</i> dataset.	97
5.7	Performance metrics on <i>big</i> dataset.	98
5.8	The top-5 targets of each action in each dataset	99
6.1	Description of public datasets used for evaluation of navigation models. . . .	110
6.2	Overall Trajectory Prediction Accuracy. The best result is shown in bold font.	114
6.3	Models' Evaluation Performance on CTIN dataset	116

CHAPTER 1: INTRODUCTION

With data explosion in many domains, such as Internet of Things (IoT) [1], scientific experiments [2, 3, 4], e-commerce, social media [5, 6] and inertial navigation, people are facing an increasing number of obstacles concerning data processing and analytics on the sheer size of these data. In this dissertation, I mainly focus on tackling the following three challenging problems using techniques of graph computation, program analysis, and deep learning:

- How to perform **dynamic graph** analytics in a parallel and robust manner within a reasonable time (cf. Chapter 3)?
- How to conduct performance optimizations on a **property graph** representing and consisting the semantics of code, data, and runtime systems for big data applications (cf. Chapter 4)?
- How to innovate **neural graph** approaches (*i.e.*, Transformer [7]) to solve realistic research problems, such as automated program repair(cf. Chapter 5) and contextual inertial navigation (cf. Chapter 6)?

To address the above issues, I first present two theory-based interaction models for detecting temporal activities in dynamic social networks in Chapter 3. Detecting nodal activities in dynamic social networks has strategic importance in many applications, such as online marketing campaigns and homeland security surveillance. How peer-to-peer exchanges in social media can facilitate nodal activity detection is not well explored. Existing models assume network nodes to be static in time and do not adequately consider features from social theories. This research developed and validated two theory-based models, Random Interaction Model (RIM) and Preferential Interaction Model (PIM), to characterize temporal nodal activities in social media networks of human agents. The models capture the network characteristics of randomness and preferential interaction due to

community size, human bias, declining connection cost, and rising reachability. The models were compared against three benchmark models (abbreviated as EAM [8], TAM, and DBMM [9]) using a social media community consisting of 790,462 users who posted over 3,286,473 tweets and formed more than 3,055,797 links during 2013–2015. The experimental results show that both RIM and PIM outperformed EAM and TAM significantly in accuracy across different dates and time windows. Both PIM and RIM scored significantly smaller errors than DBMM did. Structural properties of social networks were found to provide a simple and yet accurate approach to predicting model performances. These results indicate the models' strong capability of accounting for user interactions in real-world social media networks and temporal activity detection. The research should provide new approaches for temporal network activity detection, develop relevant new measures, and report new findings from large social media datasets. Different from prior work, the proposed models are grounded in social theories and do not assume static nodes of social networks over time. In addition, I designed and implemented these models atop Apache Spark [10] to perform graph computation using an efficient data-parallel abstraction.

In the era of data explosion, a growing number of data-intensive computing frameworks, such as MapReduce [11], Apache Hadoop [12], and Spark [13], have been proposed to handle the massive volume of unstructured data in parallel. Since programming models provided by these frameworks allow users to specify complex and diversified user-defined functions (UDFs) with predefined operations, the grand challenge of tuning up entire system performance arises if programmers do not fully understand the semantics of code, data, and runtime systems. In Chapter 4, I design a holistic *semantics-aware optimization for data-intensive applications using hybrid program analysis* (SODA) to assist programmers to tune performance issues. SODA is a two-phase framework: the *offline* phase is a *static analysis* that analyzes code and performance profiling data from the *online* phase of prior executions to generate a parameterized and instrumented application; the *online* phase is a *dynamic analysis* that keeps track of the application's execution and collects runtime

information of data and system. Technically, source code and performance log collected in prior executions are analyzed to refactor code by applying three kinds of optimizations: cache management, operation reordering, and element pruning. The offline phase is developed as a compiler plugin of the host development languages (*e.g.*, Scala, Java). However, not all performance issues can be fixed in the offline phase, while some may need support from system runtime information, such as intermediate data size, memory usage, and execution time of operations. The second phase is an *online dynamic analysis* to obtain the required runtime information, where I instantiate a parameterized framework based on the instrumentation generated in the offline phase to trace applications' execution and extract profiling information concerning data and system status. To the best of my knowledge, the implementation is the first compiler plugin to help users optimize data-intensive applications. Extensive experimental results on four real-world Spark applications show that SODA can gain up to 60%, 10%, 8%, faster than its original implementation, with the three proposed optimization strategies, *i.e.*, cache management, operation reordering, and element pruning, respectively.

Recently, a growing number of research studies are harnessing the power of graph-based deep learning techniques [14] to assist software engineering tasks, such as code completion [15, 16, 17], code clone [18, 19], bug localization and repair [20, 21, 22, 23, 24, 25, 26]. Although there are a few endeavors to repair programs by learning neural language models (NLM), many program properties, such as structure and semantics of identifiers as well as context alignment, are not well handled in sequence embedding and model design, which results in undesired performance. In Chapter 5, I propose a novel Transformer-based approach with multi-task learning of code translation and context-aware alignment to detect and fix a broad range of programming bugs automatically. First, a novel *semantics-preserved, scope-oriented, and vocabulary-closed* context abstraction approach considering code structural and semantic meanings are designed to pre-process program code and reduce vocabulary size. Then, a context-aware attention technique is designed

and orchestrated with token attention probabilities from the encoder-decoder attention sub-layers in Graph Transformer to learn context-aware alignment [7]. Finally, I conduct multi-task training by considering the homoscedastic (*i.e.* task-dependent) uncertainty of each task to detect and fix bugs accurately and automatically. I implement the proposed approach in a tool called Bug2Fix and evaluate its performance comprehensively on three datasets by generating patches to buggy code. The experimental results show that Bug2Fix significantly outperforms the state-of-art techniques by successfully predicting 56.16%, 34.12%, and 51.90% of the fixed code in these three datasets, respectively. These success rates steadily increase along with the increase of beam size. Besides, the overall syntactic correctness of all patches remains above 97%, 90%, and 49% on the three benchmarks, respectively, regardless of the beam size.

As a follow-up project on the neural graph, I extend the knowledge of graph neural Transformer network to a new domain of inertial navigation in Chapter 6. Inertial navigation is a never-ending endeavor to estimate the states (*i.e.* position and orientation) of a moving subject (*e.g.* pedestrian) by using only IMUs attached to it. An IMU sensor, often a combination of accelerometers and gyroscopes, plays a significant role in a wide range of applications from mobile devices to autonomous systems because of its superior energy efficiency, mobility, and flexibility [27]. The conventional Newtonian-based solutions to inertial navigation can benefit from these low-cost inertial sensors to approximate positions and orientations [28]. Nevertheless, the conventional Newtonian-based inertial navigation methods reveal not only poor performance but also require unrealistic constraints that are incompatible with everyday usage scenarios. Furthermore, data-driven inertial navigation approaches have been proposed and demonstrated their capability of using well-trained neural networks to obtain accurate position estimates from inertial measurement units (IMUs) measurements. In this project, I propose a novel robust **Contextual Transformer**-based network for **Inertial Navigation (CTIN)** to accurately predict velocity and trajectory. To this end, I first design a ResNet-based [29] encoder enhanced by local and global multi-head self-attention to capture

spatial contextual information from IMU measurements. Then we fuse these spatial representations with temporal knowledge by leveraging multi-head attention in the Transformer decoder [7]. Finally, multi-task learning with uncertainty reduction is leveraged to improve learning efficiency and prediction accuracy of velocity and trajectory [30, 31, 32, 33]. To the best of my knowledge, CTIN is the first Transformer-based model for inertial navigation. Through extensive experiments over a wide range of inertial datasets (*e.g.* RIDI, OxIOD, RoNIN, IDOL, and our own), CTIN is very robust and outperforms state-of-the-art models¹.

¹Note that the mathematical notations are consistent in each individual chapter, but the same symbol may refer to different concepts in different chapters.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

2.1 Detecting Trends in Dynamic Social Networks

As social media facilitate the formation of temporal online networks, the theoretical and technical aspects of social media analytics are important for research development [34]. Social and behavioral theories can help to guide the research into online social network trend detection [35], whereas traditional theories may not address specific network characteristics, such as size, partici-

Table 2.1: Theoretical and Technical Aspects of Social Media Network Trend Detection

Aspect	Basis	Construct/Explanation	Prior Work
Theoretical Aspect	Social position	Homophily theory: similar entities create social links more likely than dissimilar entities	[McPherson et al. 2001; McPherson and Ranger-Moore 1991; Wu et al. 2017]
		Social impact theory: impact is a multiplicative power function of the strength, immediacy, and number of other people who exert the impact	[Latané 1981; Sedikides and Jackson 1990; Kelman 1958; Ngai et al. 2015; Chang and Chuang 2011; Portes 1998]
		Social interaction theory: people make decisions based on their social neighbors' decisions	[Becker 1974; Akerlof 1997; Salancik and Pfeffer 1978]
Technical Aspect	Cognition and perception	Cognitive balance theory: entities that link to a third common entity tend to establish a connection	[Heider 1958; Lee et al. 2013; Zheng et al. 2015]
		Primacy and recency models: people are affected by either primary (early) or recent (later) impressions	[Hovland 1957; Deese and Kaufman 1957; Miller and Campbell 1959]
	Dynamic Graph Modeling	Modeling of dynamic social networks is done through representing entities as nodes and edges in graphs that change with time	[Xu and Hero 2014; Karrer and Newman 2011; Rohe et al. 2011; Leskovec et al. 2010; Jiang and Chen 2016]
		Link prediction	Learning-based approach: derive a model from existing links and use it to classify each non-connected pair of nodes as positive (linked) or negative (not linked) class
	Similarity-based approach: compute similarities of non-connected pairs of nodes and create links for top-ranked pairs		[Leskovec et al. 2005; Dunlavy et al. 2019; Gao and Liu 2017; Du et al. 2015]
	Node activity detection	Node activity detection	Nodal feature characterization: extract node features to identify activities
Nodal content characterization: examine topics and messages posted by nodes and structural proximities			[Pinto et al. 2015; Qiu et al. 2014; Antoniadis and Dovrolis 2015]

pants, diffusion, and evolution. Technical aspects of temporal online social networks are founded on social media analytics [36, 37, 38] and on dynamic graph modeling [39, 40]. In particular, temporal network dynamics, a specialty area in social network analysis, can be used to reveal changes in human activities in a network over time [41]. Table 2.1 summarizes the theoretical and technical aspects of this review.

2.1.1 *Theoretical Aspects of Social Network Analysis*

Theories help us understand why and under what circumstances a social network trend detection method works. A number of social, economic, and behavioral theories serve as the foundations of social network analysis [42]. For instance, homophily theory [43], social impact theory [44], social interaction theory [45], and cognitive balance theory [46] have been used to explain the formation of social links. This section highlights theories relevant to the research topics. A comprehensive review of these theories and related theoretical questions can be found in Reference [47] and Reference [48], respectively.

- *Theories Based on Social Position.* Homophily theory [43] states that similar entities create social links more likely than dissimilar entities. The famous saying “birds of a feather flock together” reflects the social choice based on similarity. People tend to link to others similar to them, and such linkage strengthens their similarity. In a social network, similarities can be observed between user profiles, geographic proximities, social engagements, and social relationships [49]. Changes in these features over time need to be incorporated for detecting dynamic trends in the networks. Social impact theory states that when other people are the source of impact and the person is the target, the impact is a multiplicative power function of the strength, immediacy, and a number of other people [44, 50]. Various aspects have been considered to produce social impact, including social influence, social identity [51, 47], and

social capital [52, 53]. Social networks provide the conduit of social impact that may be quantified and tracked over time [41]. Social interaction theory states that people make decisions based on the decisions of their social neighbors [45]. The social position plays a key role in impacting the decisions [54]. A social entity's position in a network creates a tendency for itself to conform to its neighborhood and allows information transmission to its neighbors [55]. The social neighborhood is therefore a foundation of the theory to examine the activity of social entities over time.

- *Theories Based on Cognition and Perception.* Cognitive balance theory states that two social entities that link to a third common entity tend to establish a connection [46]. In a social network, nodes that link to another well-connected node (or “Rockstar” [56]) are likely to associate with each other [57], thereby strengthening the position of the rock-star over time. Human cognition is also limited by the order of information received [58]. The primacy effect states that people are more affected by first impressions than later ones [59], while the recency effect states that people are more influenced by the latest information that they recall than the earlier one [60]. These effects can be amplified in online social networks over time. Detecting dynamic nodal activities in social networks needs to model these effects explicitly.

2.1.2 *Technical Aspects of Social Network Analysis*

Technical developments in social network analysis proliferated since the Internet became available in the 1990s [61, 62] and continued to flourish in the 2000s and beyond [36, 63, 64, 65]. Data about online social network activities and structure, if handled appropriately, could revolutionize the understanding of collective human behavior [66]. The following reviews various technical approaches for dynamic graph modeling, and temporal network dynamics focusing on link prediction and node activity detection.

- *Dynamic Graph Modeling.* Dynamic graph models have been developed to represent changing networks in a variety of applications. Xu and Hero III developed a dynamic stochastic blockmodel that performed competitively with a state-of-the-art algorithm and achieved a higher computational efficiency [40]. Karrer and Newman developed a heuristic blockmodel algorithm for community detection [39]. For a network generated from the Stochastic Block-model, Rohe et al. bound the number of nodes “misclustered” by spectral clustering [67]. Rossi et al. developed a temporal behavior model that extracts nodal roles in an evolving network using non-negative matrix factorization [9]. The approach is fully automatic but assumes static nodes and does not consider social theories about user interaction and network positions. Jiang and Chen consider nodal attributes for dynamic network analysis and achieved an increased prediction accuracy [68]. Leskovec et al. developed Kronecker graphs to generate realistic networks using a non-standard matrix operation [69]. A comprehensive survey of statistical network models found limited methodological work focusing on evaluating and comparing the predictive ability of various models [70]. Although prior research on link prediction in relational networks exists, these works focus on combining heterogeneous data to discover new links.
- *Link-prediction Approaches.* Link prediction aims to predict new links or deleted links between nodes for a future time [35]. Leskovec et al. studied a range of real-world graphs and developed a forest-fire model to produce graphs that exhibit properties observed in these graphs [71]. They observed densification of networks over time, as the log-log plot (number of edges $e(t)$ versus the number of nodes $n(t)$) shows increasing trends across different real-world datasets (ArXiv citation, patent citation, autonomous systems, and affiliation). Wu et al. incorporate social theories to model the evolution of social network user-item and user-user interactions [72]. They quantified social influence and homophily effect on users’ behaviors by using a joint model of user consumption prediction and social link prediction.

Dunlavy et al. show that both matrix and tensor-based techniques are effective for temporal link prediction [73]. Gao et al. developed a link prediction model that integrates the global network structure, the content of nodes, and the local or proximity information of a given node [74]. Li et al. developed a deep learning framework using conditional temporal restricted Boltzmann Machine that predicts links based on individual transition variance and influence introduced by node neighbors [75]. Researchers have developed dynamic influence models to study social learning and networking in social media websites [76]. Influence is modeled as conditional dependence between an entity's current state and previous states based on a Markovian assumption and a Hidden Markov Model (HMM) [77]. Community strength was measured to reveal robustness and coherence and was applied to discovering community progression and formation [78]. Simulated experiments were conducted to assess a network-based model that represents both spatial and temporal characteristics of human dynamic behavior [79].

- *Node Activity Detection Approaches.* Network nodes have been characterized to reflect their activities over time. Two classes of dynamic network metrics, emergence, and persistence, were developed and applied to temporal network activity prediction [8]. Among four models tested, the exponential aggregation model was found to outperform the average aggregation and linear aggregation models. Temporal detection was formulated as weighted time-series forecasting. Pinto et al. developed a trend detection algorithm to identify topical trends in social networks using a Hawkes process, a self-exciting point process that considers broadcasting times and user topic interaction [80]. Zhu et al. developed a socially regularized time-decay model for user activity prediction and cast the problem as a classification of active and inactive users [81].

The temporal stability of network nodes was studied in Reference [82], which developed a method to identify active valuable nodes based on static structural properties and Spatio-

temporal attributes. Experiments on two online social networks with thousands of nodes showed that the method identified valuable nodes in terms of node stability and influence. In another research [83], a simple probabilistic model was developed to capture the probability of tweet-retweet-follow (TRF) events. The model provides descriptive statistics on TRF events but falls short of predictive power.

2.2 Semantics-aware Optimizations for Big Data Applications

There is an increasing number of obstacles concerning data processing and analytics on the sheer size of these unstructured data. These obstacles include interactive computing and user-specific element-wise data transformations [84]. To break through these dilemmas, a growing number of data-intensive computing frameworks have been proposed, such as MapReduce [11], Apache Hadoop [12], and Spark [13]. Generally, a mainstream approach to gaining computing capability and scalability behind these platforms is to distribute data and computations across a cluster of nodes so that a large volume of data can be processed in a parallel and robust manner within a reasonable time [85, 86]. The successes of these frameworks owe to their MapReduce-like programming models, which are further based on data distribution techniques (*e.g.*, Resilient Distributed Dataset (RDD) in Apache Spark [87]), and high-order functions (*e.g.*, *map*, *reduce*, *filter*) that can take user-defined functions (UDFs) as arguments. The semantics of these high-order functions facilitate data-parallelism to manipulate datasets in an element-wise way while UDFs are applied to each element to produce the desired result. The programming models hide the details of scheduling, load balancing, execution, and communication from programmers, which eases programming and allows programmers to focus on data flow and UDF designs. A surge of interest in optimizing data-intensive applications using semantics-aware approaches has emerged [88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98].

Manimal is a framework that applies relation-style query optimizations for data-centric MapReduce programs via static analysis techniques [91]. An analyzer inspects already-compiled MapReduce codes before execution to automatically discover possible issues. Then an optimizer creates a pre-computed B^+ tree index to slice input data in a manner of column-wise to avoid reading unused data. In addition, semantics-aware compression techniques have been designed to reduce I/O operations for efficient representation of numerical values. There are two limitations in Manimal: 1) there is no runtime information about data and systems, and 2) the optimizations consider only selection and projection operations in a single MapReduce job. SODA can leverage profiling data from dynamic program execution to optimize an application with multiple MapReduce jobs.

Panacea is a framework for performing holistic compiler optimizations on legacy MapReduce applications [92]. It uses static program analysis to carry out additional runtime legality checks (*e.g.*, convergence loop verification) on the map and reduce functions and utilizes a trace analyzer to execute the optimized application several times for parameter auto-tuning. Although Panacea uses a hybrid way by integrating information from static and dynamic analyses, there is one limitation: statistical data needed by the trace analyzer is not real data. While SODA uses real profiling data to determine performance behaviors.

Microsoft's Scope compiler automatically optimizes a data-parallel program to eliminate unnecessary code and data [99, 100, 90, 98]. It performs early filtering and calculates small derived values to minimize the amount of data-shuffling I/O based on information derived by static code analysis. However, there is no dynamic information involved in its optimizations, which leaves some runtime-related problems for SODA.

As an extensible logical optimizer for UDF-heavy data flows, Stratosphere SOFA extracts a concise set of properties for describing the semantics of Map/Reduce-style UDFs via static analysis techniques and uses these properties and a set of rule-based rewrite templates to infer semantically

equivalent plans, then these plans would be evaluated by a cost-based optimizer [88, 94]. SODA extends this work to a more general scenario with two upgrades: 1) Redefining primitive operations with more general constraints so that SODA can easily verify if semantics change after reordering; 2) Profiling data are used to determine performance behaviors.

Spark Catalyst is an extensible query optimizer that leverages advanced programming language features (*e.g.*, Scala’s pattern matching and quasi-quotes) in a novel way within the core of the Spark SQL engine [89]. Catalyst uses a tree architecture to represent operation nodes and conduct rule-based optimizations. Finally, cost-based optimization is performed by generating multiple plans and calculating their cost to choose an optimal one. Unfortunately, it only supports Dataset and DataFrame APIs [101, 102]. Thus, applications developed by RDD API cannot benefit from it directly.

As to cache management, LRU caching policy is often used [13, 103]. To improve cache management, several research works, namely MemTune [104], LRC [105] and MRD [106], leverage directed acyclic graph (DAG), data dependency among stages, and physical schedule unit (*i.e.*, job and stage level) for new measurements of a data block reference. However, MemTune fails to answer the question of which and when each RDD will be persisted in memory. LRC updates a new reference count for each data block according to usages within a stage, however, it does not take into account the impact of data blocks spanning across multiple stages. MRD proposes a fine-grained time-locality measurement of data block reference, called reference distance. It is based on a physical schedule unit assigned by the DAG Scheduler. Nonetheless, scheduling unit orders can not reveal the real runtime executing an order to some extent. the approach in SODA is a novel stage-level global cache management policy, which emphasizes two factors that would impact system performance, especially for cache behaviors: execution order of stages and data block size.

2.3 Graph Transformer for Automated Programming Repair

Automated program repair (APR) is a never-ending endeavor to fix software bugs without human intervention [107, 108, 24]. The conventional solutions to detect and fix bugs focus on rule-based methods ranging from static approaches (*e.g.*, program analysis, bug detection, model checking, validation, and verification) to dynamic methods (*e.g.*, testing, debugging, and fault localization). However, these existing methods reveal not only poor performance but also involve extra overhead to manually define new types of bugs and corresponding rules [109, 110]. Thus, to avoid defining tedious rules, mining-based approaches have been proposed to automatically extract implicit programming patterns from code (*e.g.*, frequent itemsets or sub-graphs), then to detect potential bugs [111, 112, 113]. Nonetheless, these approaches still embrace a few critical limitations, such as an exceptionally high false-positive rate due to that they cannot distinguish incorrect code patterns from infrequent/rare correct patterns.

Recently, a growing number of research studies have taken advantage of the naturalness of software [14] to learn statistical patterns from previously reported buggy/fixed codes, then automatically detect and repair bugs in new ones [20, 114, 115, 21, 22, 116]. Specifically, the APR task can be considered a type of neural machine translation (NMT) task [117, 23] that learns translation from buggy code to produce the corresponding fixed version. With the recent advances in Neural networks, deep learning-based approaches have been designed for automated program repair tasks, which can be classified as follows.

Learning Similarity. To generate fixed code, neural models learn similarities between buggy code and fixed corpus by leveraging the capability of deep learning models [20, 118, 119]. For instance, DeepFix [20] proposes an attention-based model to first predict the buggy line and then generate a replacement statement as the repair. DeepRepair [119] utilizes a recursive auto-encoder [120] to select repair ingredients from code fragments similar to the buggy code.

Neural Machine Translation. In this category, an encoder-decoder architecture is often employed to learn translation from buggy code to fixed one [121, 115, 114, 21, 22, 116]. For instance, CODIT [115] integrates code structure in an NMT model to learn code edits and then generate fixes. SequenceR [21] and BFP [22] take advantage of sequence-to-sequence with different code abstractions to reduce the vocabulary size and then repair program errors at class-level and method-level, respectively. Particularly, SequenceR employs a sequence-to-sequence model with a copy mechanism to fix bugs, while BFP uses an LSTM-based NMT model with an attention-based encoder-decoder to generate patches. The work [116] learns code changes using sequence-to-sequence NMT with code abstraction and keyword replacing.

Structure Transformation. Recently, neural structure transformation is leveraged to detect and repair bugs with the help of some auxiliary information, such as AST, control and data flow, compiler diagnostic messages [24, 122, 123, 124]. For instance, Graph2Diff [125] uses a Graph Neural Network model to explore correlations among source code, build configuration files, and compiler diagnostic messages, and then predict a diff specifying how to modify the code’s AST. Hoppity [123] frames the problem of detecting and fixing bugs in terms of learning a sequence of graph transformations: given a buggy program modeled by a graph structure, it makes a sequence of predictions including the position of bug nodes and corresponding graph edits to produce a fix. The works [24, 122] leverage local context extracted from AST and global context from Program Dependence Graph (PDG) and Data Flow Graph (DFG) to learn context-based code presentation and then perform neural code transformation to detect and repair bugs.

2.4 Contextual Transformer For Inertial Navigation

Inertial navigation is a never-ending endeavor to estimate the states (*i.e.* position and orientation) of a moving subject (*e.g.* pedestrian) by using only IMUs attached to it. An IMU sensor, often

a combination of accelerometers and gyroscopes, plays a significant role in a wide range of applications from mobile devices to autonomous systems because of its superior energy efficiency, mobility, and flexibility [27].

Conventional Newtonian-based solutions to inertial navigation can benefit from IMU sensors to approximate positions and orientations [28]. In a strap-down inertial navigation system (SINS) [126], accelerometer measurements are rotated from the body to the navigation frame using a rotation matrix provided by an integration process of gyroscope measurements, then subtracted from the earth's gravity. After that, positions can be obtained by double-integrating the corrected accelerometer readings [127]. However, the multiple integrations can lead to exponential error propagation. To compensate for this cumulative error, step-based pedestrian dead reckoning (PDR) approaches rely on the prior knowledge of human walking motion to predict trajectories by detecting steps, estimating step length and heading, and updating locations per step [128]. Nevertheless, the conventional Newtonian-based inertial navigation methods reveal not only poor performance but also require unrealistic constraints that are incompatible with everyday usage scenarios. For example, strap-down inertial navigation systems (SINS) may obtain erroneous sensor positions by performing double integration of IMU measurements, due to exponential error propagation through integration [129]. Step-based pedestrian dead reckoning (PDR) approaches can reduce this accumulated error by leveraging the prior knowledge of human walking motion to predict trajectories [128]. However, an IMU must be attached to a foot in the zero-velocity update [130] or a subject must walk forward so that the motion direction is constant in the body frame [131].

Recently, a growing number of data-driven approaches such as IONet [132], RoNIN [133], and IDOL [134] have demonstrated their capability of using well-trained neural networks to obtain accurate estimates from IMU measurements with competitive performance over the aforementioned methods. IoNeT [132] first proposed an LSTM structure to regress relative displacement in 2D polar coordinates and concatenate to obtain the position. In RIDI [135] and RoNIN [133], IMU

measurements are first rotated from the body frame to the navigation from using device orientation. While RIDI regressed a velocity vector from the history of IMU measurements to optimize bias, then performed double integration from the corrected IMU samples to estimate positions. RoNIN regressed 2D velocity from a sequence of IMU sensor measurements directly, and then integrate positions. An LSTM-based model for classifying pedestrian activities of running and walking is exploited to provide more refined thresholds for zero-velocity detection (ZUPT) [136].

In addition, inertial sensors are often combined with additional sensors and models using Extended Kalman Filter [137] to provide more accurate estimations, where the typical sensors include WiFi [138], Bluetooth [139], LiDAR [140], or camera sensors [141]. Nonetheless, these combinations with additional sensors are posing new challenges to instrument installations, energy efficiency, and data privacy. For instance, Visual-Inertial Odometry (VIO) substantially depends on environmental factors such as lighting conditions, signal quality, blurring effects [142]. Additionally, It is also very power-demanding that seriously limiting its applications in constrained platforms. An end-to-end differentiable Kalman filter framework is proposed in Backprop KF, in which the noise parameters are trained to produce the best state estimate, and do not necessarily best capture the measurement error model since the loss function is on the accuracy of the filter outputs. TLIO provides a neural model to regress the velocity prediction and uncertainties jointly [31]. The predictions are further applied in the Kalman filter framework as an innovation, where the covariance noise measurement of the Kalman filter is generated by the same deep model. In IDOL [134] two separate networks in an end-to-end manner are exploited. The first model is used to predict orientations to circumvent the inaccuracy in the orientation estimations with smartphone APIs. Next, the IMU measurements in the world frame are used to predict the velocities using the second model.

CHAPTER 3: DETECTING TRENDS IN DYNAMIC SOCIAL MEDIA NETWORKS

3.1 Introduction and Motivation

Detecting nodal activities in dynamic social networks has strategic importance in many applications. For instance, e-commerce marketers, homeland security experts, and customer relationship managers study social networks of interested actors (e.g., customers, terrorists, and online forum users) to examine activity trends. In recent years, social media has rapidly emerged as a source of intelligence and business value [143]. The economic impact of social media on business is estimated to exceed 1 trillion due to more efficient communication and collaboration [144]. Advanced analytics play an important role in supporting understanding and trend analysis in social media [37]. In particular, social media networks provide a diverse repertoire of theories and frameworks to describe, analyze, and explain emerging behaviors in organizations and in society [34]. These networks capture the information of the agents (modeled as nodes or vertices) and their relationships (modeled as links or edges).

Our literature review shows that most technical approaches predict links based on static snapshots of social networks. For instance, the work described in Reference [75] models node transition and local neighbor influence but assumes the nodes to be static over time. This assumption may not be realistic given that people join and leave social networks frequently. Existing works also do not adequately consider features from social theories; and evaluations of link prediction approaches are inadequate, which may lead to wrong conclusions [35]. Prior research does not examine social and economic theories underlying social network link prediction methods [42]. In the prior works about stochastic blockmodeling (e.g., References [69, 40]), model fitting is the primary focus, thus

emphasizing explanation at the expense of prediction. Nodal attributes were shown to increase the prediction accuracy of network characteristics [68], but not on an individual nodal activity level. In addition, changes in an agent’s (nodal) activities are often modeled as a function of the agent’s historical behavior (e.g., Reference [8]) without adequately considering network connectivity, social influence, and social theories. Aggregation functions in linear or exponential forms do not account for the rich interaction among agents, thus adversely affecting the detection of nodal activities in temporal networks.

The contributions of this research are threefold. First, I developed two new models for temporal social network activity detection and have shown their novelty and robustness in detecting nodal activities in social media networks. Second, I provided new experimental findings to validate the performance of the proposed models, offering insights into understanding agent behaviors in large-scale social networks. Third, I developed several metrics for evaluating social network nodal activity detection and built a reusable implementation of temporal activity detection for large online social networks. The methods and findings should be useful to researchers and practitioners.

3.2 Interaction Models for Dynamic Trend Detection

To address the research gaps, I developed two theory-based interaction models to support dynamic social network activity detection and provided their instantiations, as shown in Algorithms 1 and 2, respectively. Their notation and meaning are listed in Table 3.1. The two interaction models, namely, “Random Interaction Model” and “Preferential Interaction Model”, detect agent activities based on different assumptions of agent behavior. Different from existing models (e.g., References [69, 67]), the proposed models exploit interaction among agents who form the network; cover multiple, consecutive time points of an evolving network; and are grounded in social theories and do not assume static nodes of social networks over time. Existing works such as the Erdős-

Table 3.1: Notation and Its Meaning of Terms Used in Algorithms 1 and 2

Notation	Meaning
i	Index of an agent that represents a node in a network
t	A point in time to represent a snapshot of an evolving network and its nodal activities
V_t	A set of vertices at time t
E_t	A set of edges at time t
$L(w)$	A function that specifies an entity spanning w time steps
$G_t^{L(w)}$	A dynamic graph consisting of vertex set $V_t^{L(w)}$ and edge set $E_t^{L(w)}$, representing an evolving network of nodes and links spanning the period $(t - w, t)$
v_t	Number of vertices contained in $V_t^{L(w)}$
e_t	Number of edges contained in $E_t^{L(w)}$
\hat{v}_{t+1}	Predicted number of vertices that will be contained in $V_{t+1}^{L(w)}$
\hat{e}_{t+1}	Predicted number of edges that will be contained in $E_{t+1}^{L(w)}$
G'	$G_{t+1}^{L(w)}$, a dynamic graph spanning the period $(t - w + 1, t + 1)$
d'	Density of G' . A ratio of the predicted number of edges in G' to the predicted number of possible edges in G'
$C_B(i, G')$	Betweenness centrality of Agent(i) in G'
$P(ij)$	Number of shortest paths connecting Agent(i) and Agent(j)
$P_k(ij)$	Number of shortest paths that connect Agent(i) and Agent(j) and that Agent(k) lies on
p_j	Degree ratio: a ratio of the degree of Agent(j) to the maximum degree of any single agent in the network G'
q_i	Complement of degree ratio of Agent(i). Used in evaluation of link removal
u	A set of agents sorted by ascending order of agents' degree ratios
α	A real-valued number $\in [0,1]$ that controls the primacy and recency effects in prediction

Rényi [145], Gilbert [146], preferential attachment [147], and small-world models aim to represent single time-point snapshots of a network [70]. By contrast, my models aim to represent multiple, consecutive time points of an evolving network.

The theoretical rationale of the interaction models can be explained in four aspects. (1) Homophily theory [49, 43] directs the generation of the new dynamic network to resemble the existing link structure and network density. This is reflected in line 6 of Algorithm 1 and lines 11, 17, 25, 30 of Algorithm 2; (2) The primacy and recency models of human cognition theory suggest that future human activities depend on their preferences for recent memory or first impression (older memory) [59, 58, 60]. These are modeled in line 5 of Algorithm 1 and in line 5 of Algorithm 2; (3) According to social interaction theory, people make decisions based on their social neighbors' decisions [45].

A social entity's position in a network affects information exchange with neighbors [55] and thus impacts his/her activities. Therefore, the social positions of nodes reflect activities in a social network [54, 45, 55]. Accordingly, the interaction models represent nodal activities by using the betweenness centrality algorithm that considers nodal position in a network (line 22 of Algorithm 1; line 34 of Algorithm 2); (4) Social impact theory suggests that impact is a multiplicative, power function of strength, immediacy, and a number of other people [51, 44, 50]. This directs the development of the selective procedure of forming preferential interaction in Algorithm 2 (lines 6–33).

3.2.1 Model Notation

This section provides algorithmic notation and steps of the interaction models. Let $G_t^{L(w)}$ be an undirected network that consists of V_t agents (= nodes or vertices) and E_t links (= edges) at time t . Also, let $L(w)$ be the function that specifies the network to span the most recent w time steps at and before time t .

$$G_t^{L(w)} = \{V_t^{L(w)}, E_t^{L(w)}\} \quad (3.1)$$

The proposed interaction models were designed to learn from the network characteristics of randomness and preferential interaction due to community size, human bias, declining connection cost, and rising reachability. Three general steps are used in detecting the temporal network activities of agents. (1) Initiation: The models identify the existing network's configuration (numbers of human agents and relationships) at time t , and the time frame (w days at and before t) during which agent activities are considered in activity prediction. (2) Simulation of network for time $(t + 1)$: Based on an existing network, the models predict the configuration (numbers of human agents and

relationships) of the network at $(t + 1)$ by using a single exponential smoothing procedure. (a) Derive the density of the network at time $(t + 1)$ (Equation 3.2) based on the predicted network configuration. (b) Alter agents in the network based on the projected interaction of agents:

$$D(g) = \frac{\text{Number of links in Network } g}{\text{Maximum Possible number of links in Network } g} \quad (3.2)$$

(3) Predicting activity levels of agents: The prediction uses the betweenness centrality (BC) formula to represent Agent i 's activity level [148, 149], as in Equation 3.3.

$$C_B(i, G') = \sum_{i, j, k \in G'; i \neq j, k \notin \{i, j\}} \frac{P_k(ij)/P(ij)}{D(G')} \quad (3.3)$$

While other measures may be used, BC reflects the connectivity of agents and can reveal agents' interaction and influence in the network. The aforementioned steps are to be further explained in Sections 3.2.3 and 3.2.4. Section 3.2.2 describes models used to compare against the proposed interaction models.

3.2.2 Benchmark Models

Three models, EAM, TAM, and DBMM, were used as benchmarks to compare against the two interaction models. The Exponential Aggregation Model (EAM) was found in Reference [8] to outperform two other temporal prediction models in empirical studies of four different datasets. The method predicts Agent i 's activity level by using a single exponential smoothing method that assigns high weights to activities that are closer to the current time step. The assignment is controlled by a smoothing constant $\alpha \in [0, 1]$ that reflects recency and primacy effects according to cognitive theories [59, 60]. High values of α indicate that recent data are more emphasized in prediction.

EAM has an implicit assumption that an existing trend will continue to be extended in the same intensity as in reference history. However, this may not be always true due to the multiplicative effects of human influence [44, 50]. Therefore, I developed another benchmark model named Trend-adjusted-exponential Aggregation Model (TAM), which includes a trend adjustment term $\frac{m_t - m_1}{t - 1}$ to incorporate the changing temporal trends present in the community. TAM represents the activity of a node as follows:

$$Activity^{T(\alpha \in [0,1])}(m \in \mathbb{R}^d) = [\alpha^{d-1}m_1 + \sum_{j=2}^t (1 - \alpha)\alpha^{d-j}m_j] + \frac{m_t - m_1}{t - 1} \quad (3.4)$$

where m_t represents the actual activity level of the node at time t ; α is a parameter controlling primacy and recency effects, and d is the size of the time window of reference history. In addition, the Dynamic Behavioral Mix-membership Model (DBMM) [9] was used to compare against the interaction models to examine the accuracy of role discovery of agents in temporal social networks. Different from EAM and TAM, DBMM does not predict Agent i 's activity level but predicts the likelihood of possessing different roles in the network. In DBMM, a network at time t is represented as a node-feature matrix using the approximation $G_t F \approx V_t$, where each row of G_t (an n (= node count) \times r (= role count) matrix) represents an agent (node)'s membership in different roles and each column of F (an $r \times f$ (= feature count) matrix) represents the extent to which a role contributes to feature values. DBMM consists of three steps: network feature extraction, probabilistic role discovery, and behavioral transition modeling. According to Rossi et al. [9], the summary transition model (which defines a summary behavioral snapshot $G_{S(t)}$ shown below) used in DBMM achieved best performance (with $k = 10$, $\alpha = 0.7$) and is thus used in the implementation:

$$G_{S(t)} = \alpha^{w-1}G_k + \sum_{j=2}^w (1 - \alpha)\alpha^{w-j}G_j \quad (3.5)$$

where $k = t - w$ and w, α are window size and parameters to determine the contribution of previous

Algorithm 1: Predicting the activity of Agent i with Random Interaction Model

Input : i, G, α, w, t
Output: Activity of Agent i at time $(t+1)$

```

1  $V_{t+1}^{L(w)}, E_{t+1}^{L(w)} = \{\}, \{\}; G' = \{V_{t+1}^{L(w)}, E_{t+1}^{L(w)}\}$  /* A graph with empty node and links */
2  $v_t, \hat{v}_{t+1} = |V_t^{L(w)}|, |V_{t+1}^{L(w)}|; e_t, \hat{e}_{t+1} = |E_t^{L(w)}|, |E_{t+1}^{L(w)}|$  /* The numbers of nodes and links */
3 for  $i \leftarrow V_t^{L(w)}$  do  $V_{t+1}^{L(w)} = V_{t+1}^{L(w)} \cup Agent(i)$ 
4 for  $i \leftarrow E_t^{L(w)}$  do  $E_{t+1}^{L(w)} = E_{t+1}^{L(w)} \cup Link(i)$ 
/* Predictions for numbers of agents and links and for density */
5  $\hat{v}_{t+1} = \alpha v_t + (1 - \alpha) \hat{v}_t; \hat{e}_{t+1} = \alpha e_t + (1 - \alpha) \hat{e}_t$  /*  $\hat{v}_2 = v_1$  and  $\hat{e}_2 = e_1$  */
6  $d' = \frac{2\hat{e}_{t+1}}{\hat{v}_{t+1}(\hat{v}_{t+1}-1)}$  /* Density of graph  $G'$  */
/* Prediction for  $G'$  */
7 if  $\hat{v}_{t+1} > v_t$  then
/* Add new nodes */
8 for  $i = v_t + 1$  to  $\hat{v}_{t+1}$  do
9  $V_{t+1}^{L(w)} = V_{t+1}^{L(w)} \cup Agent(i)$ 
10 for  $j \in V_{t+1}^{L(w)}$  do
11  $r = U([0, 1]);$  /* Generating a random float number  $r \in [0, 1]$  */
12 if  $r \leq d'$  then  $E_{t+1}^{L(w)} = E_{t+1}^{L(w)} \cup Link(i, j)$ 
13 else if  $\hat{v}_{t+1} < v_t$  then
/* Delete nodes */
14 for  $i \in V_{t+1}^{L(w)}$  do
15  $r = U([0, 1]);$  /* Generating a random float number  $r \in [0, 1]$  */
16 if  $r \leq \frac{v_t - \hat{v}_{t+1}}{v_t}$  then
/* Deleting a node and all links connected to that node */
17  $V_{t+1}^{L(w)} = V_{t+1}^{L(w)} - \{Agent(i)\}; E_{t+1}^{L(w)} = E_{t+1}^{L(w)} - \{Links\ of\ Agent(i)\}$ 
18 else
19 if  $\hat{e}_{t+1} > e_t$  then
20 for  $i, j \in V_{t+1}^{L(w)}$  where  $i \neq j$  do
21  $r = U([0, 1]);$  /* Generating a random float number  $r \in [0, 1]$  */
22 if  $\{Link(i, j)\} \notin E_{t+1}^{L(w)}$  &  $r \leq d'$  then Add Link( $i, j$ )
23 else if  $\hat{e}_{t+1} < e_t$  then
24 for  $j \in E_{t+1}^{L(w)}$  do
25  $r = U([0, 1]);$  /* Generating a random float number  $r \in [0, 1]$  */
26 if  $r \leq \frac{e_t - \hat{e}_{t+1}}{e_t}$  then Delete Link( $j$ )
27 else
/* Do nothing */
28  $C_B(i, G') = \sum_{i, j, k \in G', i \neq j, k \neq i, j} \frac{P_k(i, j) P(i, j)}{(\hat{v}_{t+1}-1)(\hat{v}_{t+1}-2)/2}$ 
29 return  $C_B(i, G')$ 

```

graph snapshots in the model. The prediction for the next network at $(t + 1)$ is computed as:

$$\hat{G}_{t+1} = G_t G_{S(t)} \quad (3.6)$$

3.2.3 *Random Interaction Model*

The Random Interaction Model (RIM) detects the activity level of Agent i at a time $(t + 1)$ by computing his level of interaction with other agents in $G_{t+1}^{L(w)}$, the predicted network at the time $(t + 1)$. This network is simulated based on the numbers of agents and links predicted at time t using a single exponential smoothing method with $\alpha \in [0, 1]$ (lines 6-7 of Algorithm 1). As the simulated network varies from the observed network at time t , the model adds or deletes agents and links based on a uniform random distribution. The algorithm controls the network composition by using a random variable $r \in [0, 1]$ (lines 7-27 of Algorithm 1). The predicted level of activity of Agent i at the time $(t + 1)$ is then computed as his betweenness centrality in (see Section 3.2.5 for activity measurement).

3.2.4 *Preferential Interaction Model*

The Preferential Interaction Model (PIM) detects the activity of Agent i at a time $(t + 1)$ by computing his level of interaction with other agents in $G_{t+1}^{L(w)}$, the predicted network at the time $(t + 1)$. The structure of the simulated network at the time $(t + 1)$ follows that of the observed network at time t , plus adding or deleting agents and links based on an assumption of preferential attachment. PIM differs from RIM mainly by the use of the degree ratio p_j and removal likelihood q_i . The degree ratio measures the connectivity of an agent in the social network (lines 8 and 20 of Algorithm 2). Removal likelihood indicates the likelihood of an agent being removed (lines 13

Algorithm 2: Predicting the activity of Agent i with Preferential Interaction Model

Input : i, G, α, w, t
Output: Activity of Agent i at time $(t+1)$

```

1  $V_{t+1}^{L(w)}, E_{t+1}^{L(w)} = \{\}, \{\}; G' = \{V_{t+1}^{L(w)}, E_{t+1}^{L(w)}\}$  /* A graph with empty node and links */
2  $v_t, \hat{v}_{t+1} = |V_t^{L(w)}|, |V_{t+1}^{L(w)}|; e_t, \hat{e}_{t+1} = |E_t^{L(w)}|, |E_{t+1}^{L(w)}|$  /* The numbers of nodes and links */
3 for  $i \leftarrow V_t^{L(w)}$  do  $v_{t+1}^{L(w)} = v_{t+1}^{L(w)} \cup \text{Agent}(i)$ 
4 for  $i \leftarrow E_t^{L(w)}$  do  $E_{t+1}^{L(w)} = E_{t+1}^{L(w)} \cup \text{Link}(i)$ 
5  $\hat{v}_{t+1} = \alpha v_t + (1 - \alpha) \hat{v}_t; \hat{e}_{t+1} = \alpha e_t + (1 - \alpha) \hat{e}_t$  /*  $\hat{v}_2 = v_1$  and  $\hat{e}_2 = e_1$  */
6 if  $\hat{v}_{t+1} > v_t$  then
7    $u = \text{SortByAscendingDegreeRatio}(\text{Agent} \leftarrow V_t^{L(w)})$ 
8   for  $i = v_t + 1$  to  $\hat{v}_{t+1}$  do
9      $v_{t+1}^{L(w)} = v_{t+1}^{L(w)} \cup \text{Agent}(i)$ 
10    for  $j \in V_{t+1}^{L(w)}$  do
11       $r = U([0, 1]); p_j = \frac{\text{Deg}(j)}{\max(\text{Deg}(V_{t+1}^{L(w)}))}$  /* Degree ratio of j */
12      if  $r \leq p_j$  then
13         $E_{t+1}^{L(w)} = E_{t+1}^{L(w)} \cup \text{Link}(i, j)$ 
14 else if  $\hat{v}_{t+1} < v_t$  then
15    $c = 0$ 
16   for  $i \in V_{t+1}^{L(w)}$  do
17      $r = U([0, 1]); q_i = 1 - \frac{\text{Deg}(i)}{\max(\text{Deg}(V_{t+1}^{L(w)}))}$ 
18     if  $r \leq q_i$  &  $c < (v_t - \hat{v}_{t+1})$  then
19        $V_{t+1}^{L(w)} = V_{t+1}^{L(w)} - \{\text{Agent}(i)\}; E_{t+1}^{L(w)} = E_{t+1}^{L(w)} - \{\text{Links of Agent}(i)\}$ 
20        $c = c + 1$ 
21 else
22    $c = 0$ 
23   if  $\hat{e}_{t+1} > e_t$  then
24     for  $i, j \in V_{t+1}^{L(w)}$  where  $i \neq j$  do
25        $r = U([0, 1]); p_i = \frac{\text{Deg}(i)}{\max(\text{Deg}(V_{t+1}^{L(w)}))}; p_j = \frac{\text{Deg}(j)}{\max(\text{Deg}(V_{t+1}^{L(w)}))}$ 
26       if  $\{\text{Link}(i, j)\} \notin E_{t+1}^{L(w)}$  &  $r \leq \max(p_i, p_j)$  &  $c < (\hat{e}_{t+1} - e_t)$  then Add Link( $i, j$ )
27        $c = c + 1$ 
28     else if  $\hat{e}_{t+1} < e_t$  then
29       for  $i, j \in V_{t+1}^{L(w)}$  where  $i \neq j$  do
30          $r = U([0, 1]); p_i = 1 - \frac{\text{Deg}(i)}{\max(\text{Deg}(V_{t+1}^{L(w)}))}; p_j = 1 - \frac{\text{Deg}(j)}{\max(\text{Deg}(V_{t+1}^{L(w)}))}$ 
31         if  $\{\text{Link}(i, j)\} \in E_{t+1}^{L(w)}$  &  $r \leq \max(p_i, p_j)$  &  $c < (e_t - \hat{e}_{t+1})$  then Delete Link( $i, j$ )
32          $c = c + 1$ 
33     else
34       /* Do nothing */
35  $C_B(i, G') = \sum_{ijk \in G', i \neq j, k \neq i, j} \frac{P_k(i, j)/P(i, j)}{(\hat{v}_{t+1} - 1)(\hat{v}_{t+1} - 2)/2}$ 
return  $C_B(i, G')$ 

```

and 26 of Algorithm 2). Agents with high degree ratios are preferable to be connected with other agents. Agents with high removal likelihood are more likely to be removed. This process simulates a community in which users tend to attach to those who already have high connectivity. The predicted level of activity of Agent i at a time $(t + 1)$ is then computed as his betweenness centrality in $G_{t+1}^{L(W)}$. Different from prior work, PIM explicitly models the temporal process of link alteration and allows flexible simulation of future networks by various parameters.

3.2.5 Measurement of Agent Activity

There are many metrics to measure the activity level of an agent, such as degree, centrality, and clustering coefficient [149, 150]. In this research, I used Betweenness Centrality (BC) [148] to indicate an agent’s activity level, which reflects his influence in a networked community due to its emphasis on connectivity and agent interaction. The non-normalized BC of an Agent k is calculated as the proportion of the number of geodesics between any two distinct agents ($\neq k$; whose connecting paths contain k) to the total number of geodesics between any two distinct agents. The proposed models compute a normalized version of BC to be within $[0, 1]$ by dividing the non-normalized BC by the density of the network (line 28 of Algorithm 1; line 34 of Algorithm 2). Agents with a high BC score play an important role in bridging disparate members and transferring information or goods across different nodes. Different from other measures such as closeness centrality (which measures “time-until-arrival” in network flow), BC reflects the level of agent activity through interaction and influence in the network (similar to the notion of the frequency of arrival in network flow) [151].

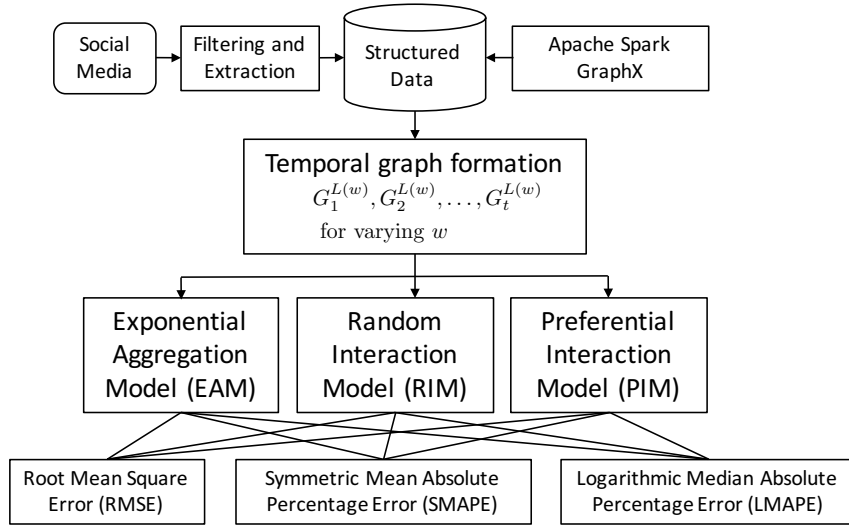




Figure 3.1: Data collection and experimental setup

3.3 Experiment Design

This section describes the design of a series of experiments to assess the models’ predictive power systematically under three temporal settings: (1) daily activity prediction over consecutive days, (2) varying sizes of reference windows w , and (3) different events that affect agent activities. The following sections describe the data used in the experiments, the experimental setup, and the performance evaluation measures (see Figure 3.1).

3.3.1 Data Description

The data used in the experiments were collected from a social media community formed by users who posted messages on the Twitter website. These messages were collected by using a set of carefully selected keywords (see Reference [36] for details of the selection) obtained from reviewing the literature on U.S. border security and immigration policy [152, 153, 154, 155]. Examples

	<p><i>RealDonaldTrump: "Amnesty lowers wages and invites more lawlessness. Obama has unilaterally cancelled any chance of immigration reform."</i></p>
	<p><i>TXOrganizingProject: "Today is Camino Americano, a rally in Washington D.C. demanding comprehensive immigration reform with a path to ..."</i></p>

of the queries include “immigration reform,” “US border security,” and “US immigration.” Between May 2013 and September 2015, the system collected over 3,286,473 tweets posted by over 790,462 users who formed more than 3,055,797 links. An undirected link is formed between two users (A and B) when User A sends a tweet targeted to User B, re-tweets another tweet written by User B, or modifies and then sends out a tweet written by User B [36]. These undirected links can represent the transparent and public nature of the links formed on Twitter. The links and the connected users are represented as edges and vertices, respectively, in an undirected graph G . A temporal graph $G_t^{L(w)}$ is formed by considering all the vertices and edges formed within the most recent w days at and prior to time t (see Section 3.1 for notation).

U.S. border security and immigration were chosen as the topic of study because of their timeliness and the dynamic networked community on social media. For example, these sample tweets reveal the diverse opinions expressed in the community:

Following Twitter’s guidelines of data collection and data privacy protection, the automated system continually collected on average 4,650 tweets per day using the selected queries. Through searching, retrieving, parsing, filtering, and storing, the system extracts temporal social network data (represented as temporal graphs) from the unstructured social media data. Figure 1 provides a high-level description of the data collection process and the experimental setup that was implemented in a distributed computing environment.

3.3.2 Datasets and Experimental Setup

Through a series of experiments, I tried to answer these research questions: Which models perform the best in temporal network activity detection in terms of different metrics and window sizes? What is the effect of window size on the performance? How do the structural properties of the temporal networks correlate with the performance? The experiments used two datasets covering two periods of time to represent two different sets of events:

- **Dataset 2013:** The dataset contains 316,686 messages posted by 111,212 users during Sept. 1–Oct. 31, 2013. This period was marked by the high fluctuation of public sentiment due to strong disappointment in the CIR bill impasse in the U.S. House of Representatives and a massive protest occurred on October 6, 2013, in Washington, DC, that led to the arrests of hundreds of people including prominent Congress members [156].
- **Dataset 2015:** The dataset contains 137,363 messages posted by 64,583 users during May 1–June 30, 2015. This period marked a turbulent time during which Donald Trump announced (on June 16, 2015) his candidacy for the U.S. presidency and made controversial remarks about border security in the U.S. Other candidates including Hillary Clinton and 16 Republican candidates announced their candidacies mostly in the same year. The analysis of this dataset can help to understand the effect of the U.S. presidential election on social media responses.

3.3.3 Experimental Setup

Two parameters in the models require tuning and selecting values: the window size (w) and the smoothing parameter (α). The value of w controls how many days of previous data were used to predict the value of the period's next day. The value of α (ranging from 0 to 1) determines

the weight assigned to primacy or recency effects in the prediction. Given the research focus on the predictive and simulation capabilities of the interaction models, a comprehensive study of all possible combinations of the parameters and their sensitivities is beyond the scope of this research.

- **Model Performance for Various Dates with a Specified w .** First, the four models (PIM, RIM, EAM, and TAM) were evaluated over a series of dates on which predictions were made based on a pre-defined w day of data. In the 61-day coverage of each dataset, I ran the models to predict once per three days. I also selected w to be 18 so the models had the same reference window to produce predictions. This value was set empirically to produce high efficiency and significance of predictions. Hence, the predictions were made for the activity levels on the 13th day of the first month, and then the 16th, 19th, 22nd, and so on. This procedure generated 17 sets of predicted values for the 2013 dataset and 16 sets for the 2015 dataset. Each set contains 12 (= 43) values that are the combinations of the four models and three metrics (RMSE, SMAPE, and LMAPE). For each model, I empirically determined the optimal smoothing constant α^* that enabled the model to achieve its best predictive performance within the period.
- **Effect of Temporal Window Size.** Second, the four models were evaluated over a set of different temporal window sizes w (*i.e.*, time spans in days), where $w = 4, 5, 6, \dots, 20$. Using these window sizes, the model performances for predicting each period's last activity levels (recorded on 10/30/2013 for Dataset 2013 and on 6/28/2015 for Dataset 2015) were aggregated and compared empirically to identify statistical differences.
- **Accuracy in Behavioral Role Discovery.** Third, the interaction models were compared against the Dynamic Behavioral Mix-membership Model (DBMM) [9] to evaluate the accuracy in discovering agents' behavioral roles. Each of the three models (DBMM, PIM, and RIM) was run, respectively, using the 2013 and 2015 datasets, each with two different win-

dow sizes ($w = 12$ and $w = 18$). To enable PIM and RIM to predict agent roles, the first two steps of DBMM (network feature extraction and probabilistic role discovery) were applied to predict node membership \hat{G}_{t+1} with the parameter values ($k = 10, \alpha = 0.7$) recommended in Reference [9].

- **Structural Analysis of Model Performances.** Fourth, I analyzed the relationship between the performance of the four models (PIM, RIM, EAM, and TAM) and the structural properties of the temporal social networks involved. Five structural properties were selected to provide information on the social networks: a total number of vertices (V), the total number of edges (E), network density (D), global clustering coefficient (C), and a number of connected components (M). Their formulas are available in Reference [63].

3.3.4 Evaluation Metrics

A measure that accounts for agent position in a network, the betweenness centrality (BC) score, represents the ability of an agent in bridging other users in the social media network. Due to its capability to model user interaction in social media networks, it is used as the ground truth to evaluate the models' performances (see Section 3.2.5 for further explanation). Based on this measure, four metrics were used to compare the interaction models and benchmarks.

- **Root Mean Square Error (RMSE).** Root Mean Square Error (RMSE) is one of the most frequently used measures of predictive performance. RMSE is calculated as the square root of the average of the squared differences between the actual values ($A_{t,i}$) and predicted values ($F_{t,i}$). n_t is the number of users in the network at time t . The RMSE formula is shown in Equation 3.7.

$$RMSE(t) = \sqrt{\frac{\sum_{i=1}^{n_t} (A_{t,i} - F_{t,i})^2}{n_t}} \quad (3.7)$$

- **Symmetric Mean Absolute Percentage Error (SMAPE).** Symmetric Mean Absolute Percentage Error (SMAPE) is a variant of Mean Absolute Percentage Error (MAPE), which expresses accuracy as a percentage value instead of absolute value to adjust for extreme values in the calculation. Different from MAPE, SMAPE uses the average of the absolute actual and forecast values as the denominator to further smooth the deviation between the two. Equation 3.8 shows the formula.

$$SMAPE(t) = \frac{100\%}{n_t} \sum_{i=1}^{n_t} \frac{|A_{t,i} - F_{t,i}|}{(|A_{t,i}| + |F_{t,i}|)/2} \quad (3.8)$$

- **Logarithmic Mean Absolute Percentage Error (LMAPE).** Research shows that MAPE and its variants (such as SMAPE) are biased toward predictions that are too low [157]. As the activity levels of temporal network agents can change dramatically, the predictive models may be required to produce extreme values that may create biases if MAPE is used. To address the problem, I developed a new measure called Logarithmic Mean Absolute Percentage Error (LMAPE; see Equation 3.9). LMAPE is designed to provide an unbiased evaluation of predictive models that may make extreme forecasts. The measure considers the absolute differences of the logarithm of actual and forecast values and divides the differences by the geometric means of the actual values. The use of geometric mean helps to alleviate the reliance on the arithmetic mean that is found to cause bias in evaluation [157]:

$$LMAPE(t) = \frac{100\%}{n_t} \sum_{i=1}^{n_t} \frac{|\ln(A_{t,i}) - \ln(F_{t,i})|}{(\prod_{i=1}^{n_t} A_{t,i})^{1/n_t}} \quad (3.9)$$

- **Frobenius Loss (FL).** Frobenius Loss (FL) can be used to reflect the approximation error between the estimated agent membership \hat{G}_{t+1} and the true agent membership \hat{G}_{t+1} when

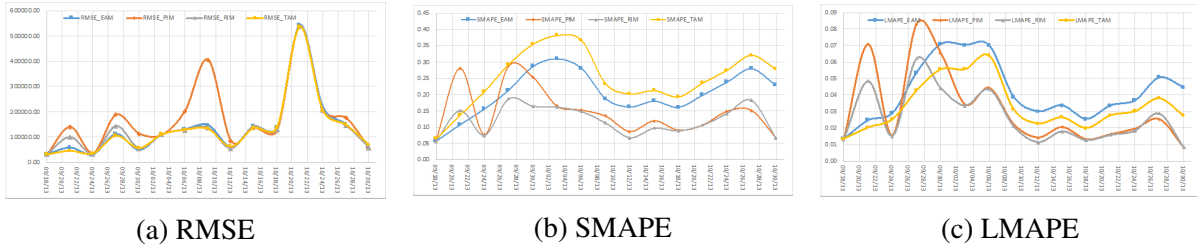


Figure 3.2: Model comparison across different dates for Dataset 2013

the models are used to predict agent roles. Equation 3.10 shows the computation of FL [9]:

$$FL(t + 1) = \|\mathbf{G}_{t+1} - \hat{\mathbf{G}}_{t+1}\|_F \quad (3.10)$$

3.4 Experimental Finding

This section reports the results of a series of experiments comparing the models using different time periods, window sizes, and performance measures. I provide statistical findings and charts to explain the performance and support the observations. I analyze the relationship between the temporal properties of the networks and the results, with a view to explaining the performance differences. To provide a further contextual explanation, I conducted case studies to show agents' interaction patterns within the temporal social media networks.

3.4.0.1 Model Comparison Across Different Dates

Figures 3.2 and 3.3 show comparisons of model performance by date across the two datasets (2013 and 2015) using three metrics (RMSE, SMAPE, and LMAPE; smaller values indicate better performance; $n = 15$ (number of data points) for Dataset 2013 (from 9/18/13 to 10/30/13 with a

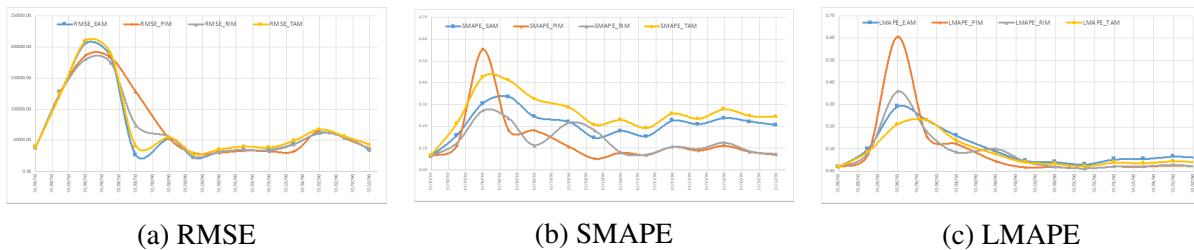


Figure 3.3: Model comparison across different dates for Dataset 2015

three-day increment), $n = 14$ for Dataset 2015 (from 5/18/15 to 6/27/15 with a three-day increment), and $w = 18$ for both datasets (this value of w was selected to provide the best performance within a dataset)). The performance comparison for both datasets indicates that PIM and RIM generally achieved better performance than EAM and TAM in terms of SMAPE and LMAPE (see Figure 3.2(b–c) and Figure 3.3(b–c)), whereas the performance of TAM and EAM was generally worse than the other models. Based on RMSE, the four models produced similar performance. Therefore, as shown in Figure 3.2a and Figure 3.3a, these differences are not clearly distinguished.

To identify the significance of the performance differences, I conducted statistical comparisons of model performances using the two datasets. The Welch two-sample pairwise t test was used, assuming unequal variances of the sample data. The results are shown in Table 3.2. Each pair of values embraced within parentheses refers to the performance values of two models being compared (e.g., the LMAPEs of EAM and RIM are 0.042 and 0.026, respectively, for Dataset 2013). Each value is the average of a model’s performance over all dates studied in a dataset. A lower value indicates better performance of the respective model (e.g., RIM in the aforementioned example) and is underlined if the performances of the models are significantly different (as shown in a significant p-value). Significant results are marked with one or more asterisks next to the p-values of the respective comparison (e.g., $p = 0.0024$ for EAM vs. RIM based on LMAPE).

Table 3.2: Performance and P-values of Model Comparison by Date

Models	RMSE	SMAPE	LMAPE
<i>Dataset 2013 Performance</i>			
EAM vs. RIM	(132744.58, 135077.55)	(0.2024, <u>0.1201</u>)	(0.0416, <u>0.0264</u>)
EAM vs. PIM	(132744.58, 170775.41)	(0.2024, <u>0.1445</u>)	(0.0416, <u>0.0310</u>)
PIM vs. RIM	(170775.41, 135077.55)	(0.1445, <u>0.1201</u>)	(0.0310, <u>0.0246</u>)
TAM vs. RIM	(130429.73, 135077.55)	(0.2495, <u>0.1201</u>)	(0.0416, <u>0.0264</u>)
TAM vs. PIM	(130429.73, 170775.41)	(0.2495, <u>0.1445</u>)	(0.0416, <u>0.0310</u>)
<i>Dataset 2013 p-value</i>			
EAM vs. RIM	0.5477	4.02028E-05***	0.0024**
EAM vs. PIM	0.0543 [^]	0.0219*	0.0848 [^]
PIM vs. RIM	0.0681 [^]	0.0531 [^]	0.0618 [^]
TAM vs. RIM	0.3233	2.95072E-06***	0.0706 [^]
TAM vs. PIM	0.0526 [^]	0.0007***	0.6694
<i>Dataset 2015 Performance</i>			
EAM vs. RIM	(68492.15, 69267.52)	(0.2075, <u>0.1320</u>)	(0.0953, <u>0.0734</u>)
EAM vs. PIM	(68492.15, 74151.85)	(0.2075, <u>0.1323</u>)	(0.0953, <u>0.0853</u>)
PIM vs. RIM	(74151.85, 69267.52)	(0.1323, <u>0.1320</u>)	(0.0853, <u>0.0734</u>)
TAM vs. RIM	(72699.44, 69267.52)	(0.2587, <u>0.1320</u>)	(0.0771, <u>0.0734</u>)
TAM vs. PIM	(72699.44, 74151.85)	(0.2587, <u>0.1323</u>)	(0.0771, <u>0.0853</u>)
<i>Dataset 2015 p-value</i>			
EAM vs. RIM	0.8515	0.0003***	0.0345*
EAM vs. PIM	0.4680	0.0161*	0.6986
PIM vs. RIM	0.2417	0.9906	0.5355
TAM vs. RIM	0.3486	9.82525E-06***	0.7787
TAM vs. PIM	0.8360	0.0069**	0.7887

Note: An underlined value indicates significantly better performance of a model in the pair. Asterisks indicate statistical significance: [^] p < 0.1; *p < 0.05; **p < 0.01; ***p < 0.001.

The results show that PIM and RIM outperformed both TAM and EAM significantly based on SMAPE (for both the 2013 and 2015 datasets) and on LMAPE (for five of eight cases in the two datasets). By contrast, the comparisons between RIM and PIM yielded diverse results. Using the 2013 dataset, RIM outperformed PIM significantly only at the $p < 0.1$ level (but not at $p < 0.05$ or finer levels). Using the 2015 dataset, the performances of the RIM and PIM are not significantly different based SMAPE and LMAPE. Because RMSE is suitable mainly for comparing linear models, performance differences among models that describe both exponential and network effects

(*e.g.*, RIM and PIM) may not be captured precisely by RMSE. Thus, many of the differences in RMSE are not significant.

I observe a small number of spikes in errors on certain dates (*e.g.*, 9/27/2013, 10/21/2013 in Figure 3.2a). These spikes may be caused by computational operations in specific metrics and by the interaction models' adjustment in nodes and links. First, the squaring operation in RMSE may contribute to the spikes shown in Figure 3.2a. For instance, on 10/21/2013 shown in the figure, the social media network consists of 49,725 vertices and 120,295 edges, which makes it one of the largest networks in the 2013 dataset. Together with the randomness of user behavior, the size and complexity of networks contribute to amplifying errors that are computed through the squaring operation in the RMSE metrics. By contrast, such amplification is not as prominent in Figure 3.2b and in Figure 3.2c, which do not use squaring in SMAPE and LMAPE, respectively. Second, the node and link adjustment processes used in PIM and RIM may contribute to the spikes shown in Figure 3.2 and Figure 3.3. For instance, for the spikes shown on 9/21/2013, 9/27/2013, and 5/24/2015 in Figure 3.2(b–c) and Figure 3.3(a–c), all these dates have bursty upward trends of users and links, for which RIM and PIM are required to create new agents and links in the simulated networks at $t + 1$ (see Algorithms 1 and 2). These data might have contributed to larger-than-required predictions (and thus the spikes) of the interaction models. Table 3.9 provides additional information about the structural information of the networks on these dates. By contrast, EAM assumes nodes to be static and does not have a similar dynamic adjustment process that PIM and RIM have. As a result, EAM tends to make relatively smooth predictions, producing fewer spikes in errors. Future studies may explore the mechanism to adjust nodes and links to increase prediction accuracy.

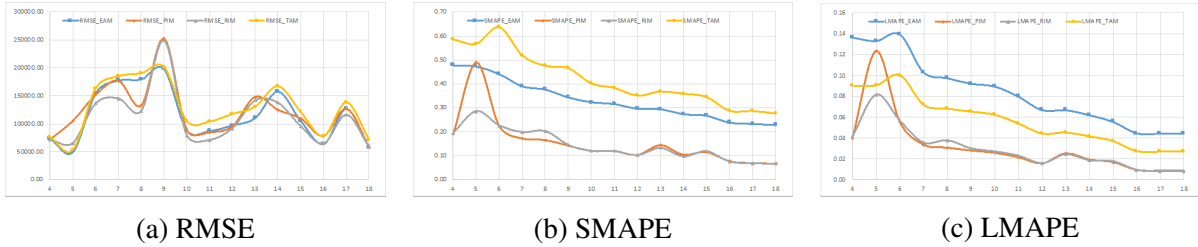


Figure 3.4: Model comparison across different window sizes for Dataset 2013

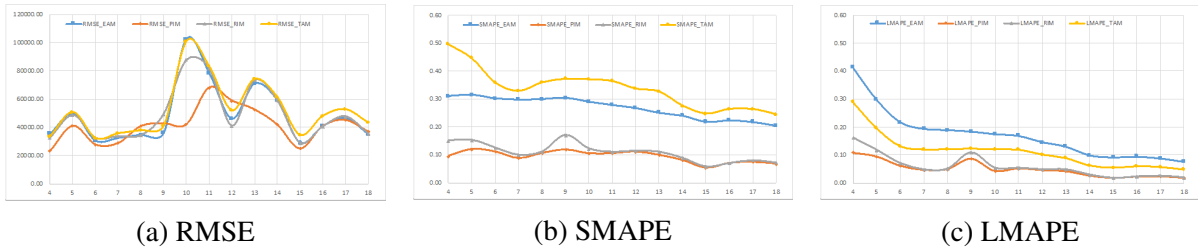


Figure 3.5: Model comparison across different window sizes for Dataset 2015

3.4.0.2 Model Comparison Across Different Window Sizes

Figures 3.4 and 3.4 show comparisons of model performances by window size across the two datasets using three metrics (RMSE, SMAPE, and LMAPE). For each window size (ranging from 4 to 20 with an increment of 1), the best alpha (that resulted in the highest performance values) was used for each model in its running. The optimal alpha value is identified for each experimental configuration (with one model, one window size, one prediction date, and one metric) by running a given model five times using five alpha values: $[0.1, 0.3, 0.5, 0.7, 0.9]$, and then choosing the alpha that yielded the best performance (lowest error) in that configuration. The prediction periods are the same as those listed in the previous section.

The performance comparison for both datasets (Figure 3.4(b–c) and Figure 3.5(b–c)) indicates

Table 3.3: Performance and P-values of Model Comparison by w

Models	RMSE	SMAPE	LMAPE
<i>Dataset 2013 Performance</i>			
EAM vs. RIM	(115269.73, <u>110059.43</u>)	(0.3318, <u>0.1439</u>)	(0.0839, <u>0.0290</u>)
EAM vs. PIM	(115269.73, <u>120501.04</u>)	(0.3318, <u>0.1538</u>)	(0.0839, <u>0.0307</u>)
PIM vs. RIM	(<u>120501.04</u> , <u>110059.43</u>)	(0.1538, <u>0.1439</u>)	(0.0307, <u>0.0290</u>)
TAM vs. RIM	(126938.53, <u>110059.43</u>)	(0.4208, <u>0.1439</u>)	(0.0569, <u>0.0290</u>)
TAM vs. PIM	(126938.53, <u>120501.04</u>)	(0.4208, <u>0.1538</u>)	(0.0569, <u>0.0307</u>)
<i>Dataset 2013 p-value</i>			
EAM vs. RIM	0.4397	2.89204E-12***	9.71997E-09***
EAM vs. PIM	0.4735	3.80808E-08***	2.27542E-07***
PIM vs. RIM	0.0062**	0.4936	0.5619
TAM vs. RIM	0.0280*	6.1488E-11***	1.05328E-07***
TAM vs. PIM	0.4082	5.88899E-09***	0.0001***
<i>Dataset 2015 Performance</i>			
EAM vs. RIM	(47690.12, <u>48065.58</u>)	(0.2600, <u>0.1062</u>)	(0.1589, <u>0.0542</u>)
EAM vs. PIM	(47690.12, <u>39917.28</u>)	(0.2600, <u>0.0913</u>)	(0.1589, <u>0.0454</u>)
PIM vs. RIM	(<u>39917.28</u> , <u>48065.58</u>)	(0.0913, <u>0.1062</u>)	(0.0454, <u>0.0542</u>)
TAM vs. RIM	(51531.55, <u>48065.58</u>)	(0.3242, <u>0.1062</u>)	(0.1049, <u>0.0542</u>)
TAM vs. PIM	(51531.55, <u>39917.28</u>)	(0.3242, <u>0.0913</u>)	(0.1049, <u>0.0454</u>)
<i>Dataset 2015 p-value</i>			
EAM vs. RIM	0.7799	5.82636E-15***	6.98436E-07***
EAM vs. PIM	0.0622 [^]	2.51315E-14***	2.43422E-06***
PIM vs. RIM	0.0238*	0.0017**	0.0178*
TAM vs. RIM	0.0090**	5.79323E-12***	1.57455E-06***
TAM vs. PIM	0.0045**	3.85823E-11***	9.45915E-06***

Note: An underlined value indicates significantly better performance of a model in the pair. Asterisks indicate statistical significance: [^]p < 0.1; *p < 0.05; **p < 0.01; ***p < 0.001.

that, in terms of SMAPE and LMAPE, PIM and RIM outperformed EAM and TAM in almost all window sizes. Based on RMSE, the four models produced similar performances (Figure 3.4a and Figure 3.5a). A possible reason is that RMSE fails to explain the network and exponential effects of PIM and RIM and treated all four models similarly.

Table 3.3 shows the results of statistical pairwise comparisons of the model performances. As in Table 3.2, each pair of values embraced within parentheses refers to the performance values of two

models being compared. Each value is the average of a model’s performance for predicting the last activity levels in each dataset (activity levels on 10/30/2013 for Dataset 2013 and on 6/28/2015 for Dataset 2015) using different window sizes ($w = 4, 5, 6, \dots, 20$). A lower value indicates better performance of the respective model.

The results show that both RIM and PIM outperformed EAM and TAM significantly based on both SMAPE and LMAPE, while no significant difference in model performances was found based on RMSE. Thus, H3 was rejected in eight of the twelve cases. Using the 2013 dataset, RIM outperformed PIM significantly based on RMSE. Using the 2015 dataset, PIM outperformed RIM significantly based on all three metrics (RMSE, SMAPE, and LMAPE). Due to inconsistent results found in two datasets, additional analyses of the environments and structural properties are needed to explain these findings.

3.4.1 Model Comparison Using Frobenius Loss

As shown in Table 3.4, the interaction models achieved on average a lower Frobenius Loss (FL) than DBMM did in all the eight comparisons in both the 2013 and 2015 datasets, indicating a generally better capability in agents’ behavioral role discovery. The two interaction models significantly outperformed DBMM in all four comparisons ($p < 0.001$) in the 2015 dataset. PIM outperformed DBMM significantly ($p = 0.01576$) in the 2013 dataset ($w = 12, n = 40$), and RIM obtained a better FL than DBMM did with moderate p values (0.1005 and 0.1551 for $w = 12$ and $w = 18$, respectively). One reason for the superior performance of the interaction models is their realistic adjustment of node composition in evolving networks. This is especially true in online social media in which the composition of social networks can change dramatically over a short time. By contrast, DBMM assumes nodes to be static and their composition unchanged over time. Although DBMM estimates the dynamic behavior of nodes, the absence of change in nodal com-

Table 3.4: Performance and P-values of Model Comparison by Frobenius Loss

Parameter	DBMM vs. PIM	DBMM vs. RIM	RIM vs. PIM
<i>Dataset 2013 Performance</i>			
w=12, n=40	(79.42, <u>72.17</u>)	(79.42, <u>74.30</u>)	(74.30, 72.17)
w=18, n=34	(100.69, 99.92)	(100.69, <u>95.62</u>)	(95.62, 99.92)
<i>Dataset 2013 p-value</i>			
w=12, n=40	0.01576 *	0.1005 [^]	0.2673
w=18, n=34	0.795	0.1551 [^]	0.3461
<i>Dataset 2015 Performance</i>			
w=12, n=40	(49.25, <u>42.34</u>)	(49.25, <u>42.97</u>)	(42.97, 42.34)
w=18, n=34	(56.07, <u>47.36</u>)	(56.07, <u>48.78</u>)	(48.78, 47.36)
<i>Dataset 2015 p-value</i>			
w=12, n=40	9.993e-06 ***	3.262e-05 ***	0.668
w=18, n=34	1.906e-04 ***	4.796e-04 ***	0.4458

Note: An underlined value indicates significantly better performance of a model in the pair. Asterisks indicate statistical significance: [^]p < 0.2; * p < 0.05; ** p < 0.01; *** p < 0.001. w is the window size used in the models and n is the sample size (number of days' data).

position in evolving networks makes the role estimation less effective and unrealistic. In addition, the rank-r approximation used in DBMM makes another static assumption of roles and nodal features. This is in contrast with the more flexible prediction of agents and links in PIM and RIM based on relevant social theories that can be used to explain dynamic human behavior. However, the only comparison showing no significant difference between DBMM and PIM is when $w = 18$ in the 2013 dataset ($p = 0.795$). It should be noted that the parameter value $\alpha = 0.7$ used in PIM and RIM (as recommended in Reference [9]) is not empirically tested to be optimal for the two interaction models. Future studies may explore the effect of parameter values (w and α) on agent role discovery.

Table 3.5: Pearson Correlations (and P-values) between Structural Properties and Model Performances for Datasets 2013 and 2015

Correlation for Dataset 2013

	RMSE_EAM	RMSE_TAM	RMSE_PIM	RMSE_RIM	SMAPE_EAM	SMAPE_TAM	SMAPE_PIM	SMAPE_RIM	LMAPE_EAM	LMAPE_TAM	LMAPE_PIM	LMAPE_RIM
NumVertices_13	0.251983	0.246338	0.232552	0.210932	-0.227823	-0.365386	-0.481636	-0.440020	-0.600474	-0.718916	-0.677971	-0.735199
NumEdges_13	0.171389	0.167132	0.163567	0.130969	-0.340960	-0.474337	-0.500433	-0.466678	-0.694097	-0.804960	-0.693455	-0.743010
Density_13	-0.378742	-0.373043	-0.354124	-0.352905	0.282294	0.396002	0.435608	0.397663	0.588011	0.696339	0.604860	0.645586
Clustering_13	-0.262539	-0.252270	-0.219916	-0.252480	0.245832	0.390168	0.604349	0.471853	0.606045	0.714359	0.769093	0.746683
ConnectedComp	0.338293	0.328962	0.323361	0.314009	-0.503724	-0.625061	-0.459404	-0.441456	-0.758702	-0.804753	-0.626318	-0.641852

p-value of cor.test for Dataset 2013

	RMSE_EAM	RMSE_TAM	RMSE_PIM	RMSE_RIM	SMAPE_EAM	SMAPE_TAM	SMAPE_PIM	SMAPE_RIM	LMAPE_EAM	LMAPE_TAM	LMAPE_PIM	LMAPE_RIM
NumVertices_13	0.329222	0.340533	0.369076	0.416416	0.379165	0.149245	0.050278	0.077141	0.010811	0.001146	0.002781	0.000772
NumEdges_13	0.510717	0.521427	0.530474	0.616348	0.180478	0.054388	0.040766	0.058965	0.001994	0.000097	0.002022	0.000632
Density_13	0.133833	0.140268	0.163148	0.164704	0.272296	0.115598	0.080504	0.113941	0.013044	0.001901	0.010101	0.005125
Clustering_13	0.308665	0.328654	0.396368	0.328238	0.341557	0.121553	0.010182	0.055843	0.009916	0.001274	0.000308	0.000574
ConnectedComp	0.184133	0.197302	0.205497	0.219666	0.039253	0.007297	0.063571	0.076069	0.000414	0.000098	0.007146	0.005475

Correlation for Dataset 2015

	RMSE_EAM	RMSE_TAM	RMSE_PIM	RMSE_RIM	SMAPE_EAM	SMAPE_TAM	SMAPE_PIM	SMAPE_RIM	LMAPE_EAM	LMAPE_TAM	LMAPE_PIM	LMAPE_RIM
NumVertices_15	0.3061	0.2013	0.3585	0.2887	0.6148	-0.6512	-0.0520	-0.0679	-0.7613	-0.7454	-0.7570	-0.8160
NumEdges_15	0.2561	0.1536	0.3082	0.2456	0.6053	-0.6655	-0.0990	-0.1153	-0.7724	-0.7455	-0.7877	-0.8307
Density_15	-0.4803	-0.3522	-0.3292	-0.3278	-0.6297	0.5187	0.0280	-0.0819	0.8269	0.9582	0.7272	0.8687
Clustering_15	0.0751	0.0201	-0.0957	0.0144	0.6037	-0.0017	0.0603	-0.0788	-0.5306	-0.5370	-0.4478	-0.5978
ConnectedComp	0.4658	0.3028	0.5191	0.4678	0.4939	-0.6499	0.1505	-0.0033	-0.7516	-0.7515	-0.6237	-0.7848

p-value of cor.test for Dataset 2015

	RMSE_EAM	RMSE_TAM	RMSE_PIM	RMSE_RIM	SMAPE_EAM	SMAPE_TAM	SMAPE_PIM	SMAPE_RIM	LMAPE_EAM	LMAPE_TAM	LMAPE_PIM	LMAPE_RIM
NumVertices_15	0.2489	0.4548	0.1727	0.2782	0.0113	0.0063	0.8483	0.8028	0.0006	0.0009	0.0007	0.0001
NumEdges_15	0.3383	0.5701	0.2456	0.3591	0.0130	0.0049	0.7154	0.6706	0.0005	0.0009	0.0003	0.0001
Density_15	0.0597	0.1809	0.2132	0.2152	0.0089	0.0395	0.9179	0.7631	0.0001	0.0000	0.0014	0.0000
Clustering_15	0.7823	0.9411	0.7246	0.9579	0.0133	0.9949	0.8243	0.7716	0.0345	0.0320	0.0820	0.0145
ConnectedComp	0.0690	0.2544	0.0393	0.0677	0.0518	0.0064	0.5780	0.9903	0.0008	0.0008	0.0098	0.0003

3.4.2 Structural Analysis of Social Networks

To understand the relationship between the model performance and the structure of the social networks in each dataset, I conducted a correlation analysis to identify the degree to which structural properties relate to performances. Table 3.5 shows the correlations (and p-values of testing their significance) between each pair of structural properties and model performances for Datasets 2013 and 2015. Shaded cells in the first and third tables highlight correlations with absolute values above 0.4 (light shade) or 0.7 (dark shade). Shaded cells in the second and fourth tables highlight p-values below 0.05 (light shade) or 0.01 (dark shade).

The results show significant correlations between LMAPE (of all models) and the five structural properties. For Dataset 2013, all 20 cases reported significant p-values, and all the absolute values of correlations are above 0.5, with the highest being 0.805 (LMAPE of TAM and clustering

coefficient). For Dataset 2015, 14 out of 15 cases reported significant p-values, and all the absolute values of the significant correlations are above 0.5, with the highest being 0.958 (LMAPE of RIM and density). The p-values in 10 of 15 cases in Dataset 2015 are less than 0.001, indicating extremely significant correlations between LMAPE of the four models and structural properties. Therefore, the findings reveal strong predictability of structural properties on LMAPE of all models in both datasets.

The results also show high to moderate correlations between SMAPE (of EAM, TAM, and RIM) and the structural properties. For Dataset 2013, 4 out of 20 cases reported significant p-values, and all these cases have absolute correlations above 0.5. For Dataset 2015, 8 cases (all involving EAM or TAM) reported significant p-values while 10 cases (involving PIM and RIM) have insignificant p-values. The findings indicate that structural properties have moderately strong predictability on SMAPE of EAM in Dataset 2015. But for RIM, the predictability is moderately strong in Dataset 2013 but is weak in Dataset 2015. The predictability for SMAPE of PIM is weak in Dataset 2015. The structural properties have weak and insignificant correlations (<0.4 in absolute value) with RMSE of the four models in most cases. All p-values are not significant in Dataset 2013 and in Dataset 2015, with the only exception being the correlation between the number of connected components and RMSE of PIM in Dataset 2015 ($p = 0.03934$, correlation = 0.5191). Therefore, the results show that structural properties are poor predictors of RMSE in all models.

3.4.3 Case Studies of User Interaction Patterns

To enable contextual reasoning and explanation of the model's predictions, I conducted case studies of the top 20 users (who have the highest activity scores) and all users and provided their interaction patterns to show different results of RIM and PIM compared with the two benchmark models. The case studies examine three dates of user interaction patterns in each of the two datasets

Table 3.6: Percentage of Overlap between ACTUAL Users and Users Predicted by Model

Model	EAM		PIM		RIM		TAM	
Date	Top20	All	Top20	All	Top20	All	Top20	All
10/3/13	55%	93.60%	50%	93.641%*	70%*	93.60%	30%	93.60%
10/6/13	65%*	97.27%	55%	97.281%*	65%*	97.27%	60%	97.27%
10/9/13	80%	99.59%	70%	99.678%*	90%*	99.58%	80%	99.59%
6/15/15	15%	96.078%*	80%*	95.98%	75%	95.99%	80%*	96.078%*
6/18/15	30%	96.79%	75%	97.05%*	80%*	96.80%	70%	96.79%
6/21/15	50%	95.792%*	75%*	95.792%*	75%*	95.792%*	65%	95.792%*

*indicates highest percentage overlap across all models in either "Top20" or "All" category.

(2013 and 2015). The three dates chosen from the 2013 dataset were 10/3/2013, 10/9/2013, and 10/9/2013, during which a massive protest took place in Washington, D.C. in support of comprehensive immigration reform in the United States. This event resulted in over 200 people arrested by police, among them 8 Democratic members of the U.S. Congress. The three dates chosen from the 2015 dataset were 6/15/2015, 6/18/2015, and 6/21/2015, during (and before) which prominent members of the Republican Party, Donald Trump and Jeb Bush (among others), both announced their candidacies for the U.S. presidency (on 6/15/2015), triggering many reactions in social media.

First, for each model, I computed the percentage of overlap between the actual users (in the Top20 or All lists) and the model's predicted users in the same two lists. The results, shown in Table 3.6, reveal that among all four models, RIM achieved (on five out of six days) the best matching (marked with *) between the actual top-20 users and the predicted top-20 users; PIM is ranked second (two out of six days). When considering all users, PIM achieved (on five out of six days) the best matching between actual users and predicted users. These results confirm the superior performance of PIM and RIM in predicting nodal activities in the social media networks of the selected dates.

Second, I selected examples of top-20 users to examine cases where all models predicted them successfully (*i.e.*, matching actual top-20 users) in contrast with cases where only RIM and PIM

Table 3.7: Interaction Patterns of Users Predicted by All Four Models to be among Top20

Date	Public User Name	Message Type					Follower Count	Friend Count
		Target Tweet	Retweet	Modified Tweet	Self retweet	Duplicate Tweet		
10/3/13	Immigraxion	2	569	34	0	2	323	180
10/6/13	SpeakerBoehner	285	698	19	4	112	803,118	7,704
10/9/13	SteveWorks4You	61	1,765	15	0	1	19,320	4,857
6/15/15	AnnCoulter	136	706	2	4	1	611,756	387
6/18/15	JebBush	157	225	2	5	16	130,736	111
6/21/15	HillaryClinton	79	577	2	1	9	3,499,414	16

predicted them successfully. Interaction patterns of these two groups of users can be used to distinguish the predictive capabilities of the models. As shown in Table 3.7, the users predicted by all four models to be among Top20 generally have significantly more retweets, more self retweets, and more duplicate tweets. By contrast, users predicted by only PIM and RIM (see Table 9) generally have fewer targeted tweets and fewer modified tweets. These message types may be appropriately captured by PIM and RIM due to their unique nature in identifying interactive users. Although the user “Hillary Clinton” appears in both tables, the dates that the user appears are different with different interaction patterns: 6/21/2015 in Table 8 and 6/15/2015 in Table 3.8. It is possible that PIM and RIM capture interaction patterns of the user better on 6/15/2015 due to their capabilities of detecting nodal activities better in an environment with highly opposing views (two Republican candidates announced candidacies on that date, whereas “HillaryClinton” belongs to the Democratic Party).

3.4.4 Implications

The experimental findings provide insights into understanding the model performances, temporal features, structural properties of networks, user interaction patterns, and attributes of datasets. Their implications for predicting dynamic social network activities are discussed below. First,

Table 3.8: Interaction Patterns of Users Predicted by Only PIM and RIM to be among Top20

Date	Public User Name	Message Type					Follower Count	Friend Count
		Target Tweet	Retweet	Modified Tweet	Self retweet	Duplicate Tweet		
10/3/13	politico	93	190	0	0	1	1,185,623	999
10/6/13	nytimes	67	353	4	0	2	17,324,215	991
10/9/13	FAIRImmigration	52	487	160	0	12	7,821	2,869
6/15/15	HillaryClinton	77	474	2	1	6	3,499,414	16
6/18/15	realDonaldTrump	129	71	0	1	0	3,071,548	46
6/21/15	ianbremmer	3	185	0	0	0	110,509	988

Table 3.9: Structural Information of Social Media Networks on Selected Dates

Date	Max(degree)	Min(degree)	Mean(degree)	NumVertices	NumEdges	Density	ClusteringCoeff	#ConnectedComponents
9/21/13	4,164	1	5	28,823	84,611	2.04E-04	1.20E-02	2,328
9/27/13	3,706	1	5	24,126	64,100	2.20E-04	1.20E-02	2,004
5/24/15	1,316	1	2	11,581	17,298	2.58E-04	5.00E-03	1,307

the proposed interaction models (RIM and PIM) performed significantly better than the benchmark models (EAM, TAM, and DBMM) across datasets of different time frames and content, indicating the models' strong capability of accounting for user interactions in activity prediction across different contexts. This result confirms that the proposed models can be useful for detecting activity levels in large dynamic online social networks. Second, the time window size (w) (the length of history that the models used to predict activity) was found to demonstrate the performance differences between the interaction models and the benchmark models. Both RIM and PIM outperformed TAM and EAM across different w . The result implies that user interaction is captured sufficiently by the two proposed models consistently, despite changes in w . Third, PIM outperformed RIM across different w in Dataset 2015, indicating that the performance of PIM is less affected by changing w . It follows that when the length of prior history is relatively short ($4 \leq w \leq 20$), PIM would be a preferred model for detecting activity levels of social network users. For longer prior histories ($w > 20$), additional data and studies will be needed to under-

stand performance differences. Fourth, both PIM and RIM outperformed DBMM significantly in terms of agents' behavioral role discovery, indicating their superior performance due to a realistic treatment of dynamic network composition. Fifth, structural properties of the social networks correlated significantly with LMAPE of all four models and with SMAPE of EAM. The result shows that macroscopic, structural factors of networks have strong to moderately strong predictability of model performances (as measured by LMAPE and SMAPE respectively). This allows the assessment of temporal network activity detection models with relative ease of computation and yet with high accuracy. Furthermore, interaction patterns of selected top users reveal that PIM and RIM provide superior predictive capabilities, especially in social media networks composed of highly opposing views.

CHAPTER 4: SEMANTICS-AWARE OPTIMIZATIONS FOR BIG DATA APPLICATIONS

4.1 Introduction and Motivation

Recently, a growing number of data-intensive computing frameworks have been proposed, such as MapReduce [11], Apache Hadoop [12], and Spark [13] to process a large volume of data in a parallel and robust manner within a reasonable time [85, 86]. The successes of these frameworks are due to their MapReduce-like programming models, which are further based on data distribution techniques (*e.g.*, Resilient Distributed Dataset (RDD) in Apache Spark [87]), and high-order functions (*e.g.*, *map*, *reduce*, *filter*) that can take user-defined functions (UDFs) as arguments. The semantics of these high-order functions facilitate data-parallelism to manipulate datasets in an element-wise way while UDFs are applied to each element to produce the desired result. The programming models hide the details of scheduling, load balancing, execution, and communication from programmers, which eases programming and allows programmers to focus on data flow and UDF designs. Despite these advantages, an endeavor to improve the performance of data-intensive applications exhibits a few challenging issues

- Usually, unstructured data expose less information about their schema if without metadata or annotation provided by programmers or help from runtime profiling tools.
- It is difficult to apply conventional database-style optimizations on unstructured data directly, such as relational algebraic reordering and filter pushdown since the programming models of current data-intensive computing platforms lack information about data schema [84].
- Although Spark can process raw unstructured data directly using DataFrame or Dataset APIs,

it needs to parse them before performing transformations (e.g, Map) and actions (Reduce-ByKey). Particularly, these applications can spend 80-90% of the entire executing time in data parsing [158].

- Programming models usually treat UDFs as black boxes and their semantics are therefore hidden from the system, resulting in insufficient information for further optimization [90, 159, 94, 160].
- Runtime factors are not fully utilized to tune the performance of a specific operation's execution, such as cache management [106, 105].

Therefore, it is vital to integrate program semantics, data property, and runtime factors to improve the performance of data-intensive applications since pure static optimizations are either limited or impossible without efficient profiling information about data and runtime systems. In this paper, SODA is proposed as a two-phase framework, *i.e.*, offline static analysis and online dynamic analysis, to interactively and semi-automatically assist programmers to scrutinize performance problems camouflaged in source code.

4.2 System Overview

4.2.1 Architecture

The framework of SODA includes offline and online phases, as shown in Figure 4.1. The offline (static) phase is developed as a compiler plugin of host programming languages (*i.e.*, Scala, Java), and analyzes source code (src) and performance log about data and runtime factors to generate a nearly-optimized and parameterized program. Firstly, *Code Analyzer* analyzes source code with the help of a local compiler to construct a directed data operation graph (DOG), which rep-

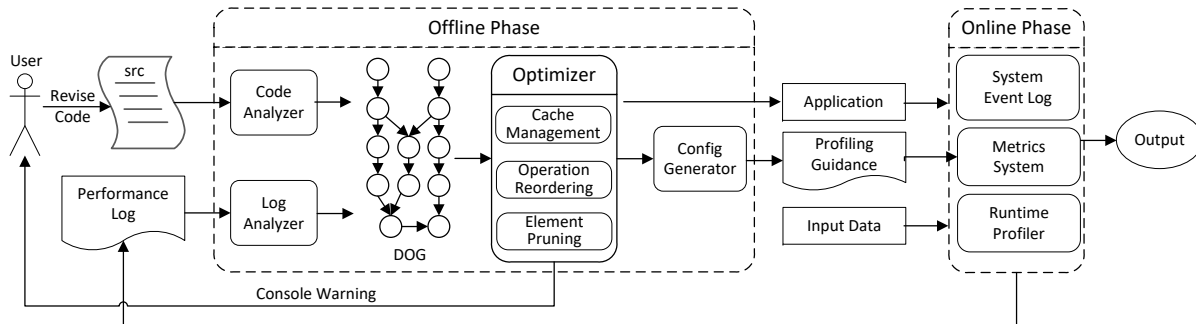


Figure 4.1: The full life cycle of Semantics-Aware Optimization Approach for Data-Intensive Applications (SODA).

resents the skeleton of an application. This graph comprises a set of nodes and edges, which denote operations and dataflows, respectively. In addition to static properties associated with corresponding operations, a group of dynamic profiling data is extracted by *Log Analyzer* from the performance log, which is accumulated in prior executions, including execution time, memory usage, input, and output data size of operations, runtime system status *etc.* This information can be extracted from system log [161, 162, 163] and provided by the profiling tool using Javassist (Java Programming Assistant), a high-level bytecode instrumentation tool to instrument APIs of Spark to expose information needed [164]. Next, three optimization strategies, *i.e.* cache management, operation reordering, and element pruning, are applied to assist users to scrutinize performance problems. When a problem is found, users would get informed about performance bugs from SODA and then refactor code. However, not all problems can be determined statically, it may need more information coming from executions. For example, SODA makes use of execution time and output size of operations to verify performance behavior and then create a global cache allocation strategy. To reduce system overhead resulting from the profiling process, *Config Generator* produces *Profiling Guidance* to inform the online phase about which operations and what kinds of computational resources need to be monitored. In the online phase, SODA initializes an application with a parameterized configuration based on *Profiling Guidance* and starts a piggyback

listener residing in each worker and master node to collect runtime information about memory usage, data property, and system configuration. The profiling data would be accumulated and then delivered back as a performance log to the offline phase for further optimizations.

Furthermore, SODA is implemented on Apache Spark, and several real-world Spark applications are used as benchmarks to evaluate its effectiveness. Apache Spark [87] is an efficient and general engine for large-scale data processing that supports the scalability of MapReduce [11]. Its main abstraction, named Resilient Distributed Dataset (RDD) [87], is a fault-tolerant and immutable collection of objects, which are logically partitioned across a cluster of computing nodes so that they can be manipulated in parallel. Spark’s programming model provides two types of operations, *transformation* and *action*. A *transformation* creates a new RDD dataset from an existing one while an *action* returns a value to the driver program. The lazy feature of transformations enables Spark to run more efficiently since they do not compute their results immediately until action is invoked. An RDD has to be recomputed when invoking an action on it unless it is persisted in memory using the *persist (or cache)* method, which facilitates much faster access. Apache Spark automatically monitors cache usage on each node and drops out old data partitions in an LRU fashion by default. In the Spark execution model, a Spark application is divided into a group of jobs executed in a sequential order¹, where a job is a parallel computation in response to a Spark action (*e.g., save, collect*); Within a *job*, multiple *stages* are generated and bounded by shuffle behaviors (*e.g., reduce*), then run in parallel if there is no data dependency among them; otherwise, they are scheduled sequentially. Internally, a stage is a physical execution unit consisting of several operations. The unit is further divided into tasks, which share identical code but run on different data partitions in parallel. Given that, I need a fine-grained profiling tool to analyze semantic properties in code as well as runtime factors, such as the evolution of data, the execution time of operations, and system status, to narrow down the gap between the programming model and

¹Multiple jobs can run simultaneously if they were submitted from separate threads

execution model).

4.2.2 Performance Problems

SODA looks for three kinds of performance problems: Cache Management (CM), Operation Re-ordering (OR), and Element Pruning (EP).

Cache Management (CM): It is crucial to managing cache resources for these data analytics frameworks [103, 165, 166, 13], which leverage in-memory computing to speed up performance and bypass the hindrance of disk and network I/O. Within these systems, intermediate computing data block would be put in memory by default. There is a block management component to manage these blocks and determine when and which one is evicted from memory. Recently, a rich line of research work proposes different data block reference measurements to improve cache hit, such as least recently used (LRU), least reference count (LRC) [105] and most reference distance (MRD) [106]. However, there remain two important factors that previous works have not taken into account, especially in Spark.

1. The executing order of all stages. This could impact system performance, especially for cache behaviors.
2. Data block size. Data blocks with the same reference in LRU or other fancy measurements, might not all fit the memory at the same time, and therefore it raises the concern about cache priority with regard to system performance.

In addition, programmers may brutally persist the desired dataset in memory by invoking corresponding APIs (*i.e.*, using the `persist` (or `cache`) method in Spark), resulting in a more complicated research problem. Therefore, an intelligent cache management mechanism using hybrid

program analysis is needed to manage memory for efficiency. In this paper, I propose a stage-level cache allocation strategy in a data-intensive system by reducing it to a convex optimization problem [167, 168, 169].

Operation Reordering(OR): A data-intensive system usually supports a rich line of operations, such as `map`, `reduce`, `filter`, `reduceByKey`, and `join`. A developer may face a variety of executing plans assembled by a sequence of operations associated with UDFs to accomplish an application. Nonetheless, not all of these arrangements will yield identical performance. Therefore, it is crucial to orchestrate the operations in an appropriate order to bypass common pitfalls affecting performance significantly. For example, filter pushdown and join reorder are two common optimization strategies to improve the performance of relational algebra-based systems when handling structured data. As to processing unstructured datasets, however, it is difficult to apply these conventional database-style techniques to systems using non-relational algebraic programming models. In this paper, I propose SODA to break through such kinds of dilemmas and extend these two strategies into a more general approach for processing unstructured data.

Element Pruning (EP): It is common that not all portions of a dataset are used to produce output, which leads to a series of redundant I/O operations, such as Disk I/O for reading/writing data and network I/O for transferring data among computing nodes. These redundant operations can become more severe when processing unstructured data. Due to the lack of a predefined schema of a given dataset, it is difficult to detect data workflow in a fine-grained granularity (*i.e.*, on attribute level), hence failing to identify unused data attributes. In particular, the redundant portion of the dataset may be a dominant barrier to performance when shuffling a huge size of data across networks in a data-intensive computing system.

4.3 Semantics-Aware Data Model

I propose *semantics-aware data model* to keep track of the evolution of dataset(s), which are manipulated by a set of well-chosen operations and elaborated UDFs.

4.3.1 Attribute-Based Data Abstraction

SODA parses and represents an unstructured dataset as a multiset of elements in which repetitive ones may be included, termed as $X = \{x_1, \dots, x_n\}$, where n is the number of elements. To exploit datasets deeper and provide more information to optimizations, SODA treats an element $x \in X$ as an ordered m -tuple: $x = \langle x[a_1], \dots, x[a_m] \rangle$, where $x[a_i]$ is the value(s) of an attribute a_i . One or two datasets can be manipulated by an operation (including user-defined function (UDF)) to generate a new dataset, where the operation can access and transform the attributes of an element. Let $Y = X.op(f)$ denote that a new dataset Y is generated by applying a unary operation op (e.g., *map*, *reduce*, and *filter*) and its corresponding UDF f to an input dataset X . Similarly, I can define binary operations. In the following discussion, I use unary operations to demonstrate the approach for simplicity, and the same idea can be applied to binary operations.

To process such a transformation in static code analysis, SODA first models attributes of X and Y by analyzing their type information, as well as the input and output of f . Let $\beta(X)$, $\beta(Y)$ denote all extracted attributes of X and Y , respectively. Next, SODA analyzes the source code of f to create dataflows between $\beta(X)$ and $\beta(Y)$ at the level of the attribute.

Table 4.1: The Definition of Primitive Operations

Operation	Notation	Examples in Apache Spark
<i>Map</i>	$Map: X \times f \mapsto Z$	map, flatmap, mapValues, mapPartitions
<i>Filter</i>	$Filter: X \times f \mapsto Z$	filter, sample, collect
<i>Set</i>	$Set: X \times Y \times f \mapsto Z$	++, intersection, union
<i>Join</i>	$Join: X \times Y \times f \times K \mapsto Z$	join, leftOuterJoin, rightOuterJoin, fullOuterJoin
<i>Group</i>	$Group: X \times f \times K \mapsto Z$	reduceByKey, groupByKey, aggregateByKey, foldByKey
<i>Agg</i>	$Agg: X \times f \times init \mapsto reg$	reduce, aggregate, fold, max, min

4.3.2 Primitive Operations

SODA defines six primitive operations to imitate common behaviors of a general data-intensive system, as shown in Table 4.1.

- *Map* : $X \times f \mapsto \{ f(x) \mid x \in X \}$ is an operation to return a new dataset by applying f to each element x of X . *Flatmap* is a special map by flattening all elements of the input.
- *Filter* : $X \times f \mapsto \{ x \mid x \in X, f(x) = True \}$ is an operation taking f as a parameter and keeps element x when $f(x)$ is true. *Filter* reduces the number of elements involved in the successive computation so as to reduce data size for computing and communication later.
- *Set* : $X \times Y \times f \mapsto \{ f(x,y) \mid x \in X, y \in Y \}$ is an operation on two input datasets, X and Y , to generate a new one by applying f to each pair of $\langle x,y \rangle$, where the two datasets X and Y should have identical attribute sets.
- *Join* : $X \times Y \times f \times K \mapsto \{ f(x,y) \mid x \in X, y \in Y, x[K] = y[K] \}$ is a binary operation on two input datasets, X and Y , to generate a new one by applying f to each pair of $\langle x,y \rangle$ with matching keys K , where K is a subset of attributes shared by both X and Y .
- *Group* : $X \times f \times K \mapsto \{ f(g_k) \mid g_k = \{x_1, \dots, x_m\} \subseteq X, x_1[K] = \dots = x_m[K] = k \}$ is a unary operation that returns a new dataset by applying f to a group of elements sharing an identical

value(s) k on key(s) K .

- $Agg : X \times f \times init \mapsto f(X, init)$ is to combine all elements in X into a single value with the help of f and an initial value $init$. Note that the result of an operation (e.g., *reduceByKey*) is not a single value, I classify it into a “Group” operation.

Actually, there is an implicit *Shuffle* operation behind the last four operations to transfer data across processing nodes, which dramatically affects the whole system performance due to expensive I/O operations. One of the ultimate goals of SODA is to reduce the amount of shuffling data as much as possible with the help of the proposed optimization strategies. Although the above definitions just involve at most two input datasets, it is easy to extend the concepts to accommodate more.

4.3.3 Data Operational Graph

SODA builds a directed data operational graph (DOG) $G = (V, E)$ to represent an application and conducts three kinds of optimizing strategies atop this graph. A vertex $v \in V$ depicts a primitive operation described in Table 4.1 and the dataset generated by this operation. An edge $e \in E$ denotes data flows between two operations. For each vertex, there is a group of properties accumulated from static analysis, dynamic analysis, or both on code, data, and runtime system, which is defined in Table 4.2 in more detail. I also add two special nodes, named *Source* and *Sink*. *Source* node is connected to all initial input datasets while all sole output of stages would point to *Sink* node. SODA conducts optimizations atop of a DOG, rather than on an abstract syntax tree (AST) due to the following considerations: 1) usually a data-intensive system supports various program language APIs (e.g., Scala, Java, python APIs in Apache Spark), a general optimization backend is compatible with different programming models; 2) SODA focuses on optimizations at the level of operations, rather than at the lower level of AST nodes; 3) There is a huge gap between AST nodes and simulating system behaviors that interpret applications and datasets.

Table 4.2: The statistics information and corresponding notations needed by SODA

Level	Notation	Data Source	Comments
Application	$G = (V, E)$	Source code	Data Operational Graph with nodes V and edges E
	S	Source code, System Log	All stages in an application
	W	Defined by SODA	A binary matrix $W \in \{0, 1\}^{ \mathcal{E}_S \times V }$
	D_1	Defined by SODA	Set of matrices $W \in \{0, 1\}^{ \mathcal{E}_S \times V }$
	D_2	Defined by SODA	Set of matrices $W \in \{0, 1\}^{ \mathcal{E}_S \times V }$ satisfying hypothesis of \mathcal{H}_s
	$F(w)$	Defined by SODA	The expected caching gain (4.4) in D_1
Stage	$L(w)$	Defined by SODA	The concave approximation (4.7) of $F(w)$
	s	Source code, System Log	A stage in an application
	\mathcal{H}_s	Defined by SODA	Cache candidate datasets after a stage s is finished
	C_s	Defined by SODA	The computational cost of a stage s
Operation	T_s	System Log, Runtime Profiler	The submission time of a stage s
	T_v	System Log, Runtime Profiler	The execution time of an operation of v
	U_f	Source code	A Use-Set of an operation op with UDF f
Dataset	D_f	Source code	A Def-Set of an operation op with UDF f
	S_v	System Log, Runtime Profiler	The size of a dataset generated by operation v
System	N_v	System Log, Runtime Profiler	The number of elements in a dataset generated by operation v
	M_{exe}	System Log	The memory size of an executor
	M_{store}	Defined by SODA	The size of storage memory

Execution model. Without loss of generality, SODA splits an execution plan of DOG into a series of stages that are bounded by shuffle behaviors, denoted by $S = \{s_1, s_2, \dots, s_n\}$. As shown in Figure 4.2, the toy application is composed of seven stages. A stage $s \in S$ is delegated as a physical scheduling unit consisting of multiple operations to fulfill a sub-job. Generally, a stage s involves an execution path between the *Source* node and its target v_t (*i.e.*, $s.target$) if no cache mechanism is provided: $s = \{v_0, \dots, v_t\}$. For example, $s_3 = \{v_0, v_1, v_2, v_5, v_6, v_7, v_8\}$ is a set of nodes involved in computing the outcome (*i.e.* v_8) of stage s_3 in Figure 4.2. A number wrapped by dashed rectangles and labeled by texts starting with s indicates a stage of computation and the dependencies of data blocks are represented by solid arrows. Generally speaking, the computational cost of a stage s is calculated by aggregating all involved operations' execution time, denoted as $C_s = \sum_{v \in S} T_v$ where let T_v denote the execution time of an operation of node v . Furthermore, The total execution time of an application is given by summing all stages' cost: $C_S = \sum_{s \in S} C_s$.

It is well known that stages can run in parallel if there is no data dependency among them in a

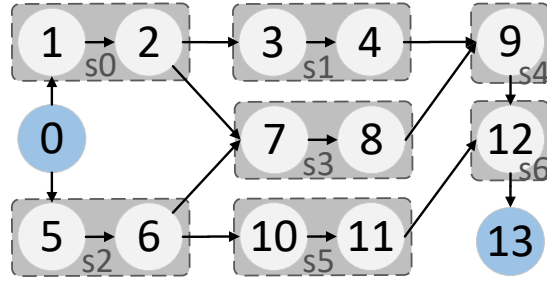


Figure 4.2: Data Operational Graph of Customer Reviews Analysis benchmark.

data-intensive system. However, without loss of generality, I assume that they are scheduled in sequential order. SODA determines this order by analyzing the data dependency of stages and the submission time T_s of stages in prior executions extracted from the performance log. An operation can be executed and interpreted simultaneously by a cluster of executors on different data partitions. Technically, these executors can be equipped with the configurable size of computing resource (*e.g.*, CPU, memory). I also assume that memory resource in an executor is divided into two sections storage (*i.e.*, caching intermediate data) and computation (*i.e.*, allocating objects). I denote M_{store} as the size of storage memory.

4.4 Optimization Strategies

There are three kinds of optimization strategies: cache management, operation reordering, and element pruning.

4.4.1 Cache Management

In this section, I go over the details of the Cache Management policy. The summary notation is categorized and listed in Table 4.2.

Maximizing Expected Caching Gain. A global cache allocation is usually preferred to minimize the aggregated execution cost of an application. In particular, I assume C_0 is the real executing time of an application without any optimizations and works as an upper bound on the expected costs. Here the objective is to determine a feasible cache allocation (*i.e.* w) that maximizes the caching gain, *i.e.*, the expected cost reduction attained by caching data, which is defined as: $F(w) = C_0 - \sum_{s \in S} C'_s$, where C'_s is defined as the predicted (or expected) computational cost of a stage s by consideration of w .

To determine a global cache allocation policy, a binary matrix $W = [w_{sv}]_{s \in \mathcal{E}_S, v \in V} \in \{0, 1\}^{|\mathcal{E}_S| \times |V|}$ is defined to indicate cache status of a data generated by node v after executing a stage s , where $\mathcal{E}_S = \{\mathcal{E}_{s_1}, \mathcal{E}_{s_2}, \dots, \mathcal{E}_{s_n}\}$ reveals the real-time scheduling order of all stages extracted from online profiling information. In the matrix, a cell with value 1 (*i.e.*, $W[s, v] = 1$) indicates that the output of the data of v is reserved in memory after a stage s is done (See Equation 4.5b); otherwise, *i.e.*, when $W[s, v] = 0$, the data is evicted from memory (See Equation 4.5c). It is worth mentioning that cache capacity constraints in an executor (M_{store} is the size of memory for storage) would limit the amount of involved data that could be reserved in memory (See Equation 4.5d). From top to bottom in a column of W , it is easy to identify which stage a data is stored in memory, and which stage it is evicted from memory. Such an allocation plan tells programmers when to persist or unpersist data in memory in code.

Given a global cache allocation, all operations involved in the computation of a stage s are well routed by following the execution path until it encounters data of v cached in memory. This data and its predecessors do not need to be recomputed so far. In the previous example of C_{s_3} , the cost is equal to $T_{v_7} + T_{v_8}$ if data generated by v_2 and v_6 are cached in memory. Next, given the current executing stage s with a global cache allocation $w \in W$, the number of re-computation times of $v_k \in V$ (because it is used again later but not cached) is needed to get the outcome of $v_l \in V$, which defined in Equation (4.1).

$$P(v_k, v_l, s) = \sum_{p \in \tau(v_k, v_l)} \prod_{v \in p} (1 - w[s.pred, v]) \quad (4.1)$$

where $\tau(v_k, v_l)$ returns a set of paths from node v_k to v_l ; if v_k is identical to v_l , then it is $\{\{v_k\}\}$; $s.pred$ reveals the previous executing stage of s . Therefore, the predicted (or expected) computational cost of a stage s can be regulated concisely under a global cache allocation policy $w \in W$, and defined in Equation (4.2).

$$\begin{aligned} C'_s &= \sum_{v \in s}^{v_i=s.target} T_v * P(v, v_i, s) \\ &= \sum_{v \in s}^{v_i=s.target} T_v * \sum_{p \in \tau(v, v_i)} \prod_{v' \in p} (1 - w[s.pred, v']) \end{aligned} \quad (4.2)$$

Finally, I try to obtain an allocation of policy w that maximizes the aggregate expected caching gain:

$$\begin{aligned} F(w) &= \sum_{s \in \mathcal{S}} C_s - \sum_{s \in \mathcal{S}} C'_s \\ &= C_0 - \sum_{s \in \mathcal{S}} \sum_{v \in s}^{v_i=s.target} T_v * \sum_{p \in \tau(v, v_i)} \prod_{v' \in p} (1 - w[s.pred, v']) \end{aligned} \quad (4.3)$$

Convex-Concave Relaxation. In particular, I seek solutions to the following problem:

$$\arg \max F(w) \quad (4.4a)$$

$$\text{s. t. } w \in D_1 \quad (4.4b)$$

where D_1 is the set of matrices $W \in \{0, 1\}^{|\mathcal{E}_S| \times |V|}$ satisfying source constraints, cache behaviors and cache capacity, *i.e.*,

$$\forall s \in \mathcal{E}_S, v \in V : W[s, v] \in \{0, 1\} \quad (4.5a)$$

$$\exists s \in \mathcal{E}_S, v \in V, W[s, v] = 1 : s \xrightarrow{\text{cached}} v \quad (4.5b)$$

$$\exists s \in \mathcal{E}_S, v \in V, W[s, v] = 0 : s \xrightarrow{\text{uncached}} v \quad (4.5c)$$

$$\forall s \in \mathcal{E}_S : \sum_{v \in V} W[s, v] * S_v \leq M_{store} \quad (4.5d)$$

where S_v denotes the size of an intermediate data generated by an operation of v . As far as I know, this deterministic, combinatorial version of (4.4) is NP-hard, even when I already have background knowledge about the submitted application and runtime statistics. Nonetheless, I can relax it to a submodular maximization problem subject to knapsack constraints and take a linear relaxation algorithm to optimize cache allocation on the stage level by minimizing the expected computational cost [168, 169]. It is obvious that Equation (4.4) is not a convex optimization problem. However, it can be approximated as follows. I can define $L : W \rightarrow \mathbb{R}$ based on Equation (4.3) as:

$$L(w) = C_0 - \sum_{s \in S} \sum_{v \in S}^{v_i = s.target} T_v * \sum_{p \in \tau(v, v_i)} (1 - \min(1, \sum_{v' \in p} w[s.pred, v'])) \quad (4.6)$$

Note that L is a concave function, and now I have the following:

$$\arg \max L(w) \quad (4.7a)$$

$$\text{s. t. } w \in D_1 \quad (4.7b)$$

According to [168], an optimal solution w to (4.7) can be approximated and guaranteed within a constant factor $(1 - 1/e)$ from the optimal value of Equation (4.4): $(1 - 1/e)L(w) \leq F(w) \leq L(w), \forall w \in D_1$.

Global Execution Distance. So far, SODA can approximate a solution to (4.7) within a $(1 - 1/e)$ factor by searching all cache allocation space, which may lead to bad runtime performance. In other words, I am convinced that knowledge about data flow and stages' dependency could have a positive effect on this defect. Therefore, I devise a new metric to measure the time-locality distance of an operation, namely execution distance, and introduce another constraint to D_1 .

Definition 4.4.1 (Global Execution Distance (GED)). For a node $v \in V$, execution distance is defined as a relative difference between the current execution point \mathcal{S}_c and a future executing stage \mathcal{S}_f in which it will be referenced: $\mathcal{S}_f - \mathcal{S}_c$.

In particular, there may have multiple execution distances for the data of v if it is used in several stages. At this point, the final number should be the sum of all these distances. For instance, Table 4.3 shows an evolution of execution distance for each node in Figure 4.2 as the workload runs along with scheduling order \mathcal{E}_S from top to bottom. In the first row of the table, I have twelve operations, which may be cached in memory after a stage is done; The leftmost two columns reveal the relationship between stages S and their corresponding scheduling order \mathcal{E}_S . The number in the rest of the cells indicates how far away from a future reference point to the current executing stage, and it should be recalculated and updated after each execution of stages every time. For example, after executing stage s_2 (its corresponding schedule order is 1), the execution distance of v_2 is updated from 5 to 3 since v_2 will be referred in stage s_1 and s_3 and their corresponding schedule order is 2 and 3, respectively. So the new value will be recalculated by $(2 - 1) + (3 - 1)$. A cell $[s, v]$ can be set to zero if 1) the data generated by v is referenced by another node in the same stage s (See case cell of $[0, v_1]$); 2) the data of v gets referenced and there is no more reference in

Table 4.3: The cache allocation policy based on Execution Distance for the workload in Figure 4.2

\mathcal{E}_S	S	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
0	s_0	0	5										
1	s_2	0	3	0	0	0	6						
2	s_1	0	1	0	2	0	4						
3	s_3	0	0	0	1	0	2	0	1				
4	s_4	0	0	0	0	0	1	0	0	2			
5	s_5	0	0	0	0	0	0	0	0	1	0	1	
6	s_6	0	0	0	0	0	0	0	0	0	0	0	0

the future (See case cell of $[3, v_2]$). The cells with empty content mean the nodes that have not been accessed so far.

With the help of GED, I can also learn a set of candidates that can be persisted in memory after a stage s is finished, termed as \mathcal{H}_s . For example, $\mathcal{H}_{s_1} = \{v_2, v_4, v_6\}$ since the corresponding cells are non-zero in the row of $\mathcal{E}_S (= 2)$. Therefore I can narrow down search space to approach an optimal solution to (4.8) by merely considering data in \mathcal{H}_s , rather than all data in V , for a stage s . Consider the following problem:

$$\arg \max L(w) \tag{4.8a}$$

$$\text{s. t. } w \in D_2 \tag{4.8b}$$

where D_2 is the set of matrices $W \in \{0, 1\}^{|\mathcal{E}_S| \times |V|}$ satisfying source constraints, cache behaviors, cache capacity, and hypothesis of \mathcal{H}_s , *i.e.*:

$$\forall s \in \mathcal{E}_{\mathcal{G}}, v \in V : W[s, v] \in \{0, 1\} \quad (4.9a)$$

$$\exists s \in \mathcal{E}_{\mathcal{G}}, v \in V, W[s, v] = 1 : s \xrightarrow{\text{cached}} v \quad (4.9b)$$

$$\exists s \in \mathcal{E}_{\mathcal{G}}, v \in V, W[s, v] = 0 : s \xrightarrow{\text{uncached}} v \quad (4.9c)$$

$$\forall s \in \mathcal{E}_{\mathcal{G}} : \sum_{v \in V} W[s, v] * S_v \leq M_{store} \quad (4.9d)$$

$$\forall s \in \mathcal{E}_{\mathcal{G}}, v \in (V \setminus \mathcal{H}_s) : W[s, v] = 0 \quad (4.9e)$$

It is apparent that D_2 is a subset of D_1 , a solution w' to (4.8) can also be fit for (4.7), as well as (4.4) with $(1 - 1/e)L(w') \leq F(w') \leq L(w'), \forall w' \in D_2$. To gain better approximating rate, I implement Pipage Rounding [168] using Gurobi optimizer APIs [170] to approximate a solution to (4.8).

4.4.2 Operation Reordering

The goal of operation reordering (*i.e.* Filter Pushdown) is to improve applications' performance by reordering operations along with the data path. There are two challenges: Is reordering correct concerning the original semantics? Does the reordering improve performance? To answer these questions, I first define Use-Set and Def-Use by following the dataflow technique in static code analysis [171].

Definition 4.4.2 (Use-Set). Given $Y = X.op(f)$, Use-Set $U_f = \{a \mid a \in \beta(X) \text{ and } a \text{ is accessed by } f\}$. Use-Set defines all attributes of input data used by f to generate Y .

Definition 4.4.3 (Def-Set). Given $Y = X.op(f)$, Def-Set $D_f = \{b \mid b \in \beta(Y) \text{ and } b \text{ is created or updated by } f\}$. Def-Set is the attribute set newly created by an operation op , or inherited directly from $\beta(X)$.

Then, SODA uses a two-step way to handle these two challenges. In the first step (static verification), Theorem 4.4.1 is proposed to ensure semantic correctness. It captures the fact that two successive operations can be reordered if a latter UDF f_2 does not use attributes that a former UDF f_1 defines.

Theorem 4.4.1. Two successive operations op_1 and op_2 on an execution path can be reordered, i.e., $X.op_1(f_1).op_2(f_2) \equiv X.op_2(f_2).op_1(f_1)$, if $U_{f_2} \cap D_{f_1} = \emptyset$.

Let's take filter pushdown as an example to illustrate this theorem. Filter pushdown is a conventional optimization that pushes a *filter* towards the direction of data loading as much as possible so that the volume of intermediate data can be reduced.

Lemma 4.4.2. For $Y = X.Map(f_1).Filter(f_2)$, *Filter* and *Map* can be reordered, if $U_{f_2} \cap D_{f_1} = \emptyset$.

The comprehensive proof statement of Lemma 4.4.2 is followed:

Proof. Assume the two plans

$$O^1 = X.Map(m).Filter(f)$$

$$O^2 = X.Filter(f).Map(m)$$

I prove that $O^1 \equiv O^2$. Assume a record $x \in X$, let $O_x^1 = f(m(x))$ and $O_x^2 = m(f(x))$ are set of element(s) generated by applying according operations by sequence to x . Notice that here $O^1 = \bigcup_{x \in X} f(m(x))$, and $O^2 = \bigcup_{x \in X} m(f(x))$, and in all the proofs, set is referring to dataset (mathematically termed as a multiset) which allows repetitive elements and union operations (here alias to sum operation in multiset) preserve repetitive elements as well. To prove $O^1 \equiv O^2$, it suffices to show that $\forall x \in X : O_x^1 \equiv O_x^2$. I prove it by justifying the following two cases: $1_f(x)$ is the indicator function of filter f to represent its selectiveness. **1.** When $f(x) = 1_f(x) \cdot x = 0 \cdot x = \emptyset$, where Then, $O_x^2 = m(f(x)) = m(\emptyset) = \emptyset$. Now since $U_{Filter} \cap D_{Map} = \emptyset$, I know that for $\forall x' \in m(x)$,

$\pi_{U_{Filter}}(x') = \pi_{U_{Filter}}(x)$, and by definition 4.4.2, f 's behavior is solely depending on attribute set U_{Filter} , I have

$$\begin{aligned}
O_x^1 &= f(m(x)) = \bigcup_{x' \in m(x)} f(x') = \bigcup_{x' \in m(x)} 1_f(x') \cdot x' \\
&= \bigcup_{x' \in m(x)} 1_f(\pi_{U_{Filter}}(x')) \cdot x' = \bigcup_{x' \in m(x)} 1_f(\pi_{U_{Filter}}(x)) \cdot x' \\
&= \bigcup_{x' \in m(x)} 1_f(x) \cdot x' = \bigcup_{x' \in m(x)} 0 \cdot x' = \emptyset
\end{aligned}$$

Thus for $\forall x$, in case 1, I have $O_x^1 \equiv O_x^2$.

2. Similarly, when $f(x) = 1_f(x) \cdot x = 1 \cdot x = \{x\}$, $O_x^2 = m(f(x)) = m(x)$. And

$$\begin{aligned}
O_x^1 &= f(m(x)) = \bigcup_{x' \in m(x)} f(x') = \bigcup_{x' \in m(x)} 1_f(x') \cdot x' \\
&= \bigcup_{x' \in m(x)} 1_f(\pi_{U_{Filter}}(x')) \cdot x' = \bigcup_{x' \in m(x)} 1_f(\pi_{U_{Filter}}(x)) \cdot x' \\
&= \bigcup_{x' \in m(x)} 1_f(x) \cdot x' = \bigcup_{x' \in m(x)} 1 \cdot x' = \bigcup_{x' \in m(x)} x' = m(x)
\end{aligned}$$

Thus for $\forall x$, in case 2, I also have $O_x^1 \equiv O_x^2$. Combining the results in both cases (which are all cases possible), I have proved for $\forall x \in X$, $O_x^1 \equiv O_x^2$, and consequently, $O^1 \equiv O^2$. \square

Correspondingly, I can get the following lemmas to determine if a *Filter* operation can be pushed down before *Group* and *Set* operations, respectively.

Lemma 4.4.3. For $Y = X.Group(f_1).Filter(f_2)$, *Filter* and *Group* can be reordered, if $U_{f_2} \cap D_{f_1} = \emptyset$.

Lemma 4.4.4. For $Z = X.Set(Y, f_1).Filter(f_2)$, *Filter* and *Set* can be reordered along with X and

Y data path safely:

$$Z = X.Filter(f_2).Set(f_1)(Y.Filter(f_2)) \text{ if } U_{f_2} \cap D_{f_1} = \emptyset.$$

In the second step (dynamic evaluation), two polynomial regression models (due to their wide applicability in engineering [172]) are trained for op_1 and op_2 respectively using profiling information, then predict the execution time of each operation on new input. If SODA gets positive feedback from predicted models, it will suggest programmers reorder these two operations.

4.4.3 Element Pruning

Element pruning is an optimization to eliminate unused attributes in an element by analyzing data dependency in the attribute level among operations. SODA analyzes an operation and its associated UDF(s) to analyze attribute dependency between the input and output dataset of this operation. Then a directed data dependency graph (DDG), $G' = (V', E')$, is built to represent the whole data flow of the application by combining all attribute dependency relationships among operations. A node $v \in V'$ represents an attribute of a dataset involved in an operation while an edge $e \in E'$ from a node s to another node d indicates that d has either data or control dependency on s . If an edge is a control dependency, it means that s and d have identical attributes. Data dependency means that the value of d is updated or created from s . An attribute node may have multiple incoming and outgoing edges. To identify an application's start and endpoints, I add two special nodes *source* and *sink* to this graph and connect all input attributes of this application to *source* and connect all output attributes to *sink*. All these dummy edges outgoing from *source* and incoming to *sink* are assigned as control dependencies. Therefore, I can reduce the complicated optimization into a problem of traversing the graph and eliminating a node v if there exists no path between v and *sink* since an attribute node can be eliminated safely if it does not make a contribution to producing an output of the application.

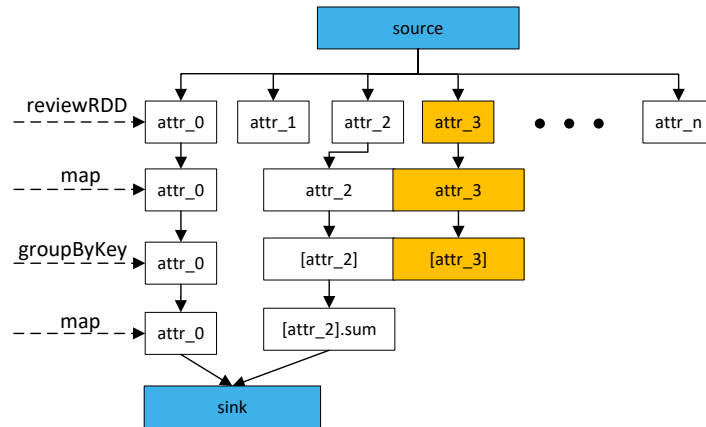


Figure 4.3: A simplified example of data dependency tree

Figure 4.3 shows an example of a data dependency graph of Listing 4.1. Each row represents a group of attributes of a dataset named by the corresponding leftmost text above the dashed arrow. A rectangle reveals an attribute labeled by the inside text. It is obvious that the attribute “[attr_3]” does not contribute to “sink” while it is grouped by *groupByKey* operation from the attribute “attr_3” in first *map*. The preliminary experiment shows that this kind of awkward design leads to a significant computation and I/O cost because of shuffling a huge size of data among computing nodes over the network. According to the proposed constraint, there is no edge between these yellow rectangles and “sink” so they can be removed without changing the snippet code purpose.

Listing 4.1: An example showing the problem of EP

```

val aggData = reviewRDD.map( row =>
  (row.getString(0), (row.getDouble(2), row.getString(3)))
).groupByKey().map{
case (attr_0, attr_2) => attr_2.map(_._1).sum }

```

4.5 Experiment and Evaluation

In this section, I use the real-world data-intensive applications in different domains to evaluate the overall effectiveness of SODA on a 9-node cluster of Apache Spark (v3.0.0) by comparing the runtime performance of these applications before and after optimization by SODA. Each node has a hardware configuration with Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 32GB main memory with DDR4-2133 MHz ECC and 1 GigE Ethernet as the internal communication channel between nodes.

4.5.1 Benchmarks

- **System Log Analysis (SLA)** is a job to find the average ranking and total advertising revenue for each website within a specified date range. There are two datasets, uservisits and pageranks.
- **Customer Reviews Analysis (CRA)** is a project aiming at ranking the top 20 brands according to average customer rating score in the book categories. The review datasets include over 138.1 million customer reviews spanning from May 1996 to July 2014 [173].
- **Social Network Analysis (SNA)** focuses on ranking the top 20 users who are the most active in a specified time period based on tweets analysis. I use a social-media community consisting of 790,462 users who posted over 3,286,473 tweets and have more than 3,055,797 links from 2013 to 2015 [5, 6].
- **Pre-Processing Job (PPJ)** is a clean task and looks for products satisfying two criteria: 1) product ID starts with "B000"; 2) average word count of the product description is greater than 100. N/A data element will be removed to avoid program crashes during runtime. The metadata dataset includes 15.5 million products.

Table 4.4: The results of running SODA on Spark Applications

Bechmark	Description	CM	OR	EP
SLA	Filter, Join, Agg	Detected	Not Present	Detected
CRA	Filter, Join, Agg	Detected	Detected	Detected
SNA	Map, Filter, Agg	<i>Failed</i>	Detected	Detected
PPJ	Map, Filter, Group	Detected	Not Present	Detected

4.5.2 Effectiveness Assessment

To evaluate the effectiveness of SODA, I first manually examine all source code to see which problems are *present* by rules-of-thumb. I then apply each optimization on four benchmarks individually to obtain their results in detecting problems: *Detected*, *Undetected*, or *Not Present*. If a problem is detected but the performance behaves worse after the revision, I label it as a *Failed* case. The results are shown in Table 4.4 allow for quantifying their performance, in which CM, OR and EP represent Cache Management, Operation Reordering, and Element Pruning, respectively. In general, most potential performance problems are detected by SODA successfully with one exception of a *Failed* case in SNA workload when being applied with CM optimization.

- **SLA** is an application working on two datasets to evaluate the performance of CM and EP, and OR is not applied in this application. SODA can scrutinize the problems successfully.
- **CRA** is a complicated student project using Filter, Join, Agg operations, which exposes problems of EP and OR to SODA. In addition, the complicated workflow allows SODA to dig into the CM issue. All of the problems can be detected by SODA successfully.
- **SNA** is a research project that involved all the optimizations. All of them can be detected by SODA statically, however, CM leads to negative feedback regarding execution time while the other two have positive effects on the application. We, therefore, label CM as *Failed*. More discussion regarding this unusual phenomenon will be given in the next section. It is

a research project in which there is a problem with OR. SODA can detect it and give some feedback about how to revise it. SODA also suggests persisting the “user” dataset in memory, since there are multiple references to that data in later computations. Unfortunately, I get a worse performance when the revised application is submitted to Spark. This unusual phenomenon is caused by two reasons: 1) This benchmark is a memory-intensive application; 2) The most of storage memory in an executor is occupied by cached dataset, which leads to high pressure on the GC thread. For this problem, SODA cannot handle it right now since SODA only considers cache memory capacity constraints and fails to take the mutual effect between storage and execution memory into consideration. I will leave this functionality to future work.

- **PPJ** is a data clean task involved in Map, Filter and Group operations. There are two problems of CM and EP, that are successfully detected by SODA.

4.5.3 Performance Behavior

I start the performance improvement evaluation of SODA on workloads for each optimization. I implemented this by submitting the revised code to spark and running it five times for each workload to obtain average experimental data. Figure 4.4 shows the experimental results of execution time, size of shuffling data, and GC time for each benchmark, where Label “RDD” refers to the performance of baseline without any optimizations. Labels “CM”, “OR”, “EP” represent performances of applications optimized by cache management, operation reordering and element pruning, respectively. Each describes experimental results in terms of execution time, shuffling data size, and garbage collection time. Table 4.5 lists the speed up each optimization achieves for each benchmark.

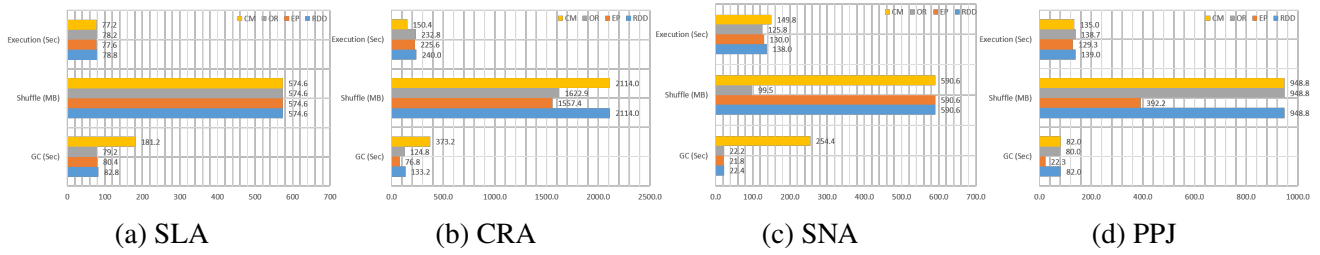


Figure 4.4: The performance of individual optimization over the baseline.

Table 4.5: System speed up of individual optimization over the baseline implementation in RDD.

Benchmark	CM	OR	EP
SLA	2.07%	0.77%	1.55%
CRA	59.57%	3.09%	6.38%
SNA	-7.88%	9.70%	6.15%
PPJ	2.96%	0.24%	7.47%

- SLA.** There are two performance problems: CM and EP, that are detected by SODA. The revised applications are submitted to Apache Spark and become 2.07% and 1.55% faster than the baseline (RDD) (see Table 4.5), respectively. Figure 4.4a reveals that these two optimizations are not related to shuffling data size, while GC time of CM is about 2.2 times faster than the others, since cached dataset triggers a frequent GC procedure to collect JVM garbage.
- CRA.** All three kinds of optimizations, CM, OR, and EP, can be used on this application and their performance speeds up by 59.57%, 3.09%, 6.38%, respectively, according to Table 4.5. In Figure 4.4b, OR and EP have a positive effect on execution time and shuffling data size, while CM has speedup over execution time and does not reduce shuffling data size. Even CM has a better performance than the other two, however, the corresponding GC time is bigger. EP can reduce shuffling data size significantly but with the minimum time consumed.

Table 4.6: Overall comparison about System Overheads incurred by SODA

Benchmark	Optimization	No	Partial	All
SLA	CM	78.8	87.3	107.6
CRA	OR	240	275.3	532.3
SNA	OR	138	153.4	197.6
PPJ	CM	153.6	176.3	317.5

- **SNA.** Table 4.5 shows that after applying the three optimizations, this application speeds up by -7.88%, 9.70%, 6.15%, respectively. I believe two reasons are causing this worse performance (-7.88%) of the revised application-optimized by CM: 1) this benchmark is a memory-intensive application; and 2) most of the storage memory in an executor is occupied by cached dataset, which leads to high pressure on garbage collection threads. Since SODA only handles cache memory capacity constraints and does not consider the mutual effect between storage and execution memory, such a case is difficult to be avoided. Additionally, OR has reduced shuffling data size significantly.
- **PPJ.** According to Table 4.5, EP and CM can speed up the application by 7.47% and 2.96%, respectively. In Figure 4.4d, the shuffling data size has been reduced by EP from 948.8 MB to 392.2 MB while the GC time is decreased to 22.3 seconds.

4.5.4 System Overhead

In this section, I conduct experiments in different granularity of monitoring, *e.g.* monitoring no operation, partial operations suggested by SODA, or all operations involved in applications, to compare system overhead. Table 4.6 shows the execution time of each application with different monitoring granularity. In the partial granularity, I get profiling guidance for SLA and PPJ based on CM's suggestions, CRA and SNA based on OR's suggestions. Monitoring on all operations

takes a longer time than the other two granularities. The rational reasons behind the acceptable system overhead lie in the lightweight design of *online* phase: 1) Enabling and customizing Spark internal *event* and *metrics* subsystems only cast needed information with a lower system overhead; 2) Exploiting data access pattern behind semantics code and DAG-based workflow provides an instrumentation guide to probe runtime system. For instance, I only consider candidate operations contributing to future ones if they are persisted in memory. It is worth mentioning that the system overhead of an application depends on its characteristic, input data size, and system configurations.

CHAPTER 5: GRAPH TRANSFORMER FOR AUTOMATED PROGRAM REPAIR

5.1 Introduction and Motivation

Usually, an Automated program repair (APR) task can be considered a neural machine translation (NMT) task that learns translation from buggy code to produce the corresponding fixed version [117, 23, 174]. Although these innovative methods have outperformed some rule- or mining-based approaches on specific tasks with desired performance, grand challenges still exist when applying NMT techniques to source code.

Challenge-1: Unbounded Vocabulary. It is extremely challenging to learn bug-fix patterns at the raw code level because of the large (or even unlimited) and sparse vocabulary caused by the fact that software developers tend to define identifiers as they like. Although a large body of research work has been proposed to handle these issues [175, 176, 177, 21, 22, 178], there are still many drawbacks. For instance, translating rare tokens with their subword units can undermine the structural representation of a program [175]; SequenceR [21] using the copy mechanism to overcome the unlimited vocabulary; BFP [22, 116] failed to take semantics and lexical scope of tokens into consideration when renaming identifiers in a buggy and fixed code to reduce vocabulary size. In this paper, I design and develop a novel pre-processing mechanism based on semantic and lexical scope knowledge of a token to prepare datasets.

Challenge-2: Structure and Semantics of Missing. Usually, a program is a stream of tokens comprising language keywords (*e.g.*, `for`, `if`), separators (*e.g.*, `"(", ":", "}"`), arbitrary identifiers, and literals defined by developers. These tokens are orchestrated structurally and semantically, which leads to the fact that tokens may represent different semantics in a variety of scopes even if

they share an identical name. Unfortunately, these NMT-based models for program repair and their corresponding embedding layers (*e.g.*, Word2Vec [179, 180], or CodeBERT [181], or a learnable neural network) fail to well capture code structural and semantic information. Thus, it is difficult to learn mappings between program tokens in the buggy and fixed code concerning structural and semantic constraints, which leads to imprecise mappings and incorrect code patches. In this work, I encode semantic information of a token and its surrounding context as a path from the root node to this terminal token in an abstract syntax tree (AST) and hypothesize this auxiliary context-path information can improve program repair tasks.

Challenge-3: Context Misalignment. Context alignment is referred to as the context of tokens in fixed code that should be consistent with corresponding ones in buggy code. It is well known that the attention mechanism used in NMT decoder models can exploit alignment probabilities between buggy and fixed input, however, attention to the context information rather than the aligned source tokens might be more helpful for translation quality [23, 182, 183, 184, 185, 186, 187]. Without such context knowledge, the models must learn to implicitly align the source code before and after the fix. The learned alignment might be imprecise, making those NMT-based models incorrectly identify the fixing location in the new buggy code. In this paper, I propose context-aware attention to enhance Transformer to learn context-aware alignment, and leverage multi-task learning to exploit the correlations between lexical translation and the alignment tasks.

In response to the observations and concerns raised above, a novel *semantics-preserved, scope-oriented, and vocabulary-closed* context abstraction approach is designed for pre-processing code corpus. I then conduct joint multi-task learning about code translation and context-aware alignment using Transformer [7] architecture for automatically detecting and repairing bugs.

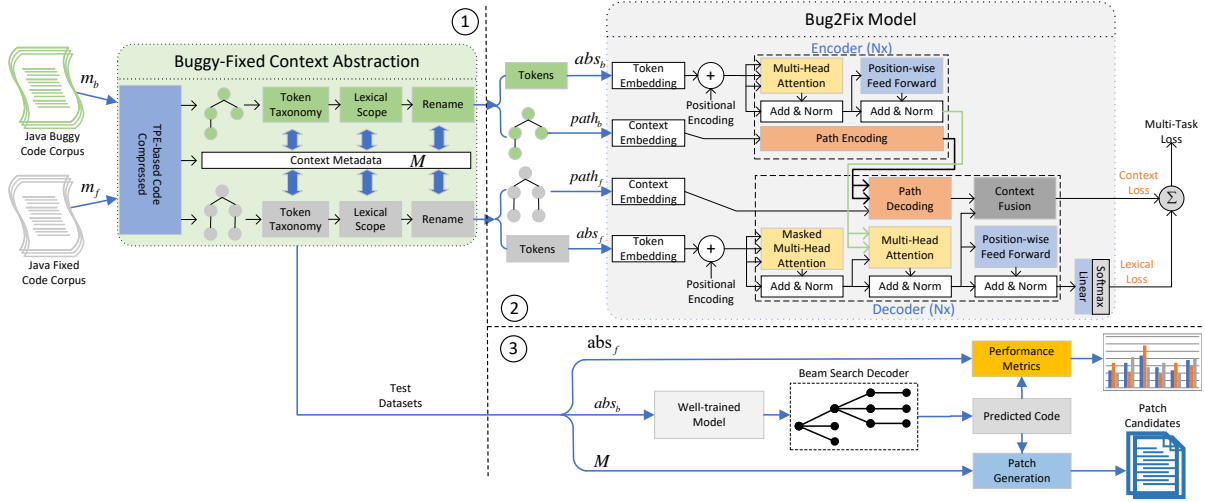


Figure 5.1: Overview of the proposed Bug2Fix for APR tasks

5.2 System Overview

Figure 5.1 shows the workflow of Bug2Fix, which involves the following key steps:

① **Pre-Processing: Context Abstraction.** I take the following steps to prepare for training and testing Bug2Fix. First, a pair of buggy (m_b) and fixed (m_f) codes are scanned by the token pair encoding (TPE) algorithm to compress code structure without losing its key semantics (See details in 5.3.1). Then, I perform renaming on identifiers in ASTs according to their types and lexical scopes to abstract code representations (*i.e.*, abs_b and abs_f are generated for m_b , m_f , respectively. See details in Section 5.3.2 & 5.3.3). Finally, I transverse the ASTs of abs_b and abs_f to extract a set of paths between the root node and each identifier as the input for further context-aware alignment task (see details in 5.3.4).

② **Multi-task Learning: Code Translation and Context-aware Alignment.** To improve the performance of program repair tasks, Bug2Fix performs two sub-tasks concurrently to jointly learn bug/fixes patterns and context alignment using multi-task learning techniques. The first task is to

learn how to translate abs_b into abs_f at the token level, which is fulfilled by a sequence-to-sequence model (*i.e.*, Transformer [7]). It is well known that code translation and token context alignment tasks are very closely related and program repair tasks can benefit from multi-task learning by exploiting the correlations between the two tasks. To this end, novel context-aware attention is designed and embedded in an encoder-decoder attention sub-layer to learn context alignment. A novel loss function combining lexical loss of code translation task and context loss of alignment task is used to update the model.

③ **Patch Inference.** A well-trained model can be used to predict and generate patches for test datasets using a beam search decoder [23, 16, 188, 189]. These patches are evaluated comprehensively in the following Sections to acknowledge the effectiveness of Bug2Fix. The major intuition behind beam search is that rather than predicting a token with the best probability at each time step, the decoding process keeps track of s hypotheses if beam size is s . At each step, then I pick the first s candidates and combine them with the prior s beam paths to form $s \times s$ paths. From these candidate paths, the decoding process keeps s sequences with the highest probability. The process continues until each hypothesis reaches the special token representing the end of a sequence. I consider these s final sentences as candidate patches for the buggy code. Note that when $s = 1$, beam search decoding is a greedy strategy.

Technically, Bug2Fix is designed and developed atop of Transformer [7] and enhanced by a novel context-aware attention layer to repair semantic bugs on the method level. given the following considerations: 1) A method represents a reasonable unit of operations and is implemented as a single task or functionality. 2) A method provides meaningful context including variables, parameters, and intraprocedural calls, to learn bug-fix patterns. 3) The other options such as class-level or package-level possess a bunch of redundant contexts, which lead to a complex model that is too hard to be trained. This selection of method-level for NLM models has been justified by recent empirical studies [121, 190, 22, 24].

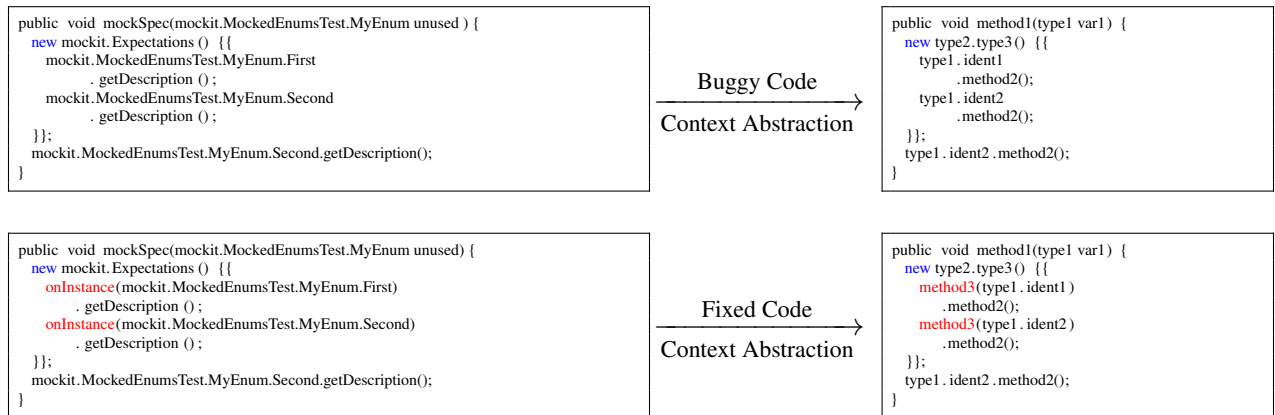


Figure 5.2: The Context Abstraction of buggy (top panel) and fixed (bottom panel) code

5.3 Pre-Processing: Context Abstraction

A pair of buggy (denoted by m_b) and the corresponding fixed (denoted by m_f) codes are used to train the NLM model to learn a translation from m_b to m_f , thus generating patches. The arbitrary and various conventions to name variables and methods in a big code project hinder the goal of learning transformations in light of a huge and unbounded vocabulary containing many rare tokens. Therefore, it is extremely challenging to learn bug-fix patterns at the level of raw source code. For this reason, I first abstract the context of the buggy and fixed codes, then generate an expressive yet vocabulary-limited representation. In this section, I propose an approach to compress and abstract the context of m_b and m_f , then generate an expressive yet vocabulary-limited representations (*i.e.* abs_b , abs_f and M).

5.3.1 Token Pair Encoding

Usually, duplicated code snippets make less contribution to code structure and semantics, which leads to a larger size of code sequence and two further research challenges: 1) undesired redun-

dant tokens for model training, 2) the repeated identifiers used under different scope contexts [191, 192, 175, 178]. In this study, I propose a novel token pair encoding (TPE) approach by extending Byte pair encoding (BPE) [193] to compress code at the token level while preserving its key structural and semantic information. BPE is a data compression technique for a character string, where the most common pair of consecutive bytes in a sequence is replaced with a single and unused byte iteratively. It has widely been used in NLP applications [194, 195, 175, 178, 181, 196]. However, when handling program code, disassembling an identifier into multiple subword units can ruin the structural representation of a program and hinder NLM tasks from learning structural information to repairing semantic bugs. To overcome the aforementioned drawbacks, TPE makes the following modifications and improvements: 1) It performs on token level, rather than character granularity; 2) It maintains a vocabulary containing all the original tokens plus the symbols created from the merge operations; 3) The ordered list of merge operations performed in each iteration is provided to recover the original sequence later. The benefits of TPE are twofold. First, the code size of a program can be reduced while its key structural information is preserved. During traversing AST to generate abstract code, the most common sub-tree found by TPE will be replaced by a single identifier according to their roles in AST. Secondly, TPE allows for fine-grained control of vocabulary size by tuning the number of merge operations. For example in Figure 5.2, there are multiple kinds of compression granularity, such as `mockit.MockedEnums` and `mockit.MockedEnumsTest.MyEnum`, which leads to two different vocabulary sizes.

5.3.2 Taxonomy, Lexical Scope, and Idioms

As mentioned earlier, a program method is coded by a formal language (*e.g.* Java) using a set of language keywords (*e.g.* `for`, `if`), special marks (*e.g.* `"`, `;`, `}`), as well as identifiers and literals, declared by developers. For an NLM task, it is essential to discern the role of each identifier (*i.e.* type name, variable or method) and the type of a literal. Identifying the lexical scope of an

identifier is also critical. Establishing a set of top- k most common tokens is also beneficial for understanding the code semantics. I will elaborate on each of these three properties in the remainder of this section.

Taxonomy. For a translation task in NLP, it is to learn word embedding using a neural network, by which an individual token is encoded as a real-valued vector in a predefined vector space. In this process, words with more similarity in context have more in common in their representations [197, 180]. From this perspective, it is critical for word embedding to identify the structural and semantic information of each token. To achieve this goal, a program is parsed and represented as an AST by a Java Parser [198] and each node of the tree denotes a construct occurring in the source code, such as the declaration of variables and methods, method calls, string, and numerical literals. Next, I traverse AST to discern the role of each identifier and the signature of a literal (*e.g.*, Integer, Long, String), and then use this information in the following renamed procedure to assign a new ID to each token if need.

Lexical Scope. It is common to use repeated identifiers to define variables, classes, and methods, such as local variables, parameters, and private methods in Java. However, these duplicated identifiers tend to be localized in various scopes and may represent different semantic meanings. Code structures used to define repeated identifiers within their corresponding nested scopes lead to the issue of nested-scope vocabulary. These repeated identifiers then may hinder NLM tasks from learning their specific meanings [176]. Therefore, a lexical scope analysis is needed to capture the local repetition within a scope s_i , and then make it available to scopes (*i.e.*, $s_i, s_{i+1} \dots$) nested within s_i . In the design, I first traverse AST with a stack data structure of scope indicator to identify the scope of each identifier and then verify whether or not a new ID is needed for an identifier if it already appears in previous scopes.

Idioms. Keywords are a group of special meaningful words reserved by a programming language so that a developer can orchestrate their combinations with identifiers and literals in a manner of structural and semantic constraints to fulfill jobs. Some research studies propose approaches to handle domain- or developer-specific vocabularies [199]. Usually, NLM tasks can benefit from these reserved keywords and idioms to exploit code structures and semantics efficiently. However, reserving a limited number of tokens and keeping their as-is text in the source code may lead to the greater complexity of models and an extreme challenge to train them. To achieve a trade-off between these two, I follow similar steps in BFP [22] to reserve the top 0.005% frequent words (outliers of the distribution) as the list of idioms according to the results achieved by analyzing the distribution of the frequencies.

5.3.3 *Semantics-preserved and Scope-oriented Rename*

Given the structural and semantic meanings of each token, the lexical scope of an identifier, keywords, idioms, identifiers, and literals are renamed to new IDs to reduce vocabulary size in this study. To this end, I leverage *implicit classes* feature in Scala programming language [200] by adding a functionality (*i.e.*, a method of `genCode`) to closed classes of AST nodes defined by a Java Parser [198]. Then I traverse the code's AST and use this functionality to generate abstracted code. For m_b and m_f , I first parse them as two independent ASTs, then traverse them to discern the role of each identifier and the signature of a literal. Next, I rename identifiers and literals to different IDs in a sequential and positional fashion, that is 1st method, variable and type name are assigned with IDs of `method1`, `var1` and `type1`, respectively; likewise the 2sts receive IDs of `method2`, `var2` and `type2`. This similar process applies to all literals, *e.g.*, `stringX`, `intX`, `floatX` where the last char X means the ordinal position appearing in the source code. During the abstraction process, global context metadata (denoted by M) is created to memorize the mappings between identifiers/literals and their corresponding abstracted IDs. As shown examples in Figure

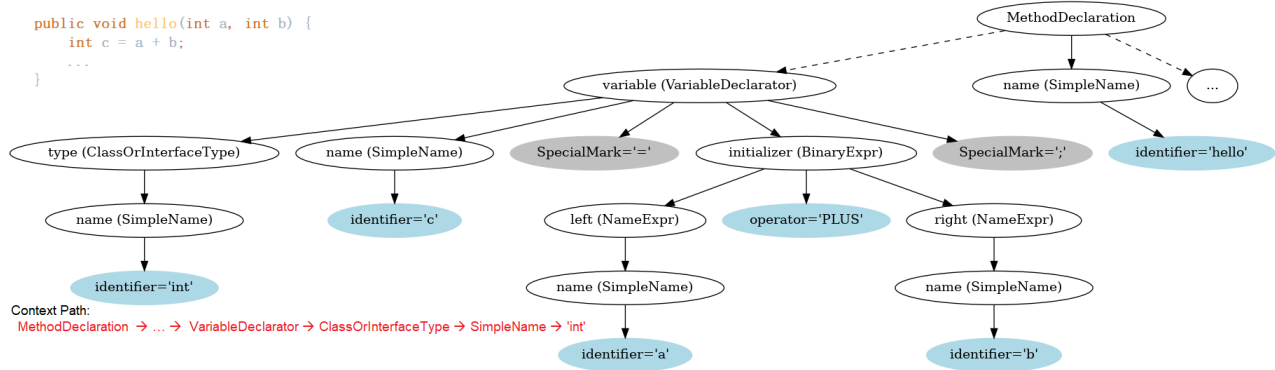


Figure 5.3: An AST example of Java program along with an example of one of the paths

5.2, the right column represents abstraction codes (*i.e.*, abs_b and abs_f) for the original ones (*i.e.*, m_b and m_f), respectively. Finally, I define (abs_b, abs_f, M) as the abstraction representation of an instance (m_b, m_f) . At this point, the abstraction code representation is composed of a stream of language keywords and idioms, separators (*e.g.* “;”, “:”, “(”, “)”,) and IDs representing identifiers and literals. Next, these abstracted code corpora will be fed to Bug2Fix to learn a code translation from abs_b to abs_f . context metadata (*i.e.* M) will be used in predicting phase to generate patches for potential errors via beam search strategy.

5.3.4 Context Path

In this section, I introduce a context path extracted from AST that represents auxiliary knowledge for each terminal node (nonterminal nodes are out of consideration). Given an AST, a *context path* of a terminal node t is a path from the root node to t , denoted by $p_t = \{v_1 \rightarrow \dots \rightarrow v_k \rightarrow t\}$, where a v_i ($i \in [1, k]$) is a nonterminal node in AST and the v_1 is the root node. The t is a token in the source code.

As shown in Figure 5.3, I first parse the program’s source code into an AST to capture its syntactic

structure, in which the ones filled in blue or grey colors are *terminal* nodes. Unlike previous approaches [201, 202, 203, 204], I additionally add *SpecialMark* nodes (e.g., the ones with grey color in Figure 5.3) with edges connecting them with their corresponding parent nodes, to store the value of tokens that do not appear in AST, such as semicolon, bracket symbols. The purpose of introducing *SpecialMark* is to provide a way to exploit all context information for each token in the source code. Then I transverse this modified AST to get all terminal nodes and their corresponding context path recursively. As shown at the left bottom of Figure 5.3, the context path of identifier `int` starts with root node `MethodDeclaration` and ends up with its value as shown in the source code.

Thus, a set of paths from the root node of AST to each terminal token are extracted, and the GumTree Spoon AST Diff tool [205] is leveraged to construct a ground truth alignment matrix between ASTs of abs_b and abs_f . Let $G_{m \times n}$ denote a 0-1 matrix such that $G_{i,j} = 1$ if the j^{th} node in the AST of abs_b is aligned to the i^{th} node of abs_f 's AST. I normalize each value in the rows of matrix G that correspond to fixed tokens to get a matrix G^P as the golden target for the supervised learning of the context-aware alignment task.

5.4 Code Translation and Context-aware Alignment

5.4.1 Token and Context Embedding

Initially, I need to compute vector representations for all the abstracted code tokens and their corresponding context path within the method pairs.

Token Embedding. I use a neural network to learn token embedding, *i.e.*, how to encode abstracted tokens to vector representations. The buggy and fixed token embedding layers share the same weight matrix and vocabulary. By comparison with BPE-based NLM models [194, 195, 175,

178, 181], I apply a pre-trained CodeBERT [181] model on plaintext code corpus to obtain vectors for all tokens to verify the effectiveness of the proposed context abstraction. CodeBERT is a bi-modal pre-trained model for a programming language (PL) and natural language (NL) and learns general-purpose representations that support downstream combined NL-PL applications such as code documentation generation and program repair. To improve the handling of rare tokens, CodeBERT divides them into a limited set of common sub-word units (“wordpieces”) for both input and output [174].

Context Embedding. Given a set of context paths $\{p_1, \dots, p_m\}$, I design a neural embedding layer atop of code2seq [202] model to generate a vector representation z_i for each path $p_i = \{v_1^i \rightarrow \dots \rightarrow v_2^i \rightarrow v_l^i\}$. Intuitively, I first get a vector representation ($E_{v_l^i}^{tokens}$) for the last terminal node (v_l^i) from Token embedding layer directly, and represent each nonterminal node in the rest of the path ($v_{<l}^i$) using a learned embedding matrix E^{node} . Next, a LSTM is leveraged to encode the path using its final states: $h_{<l}^i = LSTM(E_{v_1^i}^{node}, \dots, E_{v_{l-1}^i}^{node})$. Finally, I concatenate the path’s representation with the token representation, and apply a fully connected layer to get the vector representation: $e_i = \tanh(W_e[h_{<l}^i; E_{v_l^i}^{tokens}])$ where W_e is a learnable hyper-parameter matrix.

5.4.2 Context-aware Attention

The attention mechanism can be considered as a procedure that maps a query for a set of key-value pairs to an output [23, 7, 206]. The output is computed as a sum of weighted values and the weight assigned to each value is computed by a compatibility function of the query with the corresponding key, as shown in Equation 5.1.

$$Attention(q, K, V) = \alpha_q \times V \alpha_q = softmax\left(\frac{qK^T}{\sqrt{d_k}}\right) \quad (5.1)$$

where a query (*i.e.*, q), keys (*i.e.*, K) and values (*i.e.*, V) are vectors with dimension d_k . Accordingly, it computes the dot products of q with all K , divides each resulting element by $\sqrt{d_k}$, and applies a *softmax* function to obtain the weights (*i.e.* α_q) for V . In the attention sub-layer in Bug2Fix decoder model, the vector $\alpha_q \in \mathbb{R}^{1 \times n}$ denotes the attention probabilities for a target (fixed) token over all the source (buggy) tokens. Thus, an attention matrix $C_{m \times n}$ be constructed by stacking together α_q (s) corresponding to all the fixed tokens. Instead of implementing a single attention function, Bug2Fix also adopts multi-head attention that allows the model to jointly summarize information from different representation sub-spaces at different positions. For any particular head, there is a corresponding attention matrix $C_{m \times n}$. A common way to obtain alignments between buggy and fixed code tokens is to average all attention matrices across all heads within each layer. Unfortunately, this method is quite erroneous [207, 186, 187] and there is no context knowledge involved in the attention. In this paper, I introduce a context-aware attention network residing along with the encoder-decoder attention sub-layer in the Bug2Fix decoder model, which is supervised by external context path information. The obtained context-aware alignment loss will be combined into lexical loss by multi-task learning techniques to enhance the ability of Bug2Fix for repairing bugs precisely (See details in 5.4.3).

5.4.3 Multi-task Learning: Code Translation and Context Alignment

Contemporary NMT models are largely based on an encoder-decoder architecture [117, 23, 174], where the encoder maps an input sequence of tokens $x = (x_1, \dots, x_n)$ to a sequence of continuous representations $z = (z_1, \dots, z_n)$. Given z , the decoder then generates a sequence of output tokens, $y = (y_1, \dots, y_m)$, one token at a time. At each decoding step, the probability of the next target token

depends on the previously generated token, and can therefore be factorized as:

$$p(y_1, \dots, y_m | x_1, \dots, x_n) = \prod_{j=1}^m p(y_j | y_{<j}, z_1, \dots, z_n) \quad (5.2)$$

In this paper, I model APR tasks as a sequence-to-sequence translation problem. Bug2Fix can learn a neural translation from the buggy code to a fixed one to repair bugs automatically. In the current design, I use a regularization technique called label smoothing [208] instead of using hard labels directly, to calculate gradients to update the model, as shown in the following Equation:

$$\mathcal{L}_t = - \sum_{i=1}^m \sum_{y=1}^L [(1 - \omega)p(y|x_i) + \omega u(y|x_i)] \log q_\theta(y|x_i) \quad (5.3)$$

where L are candidate labels $\{1, 2, \dots, L\}$, $u(y|x)$ is a noise distribution of labels and $\omega \in [0, 1]$ is a weight factor. It computes cross-entropy with a weighted average of hard labels and uniform distribution over labels, rather than with hard labels. I name \mathcal{L}_t as *lexical loss*.

In order to make Bug2Fix embrace context knowledge of the input program, I introduce dedicated context attention to learn context-aware alignment. Let $C_{m \times n}$ denote the attention matrix computed by the context attention. For every fixed token i , I minimize the Kullback-Leibler divergence [209] between ground truth G_i^p and C_i which is equivalent to optimizing the following cross-entropy-based context-aware alignment loss:

$$\mathcal{L}_c(C) = - \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n G_{i,j}^p \log(C_{i,j}) \quad (5.4)$$

where n, m means the number of tokens in buggy and fixed code. I train the model to minimize $\mathcal{L}_c(C)$ in conjunction with the lexical translation loss \mathcal{L}_t . Therefore, the overall loss is: $\mathcal{L} = \mathcal{L}_t + \lambda \mathcal{L}_c(C)$, where λ is a learnable hyperparameter.

5.4.4 Patch Generation via Beam Search

I deploy the well-trained model to predict and generate patches for test datasets using a beam search decoder [23, 16, 188, 189]. The major intuition behind beam search is that rather than predicting a token with the best probability at each time step, the decoding process keeps track of s hypotheses if beam size is s . At each step, then I pick the first s candidates and combine them with the prior s beam paths to form $s \times s$ paths. From these candidate paths, the decoding process keeps s sequences with the highest probability. The process continues until each hypothesis reaches the special token representing the end of a sequence. I consider these s final sentences as candidate patches for the buggy code. In the experiments, I increase the beam size s from 1 (*i.e.*, a single patch is created by \mathcal{M}_D using a greedy decoder) to 50 (*i.e.*, 50 patches are created) with the step of 5. Finally, these generated patches can be concretized by replacing all identifiers with their actual values in M .

5.5 Research Questions

In this study, I aim to answer the following three research questions: **RQ1:** *How to evaluate the quality of context abstraction algorithm?* **RQ2:** *How to assess the overall performance of Bug2Fix for repairing bugs?* **RQ3:** *What kinds of bugs can be fixed by Bug2Fix?*

5.5.1 Quality of Context Abstraction

To evaluate the performance of context abstraction, I perform three kinds of abstraction approaches (*i.e.*, Plaintext, BFP [22], and Bug2Fix) on three datasets (*i.e.* *small*, *median* and *big*). In Plaintext, a program is kept as-is characters in the original dataset except that 1) comments and annotations are removed; 2) it is converted to a one-line representation by replacing all newline characters with

spaces; 3) redundant space symbols are removed. Plaintext’s output is used as a gold standard dataset to compare the different abstraction approaches. BFP [22] renames identifiers and literals to new ones by considering their types and absolute positions [22, 116]. Bug2Fix exploits a group of special program properties to compress and extract meaningful context of the code without damaging its key structure. Finally, I get three abstracted outputs for each dataset and evaluate these outputs in terms of data quality and model behaviors.

Quality of Abstracted Data. For each abstracted dataset, I first calculate the average number of tokens per instance and vocabulary size of the buggy and fixed code corpora, respectively. Then, Bilingual Evaluation Understudy (BLEU) score [210] between buggy and fixed corpora are computed to indicate their matching quality; perfect match results in a BLEU score of 100.0, whereas a complete mismatch leads to a score of 0.0. Finally, BFP’s and Bug2Fix’s relative degradation of score over Plaintext’s one is defined in $|(BLEU(D'_b, D'_f) - BLEU(D_b, D_f))|/BLEU(D_b, D_f)$, to quantify their difference of data quality, where D_b and D_f represent buggy and fixed code corpora in Plaintext, respectively; D'_b and D'_f denote buggy and fixed ones generated by BFP or Bug2Fix.

Model Behaviors. I aim at evaluating the abstractions’ performance using a vanilla Transformer architecture [7]. Specifically, let $\mathcal{M}_{D'}$ be a model trained and evaluated by 80% and 10% of an abstracted dataset $D' \in \{small_{abs}, median_{abs}, big_{abs}\}$, where $abs \in \{Plaintext, BFP, Bug2Fix\}$. When $\mathcal{M}_{D'}$ is converged, the test datasets T' (10% of D') are used to predict the corresponding fixed versions by a beam search strategy. The test process is performed as follows: for an instance $d = \{abs_b, path_b, M\} \in T'$, I feed abs_b to the well-trained $\mathcal{M}_{D'}$, performing inference with beam search decoder given a beam size s . The model will generate s different potential patches $p_d = \{p_d^1, \dots, p_d^s\}$. Ultimately, $\mathcal{M}_{D'}$ processes all instances in T' by following the above steps, then generates predicted code corpora P . P and T' will be evaluated by the following measurement to verify the performance of the models.

- *Success Ratio*. I would say that \mathcal{M}_D successfully fixes abs_b in d if $\exists p_d^i \in p_d : p_d^i \stackrel{s}{=} abs_f$, where $\stackrel{s}{=}$ means that these two code should be semantically equivalent [211]. In the implementation, I use the GumTree AST Diff tool [205] to compute the difference between their ASTs; they are equivalent semantically if there is no difference. p_d^i is output as a perfect and semantic match case for abs_b . In the experiments, I only report the raw count and the percentage of successfully fixed in T' as *Success* ratio. Besides that, I calculate two scores for a p_d^i if there is no equivalent prediction found in p_d : 1) action hit between abs_b and p_d^i (See details in 5.5.2); 2) BLEU score (divided by 100) between abs_f and p_d^i . Then the averaged value of these two scores is assigned as a selection indicator of p_d^i . I follow the same procedure to compute all indicators for the rest of the patches in p_d , the one with the maximum value will be chosen as the most matched prediction for abs_b . These selected best (nearly) matched patch for each instance in T' outputs as the best predicting results of the test dataset for further assessments.

It is worthwhile to mention that I replace the encoder component in Transformer with a pre-trained CodeBERT model [181] and fine-tune it to evaluate performance on the Plaintext dataset since I can not train the model because of the explosive vocabulary size in raw plaintext. CodeBERT is a bimodal pre-trained model for a programming language (PL) and natural language (NL) and learns general-purpose representations that support downstream combined NL-PL applications such as code documentation generation and program repair. To improve the handling of rare tokens, CodeBERT divides them into a limited set of common sub-word units (“wordpieces”) for both input and output [174].

5.5.2 Overall Model Performance

I evaluate the performance of Bug2Fix against two baseline models, SequenceR [21] and BFP-RNN [22] using the three datasets. SequenceR leverages a sequence-to-sequence model with a copy mechanism to overcome the unlimited vocabulary and learn translation from buggy code to fixed version. While BFP-RNN learns buggy/fixes translation using an attention-based Encoder-Decoder model with code abstraction and keyword replacing to repair bugs automatically. After conducting experiments following similar train and inference procedures in Section 5.5.1, I not only report *Success Ratio* for each model but also compare experimental results using the following two measurements:

- *Action Hit*. First of all, I use GumTree Spoon AST Diff tool [205] to compute the AST difference between abs_b and abs_f in d , denoted by A_d^g . This computes a sequence of actions performed at the AST level to transform abs_b 's AST into abs_f 's AST. Then I measure the difference between abs_b and one of its potential patch $p_d^i \in p_d$, which is termed by A_d^i , further to calculate the percentage of how many actions in A_d^i can be hit in A_d^g , that is $\frac{|A_d^i \cap A_d^g|}{|A_d^g|} \times 100\%$. Likewise, I compute all hit ratios for the rest of the patches in p_d and then average them to identify the quality of this inference given the input of the instance d . I can also expand this metric to T' by computing action hits for each instance and averaging them. In the following results, I report action hits over the whole test dataset.
- *Syntactic Correctness*. I also analyze the syntactic correctness of the patches for abs_b in d . That is, I feed each potential patch p_d^i to Java Parser [198]. In the process of compilation, I just focus on scanning the code to check whether it is lexically and syntactically correct since it is impossible to download full projects to accomplish compilation for each instance. Then I report the percentage of how many patches can be parsed in p_d . The overall score can be computed by averaging all correctness values of all instances in P .

5.5.3 Semantic Bug Repair

I investigate the types of semantic bugs that can be fixed by Bug2Fix using automated analysis. First, I only focus on instances that are successfully repaired by Bug2Fix (beam size $s = 25$) and analyze the types of AST operations performed during the fix. Let p'_d be the patch generated by the model to fix the buggy code (abs_b) in an instance (abs_b, abs_f) . Specifically, I use the GumTree AST Diff tool [205] to compute the AST difference between abs_b and its p'_d , and list all edit actions needed to repair abs_b at the AST level. Although the model does not technically work on the source code's AST, but rather on sequences of abstracted datasets, it is still worthwhile to understand the types of AST operations that such a model can emulate. Since the proposed context abstraction algorithm can produce abstractions of the buggy and fixed code with good quality. The GumTree tool offers four edit actions to transform an AST to a new one: **Delete** is a kind of operation to delete a node in the AST; **Insert** can add a new node to the AST; **Move** is a way to shift an existing node to a different location in the AST; **Update** replaces the value of a node with a new one in the AST.

5.6 Experiment and Evaluation

In this section, I shall evaluate Bug2Fix's performance comprehensively.

5.6.1 Dataset

I create three Java code corpora based on previous work [21, 22] to evaluate Bug2Fix's performance. In [22], two method-level datasets (*small* and *median*) were built according to bug-fix commits from GitHub between March 2010 and October 2017. I use their source code directly but different context abstractions to pre-process them in this study. In addition, I create a *big* dataset

Table 5.1: Three datasets and their statistical information in Plaintext, BFP, and Bug2Fix, respectively.

Dataset	Type	N ^o of Instances	Plaintext			BFP			Bug2Fix		
			Tokens	Vocabulary	BLEU	Tokens	Vocabulary	BLEU	Tokens	Vocabulary	BLEU
<i>small</i>	Buggy	58350	52	179759	70.26	32	427/433	77.94 (10.93%)	36	491/504	70.19 (0.10%)
	Fixed		47	179759		29	431/433		33	492/504	
<i>median</i>	Buggy	65455	114	347159	84.45	75	491/493	90.79 (7.51%)	85	602/607	82.88 (1.86%)
	Fixed		111	347159		73	491/493		82	574/607	
<i>big</i>	Buggy	12910	213	99528	80.75	182	599/783	97.78 (21.09%)	185	1039/1209	89.95 (11.39%)
	Fixed		214	99528		140	776/783		140	1185/1209	

using code corpus from [21]. Note that *small*, *median* and *big* refer to the number of tokens in each instance, rather than the number of instances in each dataset. More details about the numbers of tokens and instances are shown in Table 5.1. Each dataset is split into training, validation, and testing in a ratio of 8:1:1.

5.6.2 Model and Training Setting

In Bug2Fix, I first initialize token and context embedding layers with 512 and 256 dimensions, respectively. Then I stack 6 identical layers with 512 model dimensions for both encoder and decoder, the inner feed-forward layer has 2048 dimensions and I employ $h = 8$ parallel attention layers. The Adam optimizer [212] with $\beta_1 = 0.9$ and $\beta_2 = 0.98$ is leveraged to compute and update gradients, and label smoothing with ratio 0.1 is selected as loss function. In addition, I learning rate of 0.01, 2 and 0.2 for *small*, *median* and *big* datasets, respectively. For implementation, context abstraction is designed using *Scala implicit classes* and Bug2Fix is developed atop of OpenNMT-py [213]. Bug2Fix is trained on a machine with 4 Nvidia Titan RTX GPUs. For each dataset, I use an identical batch size for each model (e.g., 4096, 4096, 1024 for *small*, *median* and *big*, respectively).

5.6.3 Results of RQ1

Quality of Abstracted Data. As shown in Table 5.1, *small*, *median* and *big* datasets include 58350, 65455, 12910 instances, respectively. The BLEU in the Bug2Fix approximates Plaintext and is lower than BFP [22], which indicates the quality of data generated by Bug2Fix is closed to the original dataset. The average lengths per buggy and fixed code in these datasets are shown in the column of “Tokens”. The average lengths under BFP and Bug2Fix are reduced in comparison with Plaintext because both compress the datasets; while Bug2Fix can achieve balanced code length by a trade-off between compressing the most common pair of consecutive tokens by TPE and the reserved idioms and special separators. As to the vocabulary size of each dataset (showed by the “Vocabulary” column), it has been reduced by an order of magnitude (*e.g.*, from 179759 (Plaintext) to 433 (BFP) and 504 (Bug2Fix) on *small* dataset, respectively. In the implementation, I use a shared vocabulary for buggy and fixed code corpora (*i.e.*, combining their vocabularies) to overcome the out-of-vocabulary problem. The numbers separated by a slash marker in the “Vocabulary” column denote the original size and shared size, respectively, for buggy or fixed code. The “BLEU” column shows a BLEU score between buggy and fixed code corpora and the relative performance degradation over the BLEU score of Plaintext (if present and wrapped by parentheses). The lower degradation values in Bug2Fix show that Bug2Fix outperforms BFP significantly in all three datasets. Bug2Fix’s BLEU scores are closer to the ones of Plaintext, which means their data qualities are similar.

Model Behaviors. The *success ratio* of three models (*i.e.*, Plaintext (Transformer), BFP (Transformer), and Bug2Fix (Transformer)) on three datasets are shown in Table 5.2, 5.4 and 5.6, respectively. For each dataset, the *success* are shown an increasing trend over beams size from 1 to 50, except for Plaintext (Transformer) on *median* dataset. For example, Bug2Fix (Transformer) can fix 11.31%, 4.61%, and 25.19% of bugs with the only attempt successfully in *small*, *median* and *big*

Table 5.2: *Success Ratio* achieved by models on *small* dataset.

Beam Size s	Plaintext (Transformer)	BFP (Transformer)	Bug2Fix (Transformer)	SequenceR [21]	BFP-RNN [22]	Bug2Fix
1	8.28%	9.40%	11.31%	3.02%	10.57%	14.00%
10	25.98%	36.80%	40.00%	20.05%	37.00%	41.65%
20	29.91%	42.02%	46.00%	25.45%	44.15%	48.96%
30	31.91%	44.99%	49.00%	28.00%	47.68%	52.25%
40	33.28%	47.20%	51.00%	29.85%	49.84%	54.65%
50	34.10%	48.60%	53.00%	30.75%	51.29%	56.16%

datasets respectively, and 53%, 33.45%, and 46.28% of perfect predictions when beam sizes is 50. Bug2Fix (Transformer) outperforms the other models regardless of beam size and datasets, while Plaintext (Transformer) performs the worst. Plaintext (Transformer) fixes a few bugs, especially for the *median* dataset.

Summary of RQ1: Overall, Bug2Fix significantly outperforms the other two models. For data quality, the relative BLEU degradation of Bug2Fix is much smaller than BFP on each dataset, while Bug2Fix has a bigger average length of individual instances and a larger vocabulary. Model behaviors indicate that Bug2Fix (Transformer) can perform smoothly and stably to learn bug-fix patterns from training datasets with the help of the proposed TPE and renaming mechanism according to semantics and structures of tokens.

5.6.4 Results of RQ2

The performance of Bug2Fix is evaluated against with two baseline models (*i.e.*, SequenceR and BFP-RNN) using *small*, *median* and *big* datasets.

Small dataset. In Table 5.2, SequenceR, BFP-RNN, and Bug2Fix can predict and fix 3.02%, 10.57%, and 14.00% of buggy code with the only attempt successfully. The *success* steadily

Table 5.3: Performance metrics on *small* dataset.

Beam Size s	SequenceR [21]		BFP-RNN [22]		Bug2Fix	
	Action Hit	Syntactic Correctness	Action Hit	Syntactic Correctness	Action Hit	Syntactic Correctness
1	18.00%	41.00%	16.00%	96.00%	50.00%	100.00%
10	26.00%	29.00%	22.00%	84.00%	49.00%	99.00%
20	28.00%	23.00%	22.00%	82.00%	49.00%	98.00%
30	29.00%	20.00%	21.00%	81.00%	49.00%	98.00%
40	29.00%	18.00%	21.00%	80.00%	48.00%	98.00%
50	29.00%	17.00%	21.00%	80.00%	48.00%	97.00%

Table 5.4: *Success Ratio* achieved by models on *median* dataset.

Beam Size s	Plaintext (Transformer)	BFP (Transformer)	Bug2Fix (Transformer)	SequenceR [21]	BFP-RNN [22]	Bug2Fix
1	0.05%	6.39%	4.61%	1.02%	1.42%	7.99%
10	0.31%	20.99%	21.33%	9.15%	16.94%	21.48%
20	0.00%	25.36%	26.75%	12.67%	21.70%	26.91%
30	0.00%	28.46%	29.64%	15.05%	24.34%	29.87%
40	0.00%	30.68%	31.57%	16.30%	26.04%	32.36%
50	0.00%	32.41%	33.45%	18.41%	27.67%	34.12%

increases when more candidate patches are generated by the models, to reach 30.75%, 51.29%, and 56.16% of perfect predictions when beam size s is 50. In Table 5.3, *syntactic correctness* of Bug2Fix is much higher than that of SequenceR and BFP-RNN, where the former is also less impacted by the increasing beam size. That is, *syntactic correctness* of Bug2Fix is only reduced by 3% from beams size 1 to 50, however, the other two models are lowered by 24% and 5%, respectively. The *action hit* of Bug2Fix is bigger and more stable than that of SequenceR and BFP-RNN, regardless of beam size.

Table 5.5: Performance metrics on *median* dataset.

Beam Size s	SequenceR [21]		BFP-RNN [22]		Bug2Fix	
	Action Hit	Syntactic Correctness	Action Hit	Syntactic Correctness	Action Hit	Syntactic Correctness
1	1.00%	97.00%	2.00%	92.00%	65.00%	97.00%
10	11.00%	35.00%	8.00%	83.00%	65.00%	92.00%
20	12.00%	32.00%	8.00%	81.00%	65.00%	91.00%
30	13.00%	30.00%	8.00%	81.00%	65.00%	91.00%
40	13.00%	29.00%	8.00%	81.00%	64.00%	90.00%
50	14.00%	28.00%	8.00%	80.00%	64.00%	90.00%

Table 5.6: *Success Ratio* achieved by models on *big* dataset.

Beam Size s	Plaintext (Transformer)	BFP (Transformer)	Bug2Fix (Transformer)	SequenceR [21]	BFP-RNN [22]	Bug2Fix
1	0.70%	1.09%	25.19%	8.76%	5.58%	48.25%
10	2.48%	7.05%	42.48%	10.54%	10.93%	51.36%
20	2.79%	9.15%	44.03%	10.78%	12.40%	51.67%
30	3.10%	10.16%	45.35%	10.93%	13.26%	51.82%
40	3.10%	10.93%	45.89%	11.01%	14.03%	51.90%
50	3.57%	12.40%	46.28%	11.01%	14.26%	51.90%

Median dataset. In Table 5.5, the *action hit* of Bug2Fix is around five times of SequenceR and eight times of BFP-RNN. The *syntactic correctness* of Bug2Fix is higher than that of SequenceR and BFP-RNN by 60% and 10%, respectively. In Table 5.4, Bug2Fix can fix 7.99% of bugs with only one attempt successfully, which is significantly better than SequenceR (*i.e.*, 1.02%) and BFP-RNN (*i.e.*, 1.42%). When beam size is 50, 34.12% of bugs can be repaired by Bug2Fix, which is around two times that of SequenceR and speeds up BFP-RNN by 23%.

Big dataset. Table 5.6 shows that Bug2Fix is significant powerful to repair bugs than SequenceR and BFP-RNN. Specifically, Bug2Fix can fix 48.25% of bugs when beam size is 1, which is higher

Table 5.7: Performance metrics on *big* dataset.

Beam Size s	SequenceR [21]		BFP-RNN [22]		Bug2Fix	
	Action Hit	Syntactic Correctness	Action Hit	Syntactic Correctness	Action Hit	Syntactic Correctness
1	27.00%	12.00%	20.00%	31.00%	70.00%	60.00%
10	24.00%	7.00%	20.00%	30.00%	58.00%	50.00%
20	24.00%	6.00%	20.00%	30.00%	59.00%	50.00%
30	24.00%	6.00%	20.00%	30.00%	60.00%	50.00%
40	24.00%	6.00%	20.00%	30.00%	60.00%	49.00%
50	24.00%	6.00%	20.00%	30.00%	61.00%	49.00%

than 8.76% (SequenceR) and 5.58% (BFP-RNN). In Table 5.7, the *action hit* of Bug2Fix is around two times greater than that of SequenceR and three times greater than that of BFP-RNN. The *syntactic correctness* of Bug2Fix is higher than SequenceR’s and BFP-RNN’s by over 45%, 20%, respectively.

Summary of RQ2: Overall, the experimental results indicate that Bug2Fix significantly outperforms sequenceR and BFP-RNN in terms of all metrics on each single beam size. Bug2Fix can achieve higher and more stable values of *syntactic correctness* than others in each dataset. For *big* dataset with longer sequences, Bug2Fix’s performance is much better than SequenceR and BFP-RNN, which means the model can process a much larger size of code. In addition, the experimental results shown in Table 5.2, 5.4 and 5.6 also indicate that Bug2Fix can fix more bugs than Bug2Fix (Transformer) with the help of multi-task learning of code translation and context-aware alignment, regardless of beam size used.

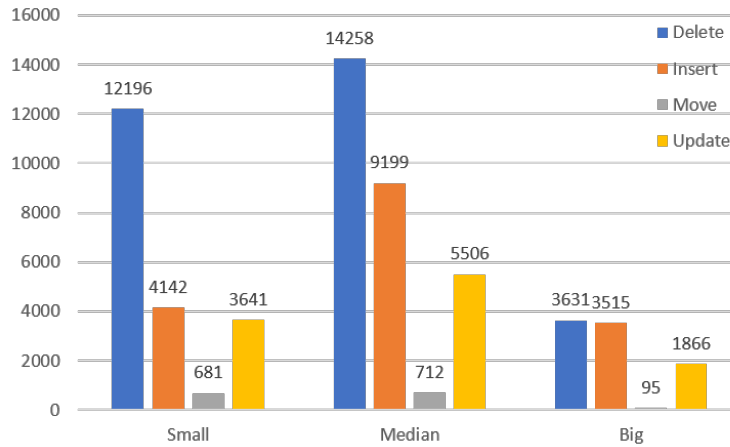


Figure 5.4: Statistics of actions performed to fix buggy code.

Table 5.8: The top-5 targets of each action in each dataset

Action	Target	<i>small</i>	<i>median</i>	<i>big</i>
Delete	VariableRead	3232/12196 (26.50%)	8288/14258 (58.13%)	2743/3631 (75.54%)
	TypeAccess	2425/12196 (19.88%)	1442/14258 (10.11%)	42/3631 (1.16%)
	Invocation	2071/12196 (16.98%)	1420/14258 (9.96%)	45/3631 (1.24%)
	FieldRead	913/12196 (7.49%)	629/14258 (4.41%)	93/3631 (2.56%)
	VariableWrite	92/12196 (0.75%)	631/14258 (4.43%)	529/3631 (14.57%)
Insert	TypeAccess	1489/4142 (35.95%)	4251/9199 (46.21%)	1250/3515 (35.56%)
	FieldRead	1361/4142 (32.86%)	3604/9199 (39.18%)	1882/3515 (53.54%)
	FieldWrite	89/4142 (2.15%)	538/9199 (5.85%)	191/3515 (5.43%)
	Literal	268/4142 (6.47%)	156/9199 (1.70%)	55/3515 (1.56%)
	Invocation	228/4142 (5.50%)	177/9199 (1.92%)	29/3515 (0.83%)
Move	Invocation	281/681 (41.26%)	263/712 (36.94%)	11/95 (11.58%)
	TypeAccess	162/681 (23.79%)	98/712 (13.76%)	31/95 (32.63%)
	FieldRead	43/681 (6.31%)	65/712 (9.13%)	36/95 (37.89%)
	BinaryOperator	44/681 (6.46%)	74/712 (10.39%)	9/95 (9.47%)
	ThisAccess	18/681 (2.64%)	12/712 (1.69%)	2/95 (2.11%)
Update	Parameter	2433/3641 (66.82%)	2138/5506 (38.83%)	657/1866 (35.21%)
	LocalVariable	449/3641 (12.33%)	2352/5506 (42.72%)	807/1866 (43.25%)
	TypeAccess	178/3641 (4.89%)	217/5506 (3.94%)	96/1866 (5.14%)
	CatchVariable	44/3641 (1.21%)	265/5506 (4.81%)	23/1866 (1.23%)
	Invocation	77/3641 (2.11%)	76/5506 (1.38%)	77/1866 (4.13%)

5.6.5 Results of RQ3

In this section, I discuss four kinds of actions performed by Bug2Fix for repairing program bugs. Due to the space limit, I only report experimental results of Bug2Fix when the beam size is set to 20. Figure 5.4 illustrates the numbers of the four actions used for program repair in each dataset. It shows that the model has the best performance on **Delete** action for all three datasets. **Insert**'s performance is also good. **Move** operation is the least performed among the four. Next, I examine the details of these four actions. Table 5.8 reports the top-5 targets of each action in each dataset. What all four actions have in common is that both *TypeAccess* and *Innovation* are among the top-5 targets though their positions (*i.e.*, percentages) vary from one to another. In addition, both **Delete** and **Move** have *FieldRead* in there lists. Look at five targets of an individual action, **Delete** of *big* dataset has a discrepancy as large as 74% and **Update** of *small* dataset is 65%. Whereas, smallest discrepancy (26%) occurs in **Delete** of *small* dataset. These results indicate that with a high consistency of the results in the three datasets, there is a high variation in each action in terms of the frequencies of targets being used. In addition, *TypeAccess* and *Invocation* are the most needed targets.

Summary of RQ3: Bug2Fix performs well on this buggy code that can be fixed by deleting or adding variables. The bugs related to type issues and method calls can also be fixed by Bug2Fix.

CHAPTER 6: CONTEXTUAL TRANSFORMER FOR INERTIAL NAVIGATION

6.1 Introduction and Motivation

Inertial navigation is a never-ending endeavor to estimate the states (*i.e.* position and orientation) of a moving subject (*e.g.* pedestrian) by using only IMUs attached to it. An IMU sensor, often a combination of accelerometers and gyroscopes, plays a significant role in a wide range of applications from mobile devices to autonomous systems because of its superior energy efficiency, mobility, and flexibility [27]. Technically, 3D angular velocity (ω) and 3D acceleration (α) provided by IMUs are subjected to bias and noise based on some sensor properties, as shown in Equation 6.1 & 6.2:

$$\omega_t = r_t^\omega + b_t^\omega + n_t^\omega \quad (6.1)$$

$$\alpha_t = r_t^\alpha + b_t^\alpha + n_t^\alpha \quad (6.2)$$

where r_t^ω and r_t^α are real sensor values measured by the gyroscope and accelerometer at timestamp t , respectively; b_t^ω and b_t^α are time-varying bias; n_t^ω and n_t^α are noise values, which usually follow a zero-mean gaussian distribution.

According to Newtonian mechanics [28], states (*i.e.* position and orientation) of a moving subject (*e.g.* pedestrian) can be estimated from a history of IMU measurements, as shown in Equation

6.3:

$$R_b^n(t) = R_b^n(t-1) \otimes \Omega(t) \quad (6.3a)$$

$$\Omega(t) = \exp\left(\frac{dt}{2} \omega(t-1)\right) \quad (6.3b)$$

$$v^n(t) = v^n(t-1) + \Delta(t) \quad (6.3c)$$

$$\Delta(t) = (R_b^n(t-1) \odot \alpha(t-1) - g^n)dt \quad (6.3d)$$

$$P^n(t) = P^n(t-1) + v^n(t-1)dt \quad (6.3e)$$

Here, the orientation $R_b^n(t)$ at timestamp t is updated with a relative orientation ($\Omega(t)$) between two discrete instants t and $t-1$ according to Equation 6.3a & 6.3b, where $\omega(t-1)$ measures proper angular velocity of an object at timestamp $(t-1)$ in the body frame (denoted by b) with respect to the navigation frame (denoted by n). R_b^n can be used to rotate a measurement $x \in [\omega, \alpha]$ from the body frame b to the navigation frame n , which is denoted by an expression $R_b^n \odot x = R_b^n \otimes x \otimes (R_b^n)^T$ where \otimes is a hamilton product between two quaternions. The navigation frame in the case is defined such that Z axis is aligned with earth's gravity g^n and the other two axes are determined according to the initial orientation of the body frame. In Equation 6.3c & 6.3d, velocity vector $v^n(t)$ is updated with its temporal difference $\Delta(t)$, which is obtained by rotating $\alpha(t-1)$ to the navigation frame using $R_b^n(t-1)$ and discarding the contribution of gravity forces g^n . Finally, positions $P^n(t)$ are obtained by integrating velocity in Equation 6.3e. Therefore, given current IMU measurements (*i.e.* α, ω), the new system states (*i.e.* P^n, v^n and R_b^n) can be obtained from the previous states using a function of f in Equation 6.4, where f represents transformations in Equation 6.3.

$$[P^n, v^n, R_b^n]_t = f([P^n, v^n, R_b^n]_{t-1}, [\alpha, \omega]_t) \quad (6.4)$$

Drawback and Solution: However, using IMUs for localization results in significant drift due to the bias and noise intrinsic to the gyroscope and accelerometer sensing can explode quickly in the double integration process. Using pure data-driven models with IMU measurements for Inertial Navigation has shown promising results in pedestrian dead-reckoning systems. To tackle

the problems of error propagation in Equation 6.4, I break the cycle of continuous integration and segment inertial measurements into independent windows, then leverage a sequence-to-sequence neural network architecture [117, 23, 174, 7] to predict velocities and positions from an input window m of IMU measurements, as shown in Equation 6.5.

$$[P^n, v^n]^{1:m} = \mathcal{F}_\theta(P_0^n, v_0^n, [R_b^n, \alpha, \omega]^{1:m}) \quad (6.5)$$

where \mathcal{F}_θ represents a latent neural system that learns the transformation from IMU samples to predict positions and velocities, where P_0^n, v_0^n are initial states.

In summary, the success of data-driven approaches for inertial navigation shows that with the aid of learned prior, the IMU sensor can provide enough information to infer low drift localization for inertial odometry. Obviously, the common shortcoming of this approach is a limitation of the training data and seen trajectories. The existing approaches reveal that similar to other data-driven models in the other domains, the generalization of the model across datasets is susceptible. The degradation of the model when trained on one dataset and tested on different datasets with varying underlying data collection settings requires more investigation. One obvious explanation is related to different bias and noise patterns in IMU data collected from different devices which prevent the model to generalize statistical motion patterns. In this project, **CTIN** is composed of two major components: *Spatial* and *Temporal* Encoders, to exploit high-level feature representations of IMU samples from the perspective of spatiality at a single timestamp and temporal across a sliding time window of measurements.

6.2 System Overall

The Attention-based architecture for inertial navigation is shown in Figure 6.1. It follows an encoder-decoder architecture using stacked self-attention and convolutional modules for *Spatial*

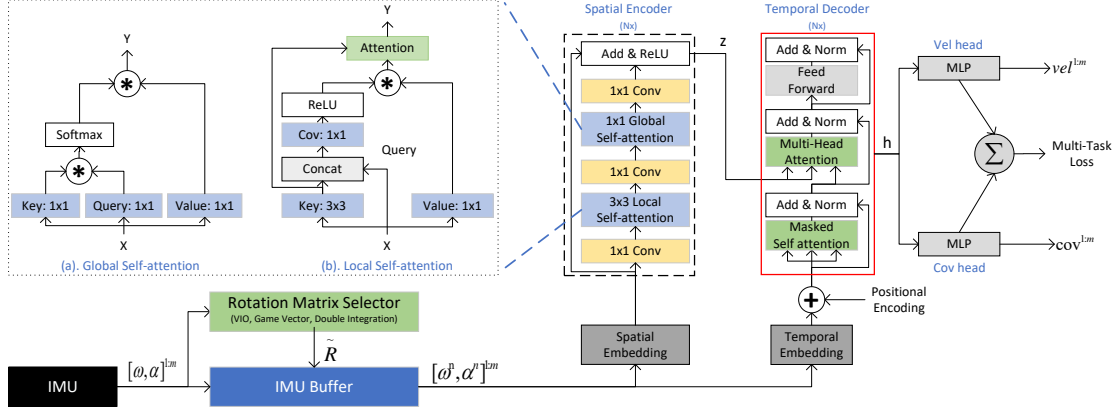


Figure 6.1: Overall workflow of the proposed contextual transformer model for inertial navigation.

Encoder, stacked self-attention and point-wise, fully connected layers for *Temporal Decoder*, shown in the right sub figures wrapped by dark dash line and red solid line, respectively. The whole workflow is depicted as follows:

- **Data Preparation.** Initially, an IMU sample is the concatenation of data from a gyroscope and accelerometer. To exploit temporal characteristics of IMU samples, I leverage a sliding window with size m to prepare datasets at timestamp t , denoted by $X_t^{1:m} = [x_{t-m+1}, \dots, x_t]$. Similarly, I adopt this rolling mechanism with the same window size to build the ground truth of velocities: $gt_{vel}^{1:m}$. Usually, IMU samples in each window are rotated from the body frame (*i.e.* ω^b, α^b) to the navigation frame (*i.e.* ω^n, α^n) using provided orientations. *Rotation Matrix Selector* is designed to select sources of orientation for training and testing automatically. Typically, I use the device orientation estimated from IMU for testing.
- **Embedding.** I need to compute feature representations for IMU samples before feeding them into the encoder and decoder. *Spatial Embedding* uses a 1D convolutional neural network followed by batch normalization and linear layers to learn spatial representations; *Temporal Embedding* adopts a 1-layer bidirectional LSTM model to exploit temporal infor-

mation, and then adds positional encoding provided by a trainable neural network.

- **Spatial Encoder.** The encoder comprises a stack of N identical layers, which maps an input sequence of $X_t^{1:m}$ to a sequence of continuous representations $z = (z_1, \dots, z_m)$. To capture spatial knowledge of IMU samples at each timestamp, I strengthen the functionality of the core bottleneck block in ResNet-18 [29] by replacing spatial convolution with a local self-attention layer and inserting a global self-attention module before the last 1×1 downsampling convolution (cf. in Section 6.3). All other structures, including the number of layers and spatial downsampling, are preserved. The modified bottleneck layer is repeated multiple times to form *Spatial Encoder*, with the output of one block being the input of the next one.
- **Temporal Decoder.** The decoder also comprises a stack of N identical layers. Within each layer, I first perform a masked self-attention sub-layer to extract dependencies in the temporal dimension. The masking emphasizes a fact that the output at timestamp t can depend only on IMU samples at timestamp less than t . Next, I conduct a multi-head attention sub-layer over the output of the encoder stack to fuse spatial and temporal information into a single vector representation and then pass through a position-wise fully connected feed-forward sub-layer. I also employ residual connections around each of the sub-layers, followed by layer normalization.
- **Velocity and Covariance.** Finally, two MLP-based branch heads regress 2D velocity ($vel_t^{1:m}$) and the corresponding covariance matrix ($cov_t^{1:m}$) using the input of h , respectively. Position can be obtained by the integration of velocity. The model of the covariance, denoted by $\Sigma : x \rightarrow \mathbb{R}^{2 \times 2}$ where x is a system state, can describe the distribution difference between ground-truth velocity and the corresponding predictions of them during training. Given that, the probability of a velocity y_v considering current system state x can be approximated by a

multivariate Gaussian distribution [214]:

$$p_c(y_v|x) = \frac{1}{\sqrt{(2\pi)^2|\boldsymbol{\Sigma}(x)|}} \times \exp\left(-\frac{1}{2}(y_v - \mathcal{F}_\theta(x))^T \boldsymbol{\Sigma}(x)^{-1} (y_v - \mathcal{F}_\theta(x))\right) \quad (6.6)$$

It is worthwhile to mention that I also leverage multi-task learning with uncertainty reduction to accomplish the desired performance (See details in Section 6.4).

6.3 Attention In Inertial Navigation

Attention can be considered as a query procedure that maps a query Q for a set of key-value pairs (K, V) to an output [7, 206], which is denoted by $ATT(Q, K, V) = \gamma(Q, K) \times V$. Typically, the output is computed as a sum of weighted values (V), where the weights $\gamma(Q, K)$ are computed according to a compatibility function of Q and K . There are two kinds of γ used in this paper [23, 215]: (1) I perform a *dot product* between Q and K , divides each resulting element by \sqrt{d} , and applies a *softmax* function to obtain the weights: $\gamma(Q, K) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)$ where d is the dimension size of vectors Q , K and V . (2) Inspired by Relation Networks [216], I investigate a form of *concatenation*: $\gamma(Q, K) = ReLU(W_\gamma[Q, K])$, where $[\cdot, \cdot]$ denotes concatenation and W_γ is a weight vector that projects the concatenated vector to a scalar. *Self-attention* networks compute a representation of an input sequence by applying attention to each pair of tokens from the sequence, regardless of their distance [7]. Technically, given IMU samples $X \in \mathbb{R}^{m \times d}$, I can perform the following transformation on X directly to obtain Q , K and V : $Q, K, V = XW_Q, XW_K, XW_V$, where $\{W_Q, W_K, W_V\} \in \mathbb{R}^{d \times d}$ are trainable parameters. Usually, these intermediate vectors are split into different representation subspaces at different positions (*i.e.* $h = 8, d_k = \frac{d}{h}$), *e.g.* $K = [K^1, \dots, K^h]$ with $K^i \in \mathbb{R}^{m \times d_k}$. For a subspace, the attention output is calculated by $head_i = ATT(Q^i, K^i, V^i)$. The final output representation is the concatenation of outputs generated by multiple attention

heads: $MultiHead(Q, K, V) = [head_i, \dots, head_h]$.

In this paper, the encoder and decoder rely entirely on an attention mechanism with different settings for embedding matrix $\{W_Q, W_K, W_V\}$ and γ to explore spatial and temporal knowledge from IMU samples.

Global self-attention in Encoder. It triggers the feature interactions across different spatial locations, as shown in Figure 6.1(a). Technically, I first transform X into Q , K , and V using three separated 1D 1×1 convolutions, respectively. After that, I obtain the global attention matrix (*i.e.* $\gamma(Q, K)$) between K and Q using a *Dot Product* version of γ . Finally, the final output Y is computed by $\gamma(Q, K) \times V$. In addition, I also adopt multi-head attention to jointly summarize information from different sub-space representations at different spatial positions.

Local self-attention in Encoder. Although performing a global self-attention over the whole feature map can achieve competitive performance, it not only scales poorly but also misses contextual information among neighbor keys. Because it treats queries and keys as a group of isolated pairs and learns their pairwise relations independently without exploring the rich contexts between them. To alleviate this issue, a body of research work [217, 218, 219, 220, 221] employs self-attention within the local region (*i.e.* 3×3 grid) to boost self-attention learning efficiently, and strengthen the representative capacity of the output aggregated feature map. In this paper, I follow up on this track and design a novel local self-attention for inertial navigation, as shown in Figure 6.1(b). In particular, I first employ 3×3 group convolution over all the neighbor keys within a grid of 3×3 to extract local contextual representations for each key, denoted by $C_1 = XW_{K,3 \times 3}$. After that, the attention matrix (*i.e.* $\gamma(Q, C_1)$) is achieved through a *concatenation* version of γ in which W_γ is a 1×1 convolution and Q is defined as X . Next, I calculate the attended feature map C_2 by $\gamma(Q, C_1) \times V$, which captures the global contextual interactions among all IMU samples. The final output Y is fused by an attention mechanism between local context C_1 and global context C_2 .

Multi-head attention in Decoder. I inherit settings from vanilla Transformer Decoder for attention mechanisms [7]. In other words, I take three separated linear layers to generate Q , K and V from X , respectively, and leverage a pairwise function of *Dot product* to calculate the attention matrix (*i.e.* $\gamma(Q, K)$). Finally, the final output Y is computed by $\gamma(Q, K) \times V$.

6.4 Jointly Learning Velocity and Covariance

I leverage multi-task learning with uncertainty reduction to improve learning efficiency and prediction accuracy of the two regression tasks: prediction of 2D velocity and its covariance. Inspired by [30, 31, 32, 33], I derive a multi-task loss function by maximizing the Gaussian likelihood with uncertainty [222]. First, I define the likelihood as a Gaussian with mean given by the model output as $p_u(y|\mathcal{F}_\theta(x)) = \mathcal{N}(\mathcal{F}_\theta(x), \delta^2)$, where δ is an observation noise scalar. Next, I derive the model’s minimization objective as a Negative Log-Likelihood (NLL) of two model outputs y_v (velocity) and y_c (covariance): $\mathcal{L}(\mathcal{F}_\theta, \delta_v, \delta_c)$

$$\begin{aligned}
&= -\log(p_u(y_v, y_c | \mathcal{F}_\theta(x))) \\
&= -\log(p_u(y_v | \mathcal{F}_\theta(x)) \times p_u(y_c | \mathcal{F}_\theta(x))) \\
&= -(\log(p_u(y_v | \mathcal{F}_\theta(x))) + \log(p_u(y_c | \mathcal{F}_\theta(x)))) \\
&= -(\log(\mathcal{N}(y_v; \mathcal{F}_\theta(x), \delta_v^2)) + \log(\mathcal{N}(y_c; \mathcal{F}_\theta(x), \delta_c^2))) \tag{6.7} \\
&\propto \underbrace{\frac{\|y_v - \mathcal{F}_\theta(x)\|^2}{2\delta_v^2} + \log \delta_v}_{\text{Velocity}} + \underbrace{\frac{\|y_c - \mathcal{F}_\theta(x)\|^2}{2\delta_c^2} + \log \delta_c}_{\text{Covariance}} \\
&= \frac{1}{2\delta_v^2} \mathcal{L}_v + \frac{1}{2\delta_c^2} \mathcal{L}_c + \log \delta_v \delta_c
\end{aligned}$$

where δ_v and δ_c are observation noises for velocity and covariance, respectively. Their loss functions are denoted by \mathcal{L}_v and \mathcal{L}_c , and depicted as follows:

Integral Velocity Loss (IVL, \mathcal{L}_v). Instead of performing mean square error (MSE) between pre-

dicted velocity (\hat{v}) and the ground-truth value (v), I first integrate predicted positions from \hat{v} (cf. Equation 6.3e), and then define an L2 norm against the ground-truth positional difference within the same segment of IMU samples, denoted by \mathcal{L}_v^p . In addition, I calculate cumulative error between \hat{v} and v , denoted by \mathcal{L}_v^e . Finally, \mathcal{L}_v is defined as $\mathcal{L}_v^p + \mathcal{L}_v^e$.

Covariance NLL Loss (CNL, \mathcal{L}_c). According to the covariance matrix in Equation 6.6, I define the Maximum Likelihood loss as the NLL of the velocity with consideration of its corresponding covariance Σ :

$$\begin{aligned}
 \mathcal{L}_c &= -\log(p_c(y_v|x)) \\
 &= \frac{1}{2}(y_v - f(x))^T \Sigma(\mathbf{x})^{-1} (y_v - f(x)) + \frac{1}{2} \ln |\Sigma(\mathbf{x})| \\
 &= \frac{1}{2} \|y_v - f(x)\|_{\Sigma(\mathbf{x})}^2 + \frac{1}{2} \ln |\Sigma(\mathbf{x})|
 \end{aligned} \tag{6.8}$$

There is a rich body of research work to propose various covariance parametrizations for neural network uncertainty estimation [31, 214]. In this study, I simply define the variances along the diagonal, which are parametrized by two coefficients of a velocity.

6.5 Experiment and Evaluation

I evaluate CTIN on five datasets against four representative prior research works. CTIN was implemented in Pytorch 1.7.1 [223] and trained using Adam optimizer [224]. During training, early stopping with 30 patience [225, 226] is leveraged to avoid overfitting according to model performance on the validation dataset. To be consistent with the experimental settings of baselines, I conduct both training and testing on NVIDIA RTX 2080Ti GPU.

Table 6.1: Description of public datasets used for evaluation of navigation models.

Dataset	Year	IMU Carrier	Sample Frequency	N ^e of Subjects	N ^e of Sequences	Ground Truth	Motion Context	Source
RIDI	2017	Lenovo Phab2 Pro	200 Hz	10	98	Google Tango phone	Four attachments: leg pocket, bag, hand, body	Public [135]
OxIOD	2018	iPhone 5/6, 7 Plus, Nexus 5	100 Hz	5	158	Vicon	Four attachments: handheld, pocket, handbag, trolley	Public [227]
RoNIN	2019	Galaxy S9, Pixel 2 XL	200 Hz	100	276	Asus Zenfone AR	Attaching devices naturally	Public [133]
IDOL	2020	iPhone 8	100 Hz	15	84	Kaarta Stencil	Attaching devices naturally	Public [134]
CTIN	2021	Samsung Note, Galaxy	200 Hz	5	100	Google ARCore	Attaching devices naturally	Collected by the own and will be released soon

6.5.1 Dataset

As shown in Table 6.1, all selected datasets with rich motion contexts (*e.g.* handheld, pocket, and leg) are collected by multiple subjects using two devices: one is to collect IMU measurements and the other provides ground truth, like position and orientation. All datasets are split into training, validation, and testing datasets in a ratio of 8:1:1. For testing datasets except in CTIN, there are two sub-sets: one for subjects that are also included in the training and validation sets, the other for unseen subjects.

Data Description and Acquisition. For these open-source datasets, I developed data loaders following the protocol in the RoNIN project [133] to load and prepare training/testing datasets. To collect the CTIN dataset, I use the two-device framework for IMU and six-degrees-of-freedom ground truth data acquisition. One device is used to capture IMU data and the other device is used to collect Google ARCore poses (translation and orientation). I use Samsung Galaxy devices in all the sensory experiments. Loop closure measurement is performed before each sensory experiment to ensure high-quality ground truth poses with low drift. An in-house Android application is installed on the devices for IMU data measurements. I use the calibrated IMU data from the device and further remove the offset from acceleration and gyro data through the sensory data in the table test experiment. The IMU data and ARCore data are captured at 200 HZ and 40 HZ, respectively, which leads to spatial and temporal alignment issues. To resolve them, the device system clock is

used as the timestamp for sensor events and time synchronization. ARCore data is interpolated at 200 HZ to synchronize the IMU and ARCore devices. For spatial alignment IMU data, ARCore data have to be represented in the same coordinate system. The camera and IMU local coordinate systems are aligned using the rotation matrix estimated by Kalibr toolbox [228]. The data is captured by 5 subjects and it includes various motion activities constitutes from walking and running. For each sequence, a subject moves for 2 to 10 minutes. The IMU device is mounted to the chest by a body harness and the ARCore device is attached to the hand to have a clear line of sight.

Data Preparation. During training, I use a sliding window ($N=200$) with an overlapping step size (20 for OxIOD, 50 for RIDI, and 10 for the rest of datasets) on each sequence to prepare input 6D IMU samples, ground truth 2D velocities, and 2D positions. In addition, a random shift is applied to a sliding window to enhance the robustness of the model to the indexing of sliding windows. Since ground truth data are provided in the navigation frame and the network can capture a motion model concerning the gravity-aligned IMU frame, IMU samples in each window are rotated from the IMU body frame to the navigation frame using device orientations at beginning of the window. In this study, the navigation frame is defined that the Z axis aligned with the negation of the gravity axis, and a coordinate frame augmentation agnostic to the heading in the horizontal frame is applied. This will indirectly provide the gravity information to the network, while augmentation of the sample by rotating around the Z axis in the horizontal plane would remove heading observability as it is theoretically unobservable to the data-driven model and the model should be invariant to rotation around the Z axis.

In this study, I design a component of *Rotation Matrix Selector* to choose orientation sources automatically for training, validation, and testing. For the RIDI dataset, I use the orientation estimated from IMU for training, validation, and testing; For the OxIOD dataset, I use ground-truth orientations from Vicon during training/validation, and Euler Angle from the device for testing, because of significant erroneous accuracy of estimated orientations. For the RoNIN dataset, I follow up on

the same procedures in the RoNIN project to choose orientations for training and testing. That’s, estimated orientations are used for testing; during training/validation, estimated orientations are selected if the end-sequence alignment error is below 20 degrees, otherwise, orientations from ground-truth are chosen to minimize noise during training. For IDOL and CTIN datasets, I use orientations from ground truth during training, validation, and testing. In addition to using the uncertainty reduction strategy to train the model, I also increase the robustness of the network against IMU measurements noise and bias by random perturbation of samples, since these perturbations can decrease the sensitivity of the network to input IMU errors. The additive bias perturbations for acceleration and gyroscope data are different. The additive sample bias for acceleration and gyroscope is sampled uniformly from the interval $[-0.2, 0.2] m/s^2$ and $[-0.05, 0.05] rad/s$ for each sample, respectively. The experimental results demonstrate that CTIN can be more generalized than other baselines to wider use cases or other datasets.

6.5.2 Baseline

The selected baseline models are listed below:

- *Strap-down Inertial Navigation System (SINS)*: The subject’s position can be obtained from double integration of linear accelerations (with earth’s gravity subtracted). To this end, I need to rotate the accelerations from the body frame to the navigation frame using device orientations and perform an integral operation on the rotated accelerations twice to get positions [126].
- *Pedestrian Dead Reckoning (PDR)*: I leverage Adaptive¹ to detect foot-steps and update positions per step along the device heading direction. I assume a stride length of 0.67m/step.

¹An Adaptive Jerk Pace Buffer Step Detection Algorithm

- *Robust IMU Double Integration (RIDI)*: I use the original implementation [135] to train a separate model for each device attachment in RIDI and OxIOD datasets. For the rest of the datasets, I train a unified model for each dataset separately, since attachments during data acquisition in these datasets are mixed.
- *Robust Neural Inertial Navigation (RoNIN)*: I use the original implementation [133] to evaluate all three RoNIN variants (*i.e.* R-LSTM, R-ResNet, and R-TCN) on all datasets.

6.5.3 Evaluation Metrics

Usually, positions in trajectory can be calculated by performing integration of velocity predicted by CTIN. The major metric used to evaluate the accuracy of positioning is a Root Mean Squared Error (RMSE) with various definitions of estimation error: $RMSE = \sqrt{\frac{1}{m} \sum_{t=1}^m \| E_t(x_t, \tilde{x}_t) \|^2}$, where m means the number of data points; $E_t(x_t, \tilde{x}_t)$ represents an estimation error between a position (*i.e.* x_t) in the ground truth trajectory at timestamp t and its corresponding one (*i.e.* \tilde{x}_t) in the predicted path. In this study, I define the following metrics [229]:

- **Absolute Trajectory Error (ATE)** is the RMSE of estimation error: $E_t = x_t - \tilde{x}_t$. The metric shows a global consistency between the trajectories and the error is increasing by the path length.
- **Time-Normalized Relative Traj. Error (T-RTE)** is the RMSE of average errors over a time-interval window span (*i.e.* $t_i = 60$ seconds in the case). The estimation error is defined formally as $E_t = (x_{t+t_i} - x_t) - (\tilde{x}_{t+t_i} - \tilde{x}_t)$. This metric measures the local consistency of the estimated and ground truth path. Formally, the error for the timestamp t with the interval of size m The metric measures the squared root distance between the displacements of positioning across a window of specified size.

Table 6.2: Overall Trajectory Prediction Accuracy. The best result is shown in bold font.

Dataset	Test Subject	Metric	Performance (meter)							Perf. Improvement		
			SINS	PDR	RIDI	RoNIN			CTIN	CTIN improvement over RoNIN		
						R-LSTM	R-ResNet	R-TCN		R-LSTM	R-ResNet	R-TCN
RIDID	Seen	ATE	6.34	22.76	8.18	2.55	2.33	3.25	1.39	45.36%	40.10%	57.13%
		T-RTE	8.13	24.89	9.34	2.34	2.36	2.64	1.99	15.00%	15.78%	24.80%
		D-RTE	0.52	1.39	0.97	0.16	0.16	0.17	0.11	32.47%	32.26%	35.91%
	Unseen	ATE	4.62	20.56	8.18	2.78	1.97	2.06	1.86	33.07%	5.40%	9.68%
		T-RTE	4.58	31.17	10.51	2.95	2.47	2.43	2.49	15.66%	-0.70%	-2.36%
		D-RTE	0.36	1.19	1.09	0.15	0.14	0.14	0.11	28.00%	21.22%	22.72%
OxIOD	Seen	ATE	15.36	9.78	3.78	3.87	2.40	3.33	2.32	40.10%	3.52%	30.27%
		T-RTE	11.02	8.51	3.99	1.56	1.83	1.49	0.62	60.40%	66.27%	58.67%
		D-RTE	0.96	1.16	2.30	0.20	0.56	0.19	0.07	61.94%	86.67%	61.21%
	Unseen	ATE	13.90	17.72	7.16	5.22	3.51	6.16	3.34	35.90%	4.61%	45.69%
		T-RTE	10.51	17.21	7.65	2.65	2.51	2.61	1.33	50.00%	47.18%	49.15%
		D-RTE	0.89	1.10	2.62	0.29	0.49	0.24	0.13	55.57%	73.45%	45.48%
RoNIN	Seen	ATE	7.89	26.64	16.82	5.11	3.99	6.18	4.62	9.49%	-15.81%	25.23%
		T-RTE	5.30	23.82	19.50	3.05	2.83	3.27	2.81	7.70%	0.69%	13.91%
		D-RTE	0.42	0.98	4.99	0.22	0.19	0.20	0.18	18.94%	2.75%	10.15%
	Unseen	ATE	7.62	23.49	15.75	8.73	5.76	7.49	5.61	35.77%	2.60%	25.11%
		T-RTE	5.12	23.07	19.13	4.87	4.50	4.70	4.48	8.04%	0.42%	4.61%
		D-RTE	0.43	1.00	5.37	0.29	0.25	0.26	0.25	12.63%	0%	4.83%
IDOL	Seen	ATE	21.54	18.44	9.79	4.57	4.44	4.68	2.90	36.49%	34.63%	37.98%
		T-RTE	14.93	14.53	7.97	1.72	1.58	1.77	1.35	21.47%	14.54%	23.46%
		D-RTE	1.07	1.14	0.97	0.19	0.26	0.18	0.13	28.39%	48.21%	25.12%
	Unseen	ATE	20.34	16.83	9.54	5.60	3.81	5.89	3.69	34.19%	3.28%	37.40%
		T-RTE	18.48	15.67	9.07	1.99	1.67	2.21	1.65	16.73%	1.02%	25.30%
		D-RTE	1.36	1.31	1.04	0.20	0.22	0.20	0.15	25.36%	30.14%	25.52%
CTIN	Seen	ATE	5.63	12.05	4.88	2.22	2.39	2.02	1.28	42.25%	46.45%	36.68%
		T-RTE	5.34	16.39	4.21	2.10	2.01	1.73	1.29	38.54%	35.87%	25.55%
		D-RTE	0.50	0.79	0.18	0.11	0.16	0.11	0.08	28.91%	50.56%	24.61%

- **Distance Normalized Relative Traj. Error (D-RTE)** is the RMSE across all corresponding windows when a subject travels a certain distance d , like d is set to 1 meter in the case. The estimation error is given by $E_t = (x_{t+t_d} - x_t) - (\hat{x}_{t+t_d} - \hat{x}_t)$ where t_d is the time interval needed to traverse a distance of d .
- **Position Drift Error (PDE)** measures final position (at timestamp m) drift over the total distance traveled (*i.e.* traj._len): $(\|x_m - \hat{x}_m\|) / \text{traj_len}$

6.5.4 Overall Performance

Table 6.2 shows experimental trajectory errors across entire test datasets. It demonstrates that CTIN can achieve the best results on most datasets in terms of ATE, T-RTE, and D-RTE metrics, except for two cases in RoNIN and RIDI datasets. R-TCN can get a smaller T-RTE number than CTIN in the RIDI-unseen test case; R-ResNet reports the smallest ATE of 3.99 for RoNIN-seen. In particular, CTIN improves an average ATE on all seen test datasets by 34.74%, 21.78%, and 37.46% over R-LSTM, R-ResNet, and R-TCN, respectively; the corresponding numbers for all unseen test datasets are 34.73%, 3.97%, and 29.47%.

The main limitation of RoNIN variants (*i.e.* R-LSTM, R-ResNet, and R-TCN) is that they do not capture the spectral correlations across time series which hampers the performance of the model. Therefore, it is convincing that CTIN achieves better performance over these baselines. Table 6.2 also shows that CTIN generalizes well to unseen test sets, and outperforms all other models on test sets. PDR shows a persistent ATE due to the consistent and precise updates owing to the jerk computations. This mechanism leads to PDR failure on long trajectories. Over time, the trajectory tends to drift owing to the accumulated heading estimation and the drift would increase dramatically, which results in decentralized motion trajectory shapes. R-LSTM does not show satisfactory results over large-scale trajectories. The margin of the outperforms of CTIN compared to R-LSTM and R-TCN is notable. The results for SINS show a large drift that highlights the noisy sensor measurements from smartphones.

6.5.5 Ablation Study

In this section, I evaluate model behaviors, the effectiveness of the attention layer, and loss functions used in CTIN on all dataset.

Table 6.3: Models’ Evaluation Performance on CTIN dataset

Model	N ² of Params (10 ⁶)	GFLOP Per Second (10 ⁹)	Average GPU time (ms)	Trajectory Error (meter)		
				ATE	T-RTE	D-RTE
CTIN	0.5571	7.27	65.96	1.28	1.29	0.08
R-LSTM	0.2058	7.17	704.23	2.22	2.10	0.11
R-TCN	2.0321	33.17	19.05	2.02	1.73	0.11
R-ResNet	4.6349	9.16	75.89	2.39	2.01	0.16

Model Behaviors. As shown in Figures from 6.2 to 6.6, CTIN outperforms the three RoNIN variant models (*i.e.* R-LSTM, R-ResNet, R-TCN) significantly on CTIN, RIDI, OxIOD, and IDOL, and lightly better on RoNIN. Specifically, the blue line of CTIN in most sub-figures regarding trajectory errors is steeper than in other plots. Sub-figures in the right column show that CTIN and R-ResNet can obtain lower scores of *avg MSE Loss* between ground truth velocity and predicted one, and Position Drift Error (*PDE (%)*), than the other two models. However, the *PDE (%)* performance of CTIN is better than R-ResNet, which is consistent with the performance pattern shown in the *ATE* metric. For the RoNIN dataset, the best performance is in a tangle of CTIN and R-ResNet. RoNIN is a group of 276 sequences, which is collected by 100 different subjects who perform various motion activities as well. Technically, this dataset should be more comprehensive than others. Unfortunately, only 50% of the dataset is released by authors, and these 138 ($=276 \times 50\%$) sequences may be gathered by total different 100 subjects, which leads to a significant difference in motion context, and various IMU sensor bias and noise. Therefore, it is difficult for CTIN to learn repeated and shared patterns from this undesired dataset. During training, I also perform random perturbations on the sensor bias, CTIN manifests less sensitivity to these input errors and achieves the desired performance.

For example, each plot shows the cumulative density function (CDF) of the chosen metric on the entire test datasets in Figure 6.2a. The blue line of CTIN is steeper than other plots, which indicates that CTIN shows significantly lower overall errors than all RoNIN variants for all presented met-

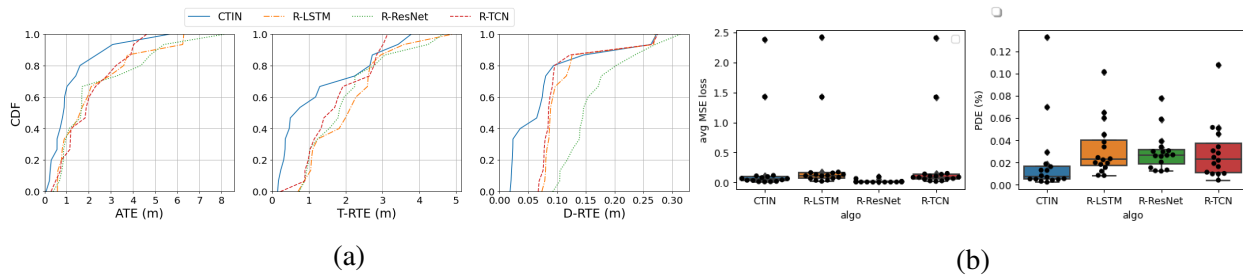


Figure 6.2: Performance Comparison of CTIN and RoNIN variant models on CTIN dataset

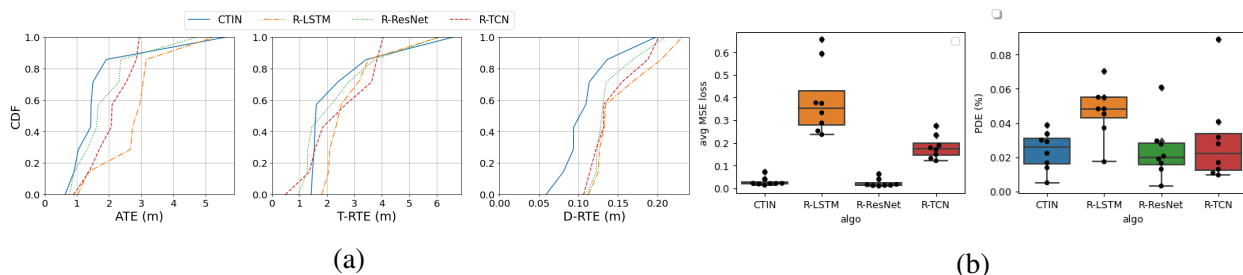


Figure 6.3: Performance Comparison of CTIN and RoNIN variant models on RIDI dataset

rics. As shown in Figure 6.2b, although CTIN’s overall MSE is higher than R-Resnet and smaller than R-LSTM and R-TCN, its position drift error (*i.e.* PDE (%)) is the smallest (*i.e.* the best). In Table 6.3, I show the number of parameters for each model, GFLOPs performed by GPU during testing and the average GPU execution time for testing a sequence of IMU samples (excluding the time to load data and generate trajectories after model prediction) and trajectory errors. Overall, CTIN possesses a significantly smaller number of parameters than R-TCN and R-ResNet, and more parameters than R-LSTM, achieving a competitive runtime performance with lower trajectory errors in a real deployment. Therefore, CTIN performs better than all RoNIN variants.

Attention Effectiveness. In this paper, I propose a novel attention mechanism to exploit local and global dependencies among the spatial feature space, and then leverage the multi-head attention

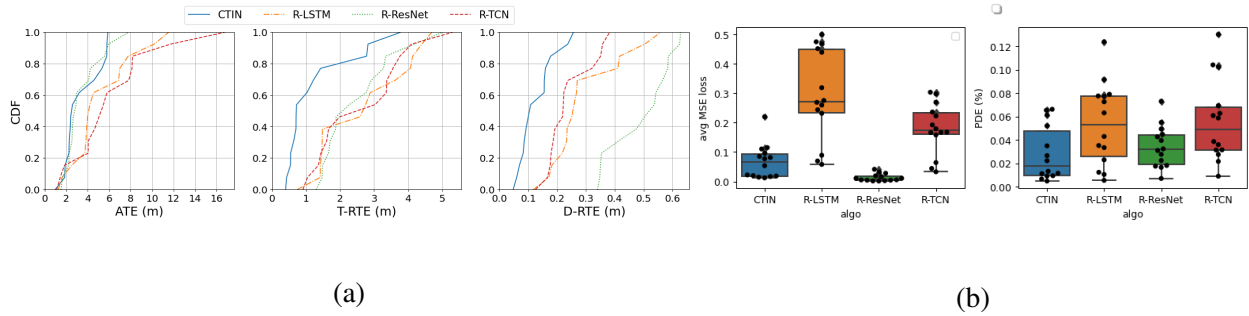


Figure 6.4: Performance Comparison of CTIN and RoNIN variant models on OXIOD dataset

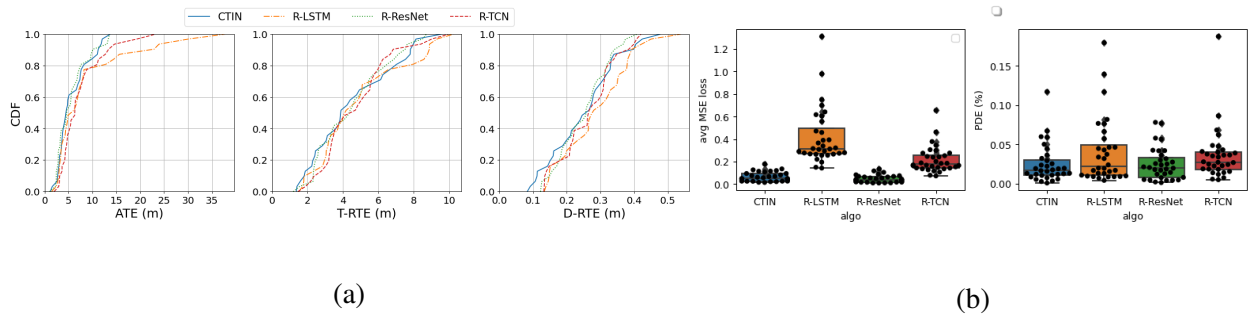


Figure 6.5: Performance Comparison of CTIN and RoNIN variant models on RoNIN dataset

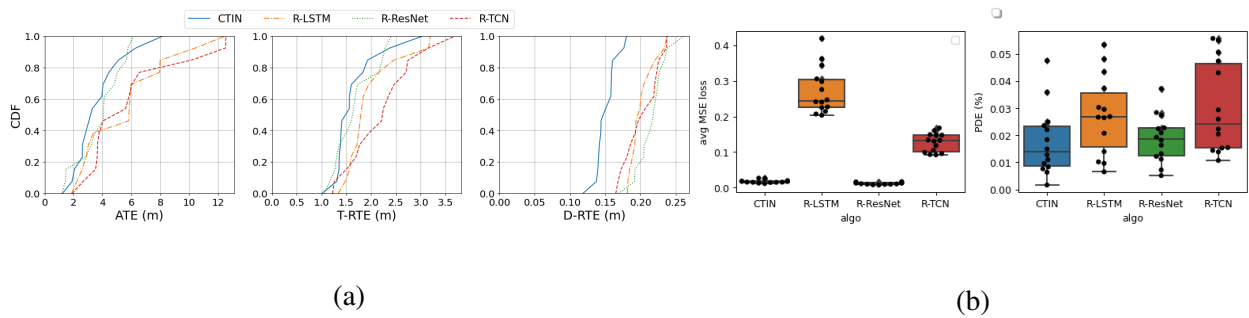


Figure 6.6: Performance Comparison of CTIN and RoNIN variant models on IDOL dataset

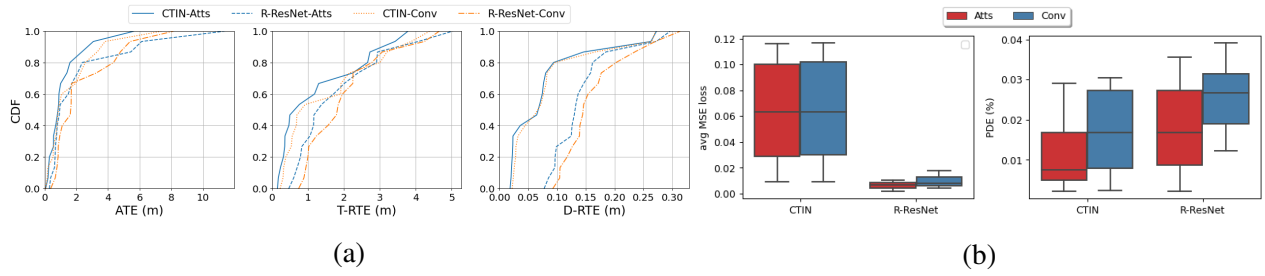


Figure 6.7: The effectiveness of proposed attention layers on CTIN dataset.

layer to combine spatial and temporal information for better accuracy of velocity prediction. To evaluate their effectiveness, I conduct a group of experiments using CTIN/R-ResNet and their variant without/with the capability of attention mechanism. The experimental results are shown in Figures from 6.7 to from 6.11. In each sub-figure, “*-atts” means CTIN or R-ResNet models with attention functionalities; “*-Conv” represents the models using a conventional spatial convolution instead. Overall, the effectiveness of the proposed attention mechanism has been demonstrated in all figures. For trajectory errors shown in the left column of figures, CTIN and R-ResNet capability of attention mechanism outperforms the ones with spatial convolution layers instead, respectively, especially for OxIOD and IDOL datasets. Attention-based models can achieve lower score of *Avg MSE Loss* and *PDE (%)*. For example, Figure 6.7a shows that *CTIN-Atts* and *R-ResNet-Atts* models outperform the models without attention layer. Furthermore, *CTIN-Atts* perform the best for all metrics, and the performance of *CTIN-Conv* is better than all R-ResNet variants. In Figure 6.7b, *CTIN-Atts* and *R-ResNet-Atts* have lower average MSE loss of velocity prediction and smallest PDE than *CTIN-Conv* and *R-ResNet-Conv*. Overall, CTIN and R-ResNet can benefit from the proposed attention mechanism.

Loss function. I expand the experiments on four extra datasets to evaluate the performance of multi-task loss (*i.e.* IVL+CNL) by performing a group comparison experiments using different

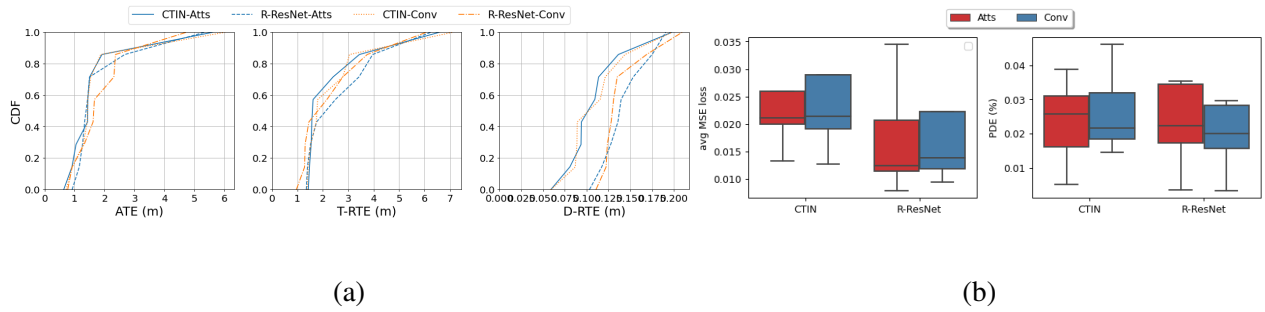


Figure 6.8: The effectiveness of proposed attention layers on RIDI dataset.

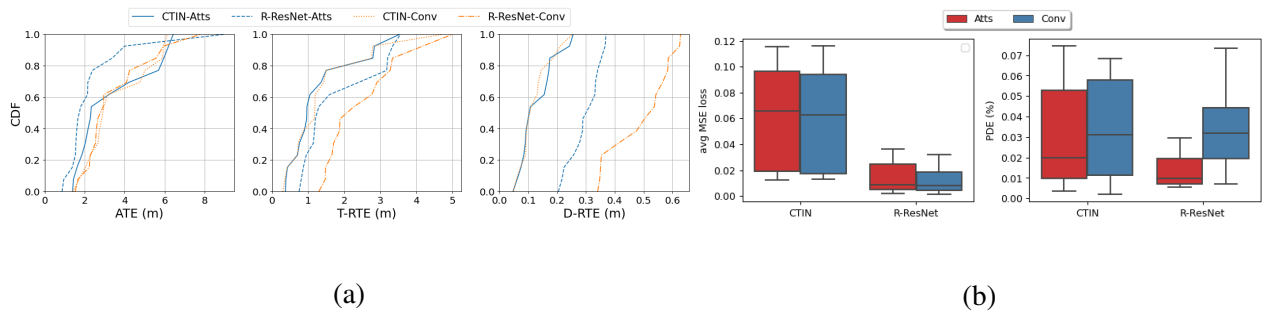


Figure 6.9: The effectiveness of proposed attention layers on OxIOD dataset.

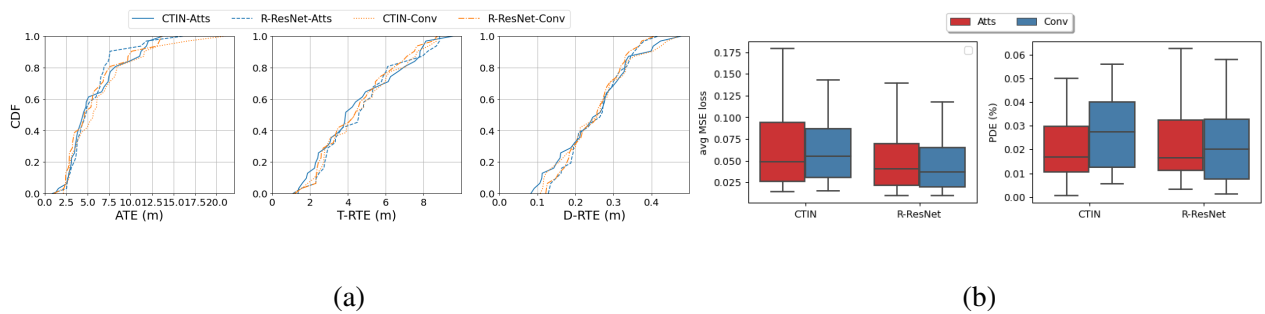


Figure 6.10: The effectiveness of proposed attention layers on RoNIN dataset.

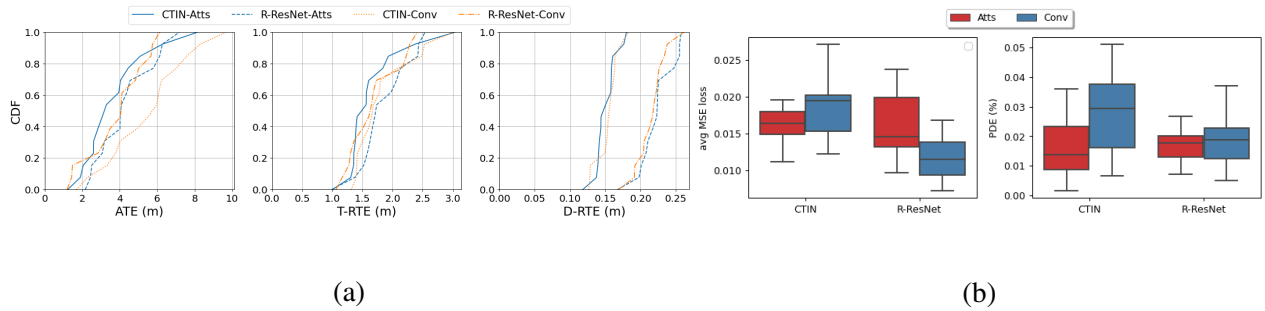


Figure 6.11: The effectiveness of proposed attention layers on IDOL dataset.

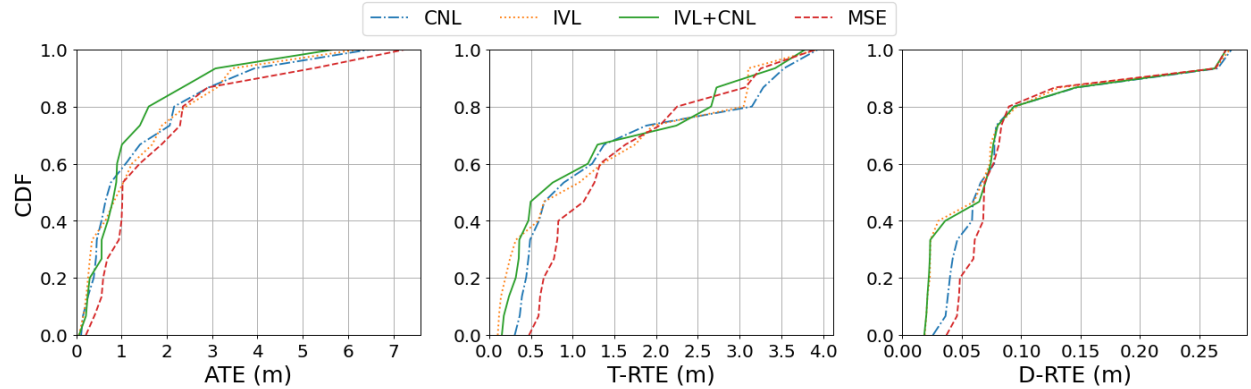


Figure 6.12: The performance of CTIN network with different loss functions evaluated on CTIN dataset.

loss functions, such as mean square error (MSE), Integral Velocity Loss (IVL) and Covariance NLL Loss (CNL), to train the models. Figures from 6.12 to 6.16 verifies the performance of CTIN with loss of IVL+CNL. Accordingly, these four-loss functions can achieve similar performance behaviors. CTIN with loss of IVL+CNL achieves better performance in RIDI OxIOD and IDOL. For RoNIN, the performance of CTIN with CNL is the best, and the model with IVL+CNL is better than the rest of the two loss functions. For example, As shown in Figure 6.12, CTIN with a loss of IVL+CNL achieves the best performance for ATE and D-RTE metrics.

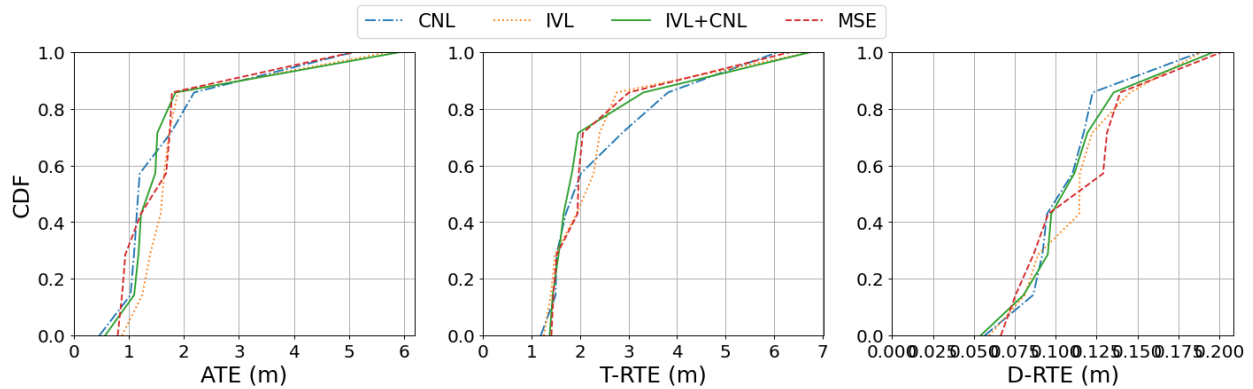


Figure 6.13: The performance of CTIN network with different loss functions evaluated on RIDI dataset.

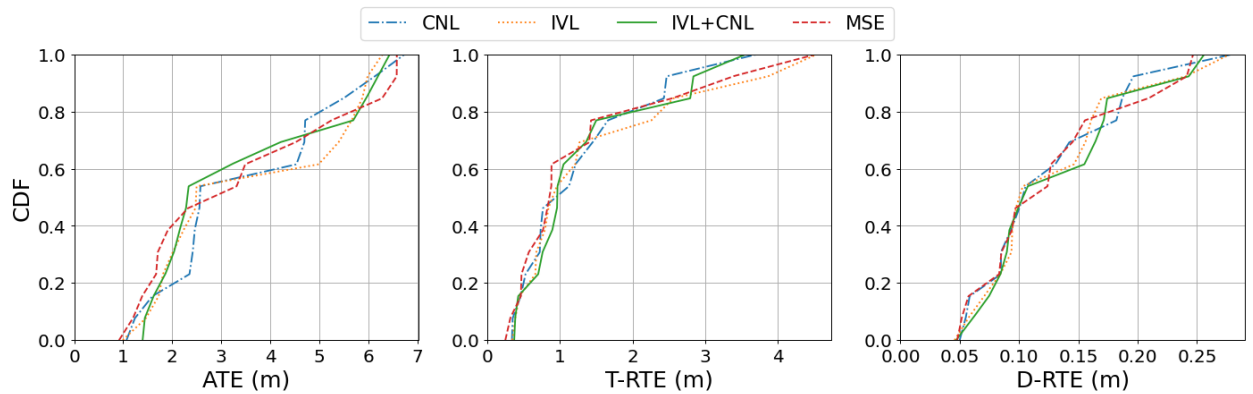


Figure 6.14: The performance of CTIN network with different loss functions evaluated on OxIOD dataset.

Selected Visualization of Trajectory. Two selected sequences visualization of reconstructed trajectories against the ground-truth for each dataset are shown from Figure 6.17 to Figure 6.20. I only show CTIN and three RoNIN variants methods. For each sequence, I mark it with the sequence name and the trajectory length, also report both ATE, T-RTE D-RTE, and PDE of selected approaches. The trajectory with blue color is generated by the models and the orange one is built

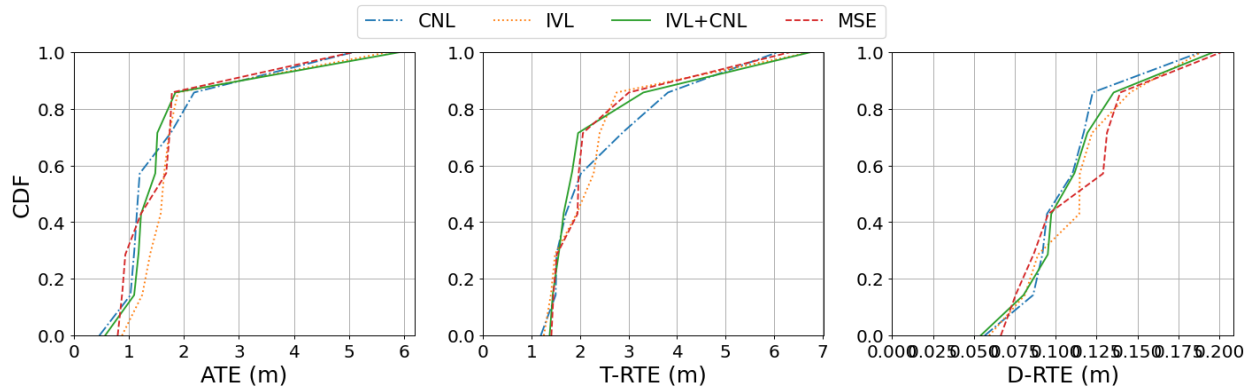


Figure 6.15: The performance of CTIN network with different loss functions evaluated on RoNIN dataset.

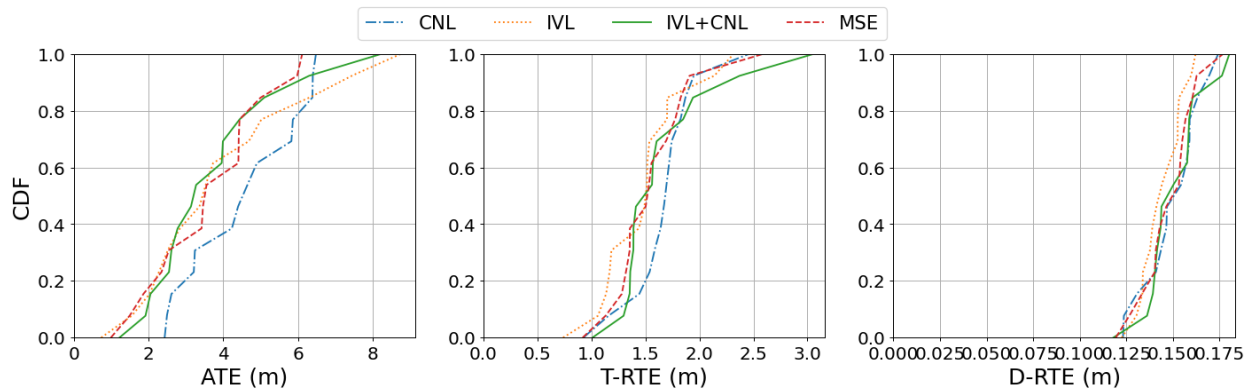


Figure 6.16: The performance of CTIN network with different loss functions evaluated on IDOL dataset.

from ground truth data. Positional errors are marked within each figure, where “AT”, “TR”, “DR”, and “PD” denote metrics of ATE, T-RTE, D-RTE, and PDE, respectively. Sub-figures in a row show the visualizations of a selected sequence (named by the title of the first sub-figure) between the ground truth trajectory and predicted ones generated by CTIN, R-LSTM, R-ResNet, and R-TCN, sequentially. Due to the uncertainty of predicted trajectories, there may be different shapes of ground truth trajectory for a sequence. For example in Figure 6.18, it looks like the

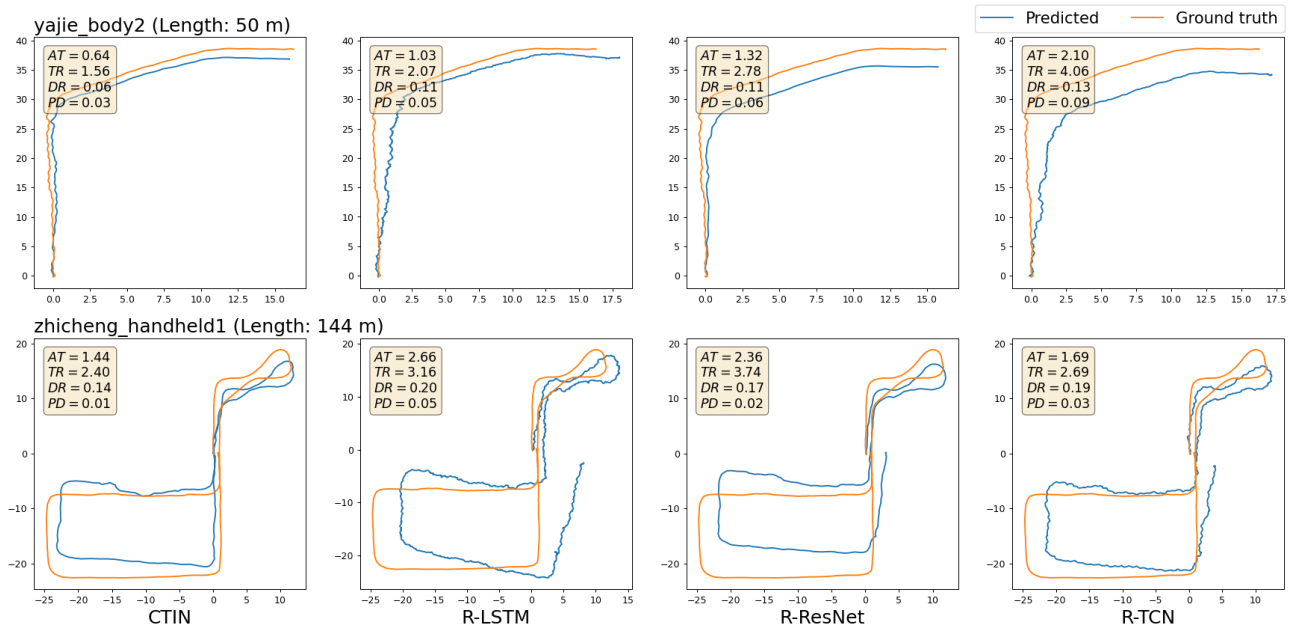


Figure 6.17: Selected visualizations of trajectories from CTIN and RoNIN variants models on the RIDI dataset.

shapes of ground truth trajectory for the sequence “handbag_data2_seq2 (Length: 494m)” are different because of different scales of axes. Actually, they are the same and use identical data to draw them.

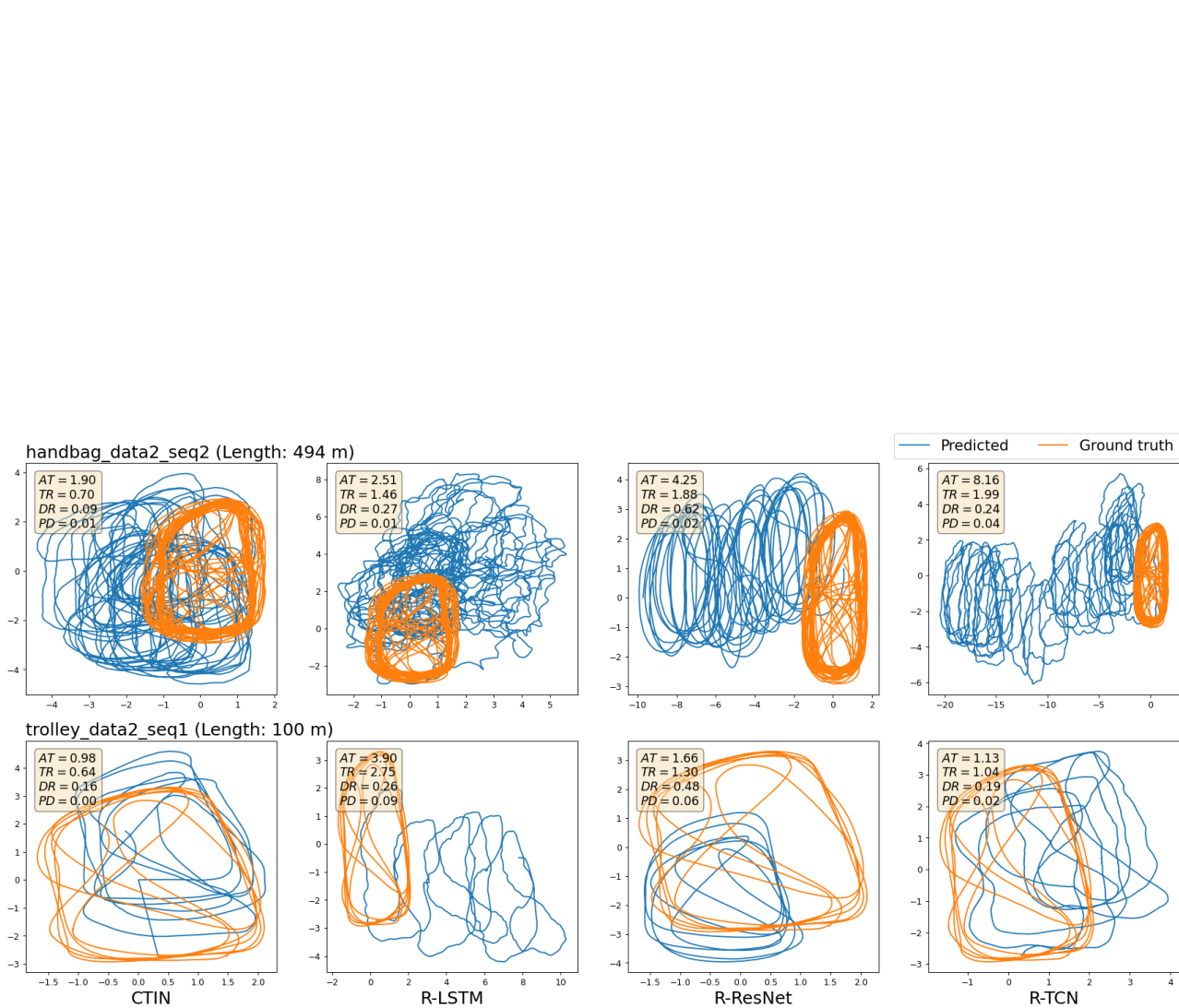


Figure 6.18: Selected visualizations of trajectories from CTIN and RoNIN variants models on OxIOD dataset.

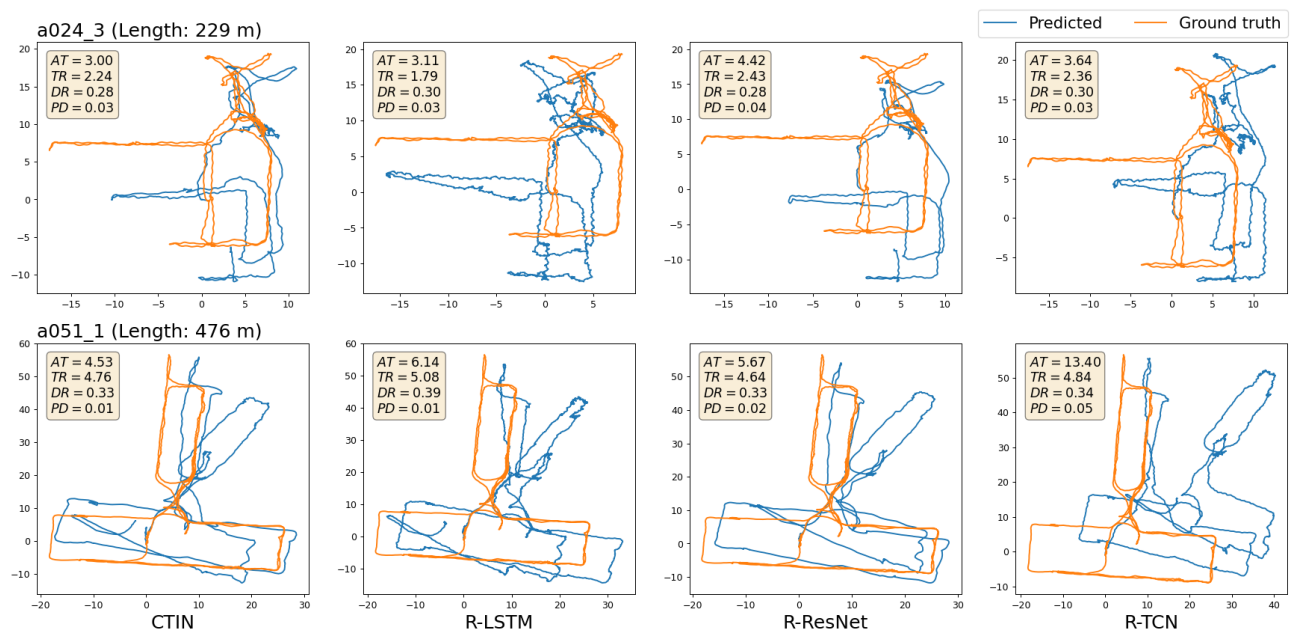


Figure 6.19: Selected visualizations of trajectories from CTIN and RoNIN variants models on the RoNIN dataset.

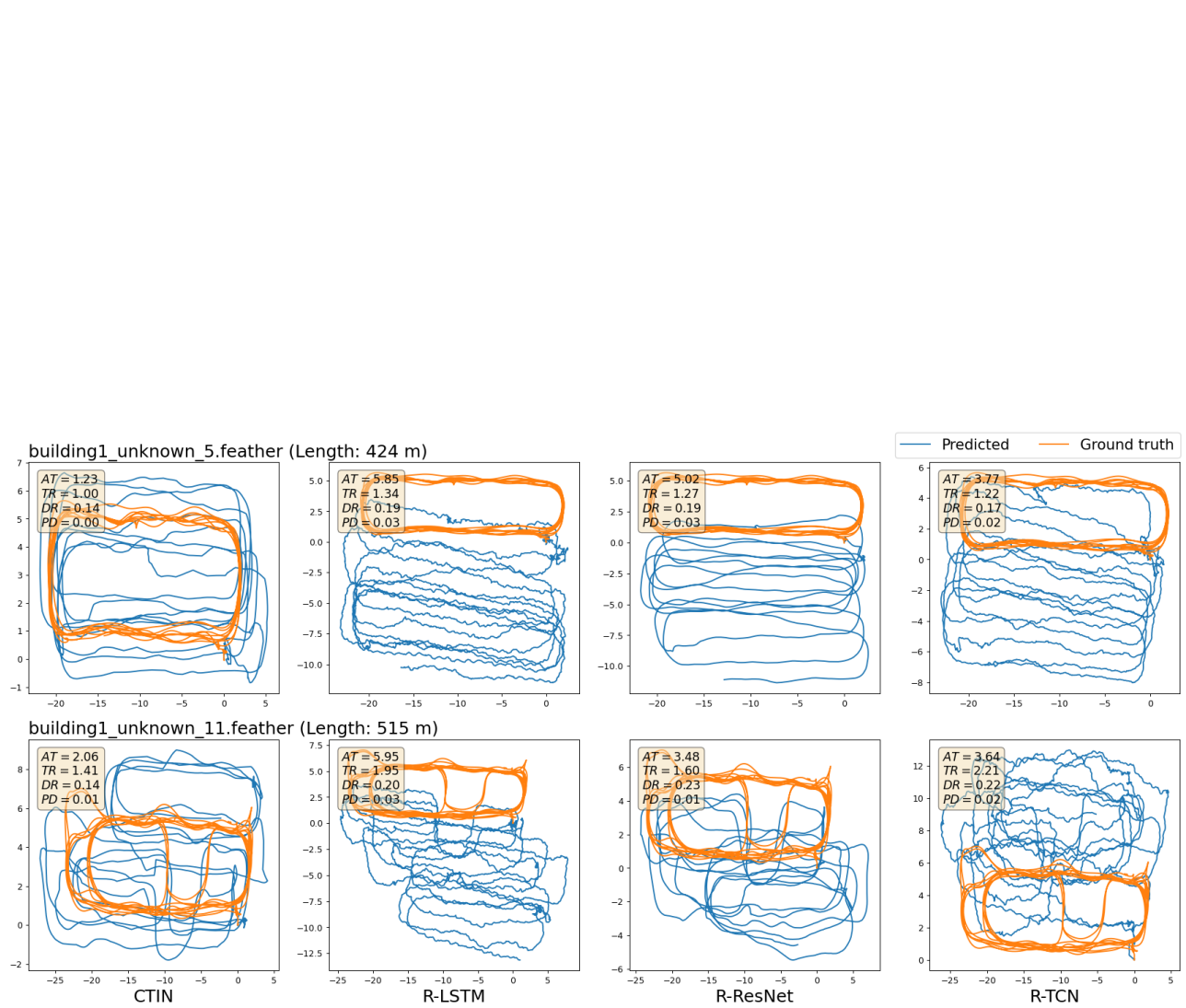


Figure 6.20: Selected visualizations of trajectories from CTIN and RoNIN variants models on the IDOL dataset.

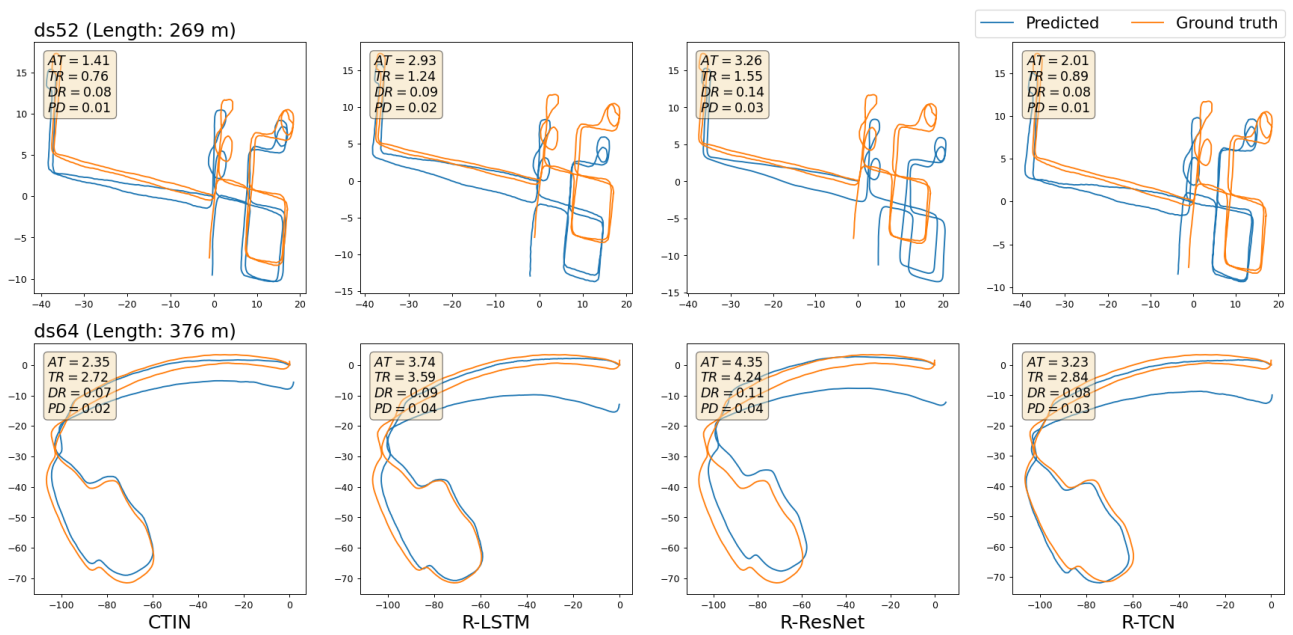


Figure 6.21: Selected visualizations of trajectories from CTIN and RoNIN variants models on CTIN seen dataset.

CHAPTER 7: CONCLUSION AND FUTURE WORK

7.1 Detecting Trends in Dynamic Social Networks

As social media networks are increasingly used to represent human activities in online environments, effective methods for detecting temporal nodal activities are needed. This research has developed and validated a new class of models that examine user interactions to support activity detection in dynamic social networks. The models incorporated information about agents' activities in the network and their interaction over time to detection at the next time step. The innovation of the models consists of a novel simulation of online social networks, accurate detection based not only on historical activities but also on agent interaction, and adaptive link addition or removal based on ongoing activity patterns.

7.1.1 Summary of Findings

The experimental findings show that the two interaction models outperformed the benchmark models significantly across different dates and across different temporal window sizes. PIM was also found to significantly outperform RIM across different window sizes in Dataset 2015, demonstrating the best accuracy among all four models when w ranges from 4 to 20. For a fixed window size, performances of RIM and PIM are not significantly different across dates. This finding shows that both PIM and RIM capture unique activity patterns that are complementary to each other. Behaviors exhibited in random interaction and preferential interaction both contribute to future activity levels detected by these models. Therefore, both patterns demonstrate a significant presence in user activities in the temporal networks. In addition, the two interaction patterns (preferential interaction and random interaction) do not have a significant difference in their contribution to user

activities. Such lack of clear distinction is not because their individual effects are not significant, but possibly because users adopt these behavior patterns selectively, contingent on the circumstances that they face. In other words, although the majority of users favor preferential interaction, most activities are characterized by random interaction as users make contingent choices to adjust to reality. Examining these contingencies can possibly reveal patterns of strategic behavioral adoption and its relationship with external factors.

Regarding evaluation metrics, the experimental results demonstrate significant advantages of SMAPE and LMAPE over RMSE. The results show that the linear assumption of RMSE often fails to characterize the differences in model performance, making it less capable to measure activities that exhibit primarily non-linear patterns. The findings show that SMAPE and LMAPE captured the random interaction and exponential activity patterns very well, and hence should be more suitable for measuring online social network activities than traditional approaches such as RMSE (e.g., Reference [27]).

Results of the correlation analysis show that the structural properties of social networks provide a simple and yet accurate approach to predicting model performance (measured by LMAPE and SMAPE). These properties allow linear projection of the performance values that are based on non-linear model prediction. However, the directions and intensity of correlation vary according to datasets and performance measures. Future research may examine in more detail these variations. As online social network activities tend to change dramatically over time, there is a strong potential to use these properties to support advanced analysis and understanding of temporal network activities.

7.1.2 Contributions and Limitations

This research should provide several contributions. The interaction models for temporal network prediction are shown to advance traditional models (e.g., References [57, 67]) by explicitly modeling agent interaction and by simulating network behavior. New experimental findings obtained in this research can be used to inform model development and performance. Moreover, the research developed a reusable implementation of temporal prediction for large networks in a social media community, thus contributing to standardizing the evaluation of temporal social networks [27]. The computational framework developed in this research provides a reusable graph construction and transformation pipeline that is not available in prior research [37]. The methods and findings should be useful to computer and information scientists, and intelligence experts (e.g., for cyber-surveillance [14]), social researchers (e.g., for public- and health-policy decision-making [42]), and business practitioners (e.g., business analytics [10, 12]), among others. There are several limitations to this research. The datasets used in this research came from one specific source (Twitter). Although different time spans of datasets and different analysis aspects were considered, the choice of data source may limit the generalizability of the findings. There is a lack of prior theoretical study of temporal nodal activity detection in online social networks. The identification of relevant theoretical and technical aspects was non-trivial in this research before the technical infrastructure was built. The resources available to collect and handle the data were limited, restricting additional analysis that could be performed.

There are several limitations to this research. The datasets used in this research came from one specific domain. Although different time spans of datasets and different analysis aspects were considered, the choice of the domain may limit the generalizability of the findings. There is a lack of prior theoretical study of temporal social network activity trend detection in the sharing economy. The identification of relevant theoretical and technical aspects was non-trivial in this

research before the technical infrastructure was built. The resources available to collect and handle the data were limited, restricting additional analysis that could be performed.

7.1.3 Future Directions

In the future, this research may be extended in several ways. Extending the temporal window w to study its impact on predictive performance using different models may help to characterize the domains being predicted for and the models being used. Multiple domains could be studied to examine their effect on model performance. Additional types of interaction and datasets may be modeled to capture their unique activity patterns. Social links other than geodesics used in BC calculation may be explored (e.g., direct links). Comparison of which behavioral assumptions work well in which predictive circumstances should advance understanding of model design and deployment.

7.2 Semantics-aware Optimizations for Big Data Applications

In this paper, I propose a semantics-aware optimization approach to assist programmers to develop and optimize an application interactively and semi-automatically. I propose three kinds of optimization strategies: cache management, operation reordering and element pruning. Element pruning is a static rule-based model and the other two are hybrid models using static and dynamic information. To get dynamic information about data and runtime system, the online phase is developed as a piggyback monitoring tool by integrating spark internal event component, metrics system and source code profiling tools. Extensive empirical results on several real-world benchmarks using Spark RDD APIs reveal that the approach achieves better performance on the optimized code than their original implementation.

In the future work, I will extend the optimization of operation reordering to *map* as well as other operations. So far SODA can only take care of filter and join reordering, and help programmers choose the right operation with acceptable performance. For example, *reduceByKey* can replace *groupByKey* to reduce shuffling data size. Another promising area is to add a growing number of performance-oriented constraints to the global cache management policy. For example, I can require that all datasets needed by an operation are persisted in memory simultaneously to gain better performance.

7.3 Graph Transformer for Automated Programming Repair

In this study, I introduce Bug2Fix, a novel Transformer model enhanced by the proposed context-aware attention for APR tasks. In particular, a novel *semantics-preserved, scope-oriented, and vocabulary-closed* context-aware abstraction approach is designed for pre-processing code corpus, and then *joint multi-task learning of code translation and context-aware alignment* is proposed for automatically detecting and repairing bugs. The extensive empirical evaluation suggests that Bug2Fix significantly outperforms the baseline models in terms of three research questions. Specifically, Bug2Fix can successfully predict the fixed code of 14.00%, 7.99%, and 48.25% of three different datasets with only one attempt, respectively. The success rate steadily improves over an increasing beam size, and it eventually achieves 56.16%, 34.12%, and 51.90%, respectively, when 50 candidate patches are considered. In addition, regardless of beam size used, the overall syntactic correctness of all patches are very high, *i.e.*, above 97%, 90%, 49% for the three datasets, respectively.

In future work, I would like to improve the proposed approach by overcoming a few limitations. Integrating more information from AST into the model is of great interest. For example, it is effective to use a token's position in AST, rather than its absolute position in the sequence. This

is because the tree-based positional information can help the model exploit and understand the code from the perspective of a tree structure. In addition, the semantic meaning of a token in AST, such as path context [202], can be used to design an attention mechanism in the model. Next, enhancing the model’s capability to process class even package is more desired. Expanding granularity from method to class or package level can accommodate more context for certain bugs, such as interprocedural issues. However, this may yield a more complex model and more difficulty in training.

7.4 Contextual Transformer For Inertial Navigation

In this paper, I propose CTIN, a novel robust contextual Attention-based model to regress accurate 2D velocity and trajectory from segments of IMU measurements. To this end, I first design a ResNet-based encoder enhanced by local and global self-attention layers to capture spatial contextual information from IMU measurements, which can guide the learning of an efficient attention matrix and thus strengthens the capacity of inertial representation. I further fuse these spatial representations with temporal knowledge by leveraging multi-head attention in the Transformer decoder. Finally, multi-task learning using uncertainty is leveraged to improve learning efficiency and prediction accuracy of 2D velocity. Through extensive experiments over a wide range of inertial datasets, CTIN is very robust and outperforms state-of-the-art models.

The main limitation of CTIN is to use 3D orientation estimation generated by the device (*e.g.* Game Vector), which can be inaccurate. In future work, I will extend CTIN with better orientation estimations. Secondly, although the proposed pipeline of CTIN achieves good accuracy on pedestrian inertial observations, the accuracy of CTIN on vehicle IMU data is not desirable due to the errors in the uncertainty of sensory data such as noisy sensory data, inhomogeneous offset values across devices, and variant environments.

LIST OF REFERENCES

- [1] K. Ashton *et al.*, “That ‘internet of things’ thing,” *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.
- [2] Y. Demchenko, P. Grosso, C. De Laat, and P. Membrey, “Addressing big data issues in scientific data infrastructure,” in *2013 International conference on collaboration technologies and systems (CTS)*. IEEE, 2013, pp. 48–55.
- [3] V. Subramanian, H. Ma, L. Wang, E.-J. Lee, and P. Chen, “Rapid 3d seismic source inversion using windows azure and amazon ec2,” in *2011 IEEE World Congress on Services*. IEEE, 2011, pp. 602–606.
- [4] H. Zhang, Z. Sun, Z. Liu, C. Xu, and L. Wang, “Dart: A geographic information system on hadoop,” in *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 2015, pp. 90–97.
- [5] W. Chung, B. Rao, and L. Wang, “Dynamic trend detection in us border security social-media networks,” *Simulation and Education Conference (I/ITSEC), In 2016 Interservice/Industry Training*, 2016.
- [6] —, “Interaction models for detecting nodal activities in temporal social media networks,” *ACM Transactions on Management Information Systems (TMIS)*, vol. 10, no. 4, pp. 1–30, 2019.
- [7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

- [8] W. Wei and K. M. Carley, “Measuring temporal patterns in dynamic social networks,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–27, 2015.
- [9] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson, “Modeling dynamic behavior in large evolving graphs,” in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 667–676.
- [10] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, “Graphx: A resilient distributed graph system on spark,” in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013.
- [11] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, 2008.
- [12] Apache, “Hadoop,” 2009.
- [13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, “Spark: Cluster computing with working sets.” *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.
- [14] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, “On the naturalness of software,” in *2012 34th International Conference on Software Engineering (ICSE)*, IEEE. IEEE Computer Society, 2012, pp. 837–847.
- [15] M. Bruch, M. Monperrus, and M. Mezini, “Learning from examples to improve code completion systems,” in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*. ACM, 2009, pp. 213–222.
- [16] V. Raychev, M. Vechev, and E. Yahav, “Code completion with statistical language models,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2014, pp. 419–428.

- [17] F. Liu, G. Li, B. Wei, X. Xia, Z. Fu, and Z. Jin, “A self-attentional neural architecture for code completion with multi-task learning,” in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 37–47.
- [18] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, “Neural detection of semantic code clones via tree-based convolution,” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 70–80.
- [19] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, “Duplicate bug report detection using dual-channel convolutional neural networks,” in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 117–127.
- [20] R. Gupta, S. Pal, A. Kanade, and S. Shevade, “Deepfix: Fixing common c language errors by deep learning,” in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 1345–1351.
- [21] Z. Chen, S. J. Kommrusch, M. Tufano, L.-N. Pouchet, D. Poshyvanyk, and M. Monperrus, “SequenceR: Sequence-to-sequence learning for end-to-end program repair,” *IEEE Transactions on Software Engineering*, 2019.
- [22] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, “An empirical study on learning bug-fixing patches in the wild via neural machine translation,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 4, pp. 1–29, 2019.
- [23] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

- [24] Y. Li, S. Wang, T. N. Nguyen, and S. Van Nguyen, “Improving bug detection via context-based code representation learning and attention-based neural networks,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–30, 2019.
- [25] J. Zhang, R. Xie, W. Ye, Y. Zhang, and S. Zhang, “Exploiting code knowledge graph for bug localization via bi-directional attention,” in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 219–229.
- [26] L. Wu, F. Li, Y. Wu, and T. Zheng, “Ggf: A graph-based method for programming language syntax error correction,” in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 139–148.
- [27] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen, “A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned,” in *Proceedings of the 14th international conference on information processing in sensor networks*, 2015, pp. 178–189.
- [28] M. Kok, J. D. Hol, and T. B. Schön, “Using inertial sensors for position and orientation estimation,” *arXiv preprint arXiv:1704.06053*, 2017.
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [30] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491.

- [31] W. Liu, D. Caruso, E. Ilg, J. Dong, A. I. Mourikis, K. Daniilidis, V. Kumar, and J. Engel, “Tlio: Tight learned inertial odometry,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5653–5660, 2020.
- [32] J. Yao, W. Xing, D. Wang, J. Xing, and L. Wang, “Active dropblock: Method to enhance deep model accuracy and robustness,” *Neurocomputing*, vol. 454, pp. 189–200, 2021.
- [33] Y. Yang, W. Xing, D. Wang, S. Zhang, Q. Yu, and L. Wang, “Aevrnet: Adaptive exploration network with variance reduced optimization for visual tracking,” *Neurocomputing*, vol. 449, pp. 48–60, 2021.
- [34] G. C. Kane, M. Alavi, G. Labianca, and S. P. Borgatti, “What’s different about social media networks? a framework and research agenda,” *MIS quarterly*, vol. 38, no. 1, pp. 275–304, 2014.
- [35] P. Wang, B. Xu, Y. Wu, and X. Zhou, “Link prediction in social networks: the state-of-the-art,” *Science China Information Sciences*, vol. 58, no. 1, pp. 1–38, 2015.
- [36] W. Chung and D. Zeng, “Social-media-based public policy informatics: Sentiment and network analyses of us immigration and border security,” *Journal of the Association for Information Science and Technology*, vol. 67, no. 7, pp. 1588–1606, 2016.
- [37] W. Fan and M. D. Gordon, “The power of social media analytics,” *Communications of the ACM*, vol. 57, no. 6, pp. 74–81, 2014.
- [38] D. Zeng, “Policy informatics for smart policy-making,” *IEEE Intelligent Systems*, vol. 30, no. 06, pp. 2–3, 2015.
- [39] B. Karrer and M. E. Newman, “Stochastic blockmodels and community structure in networks,” *Physical review E*, vol. 83, no. 1, p. 016107, 2011.

- [40] K. S. Xu and A. O. Hero, "Dynamic stochastic blockmodels for time-evolving social networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 4, pp. 552–562, 2014.
- [41] X. Zheng, Y. Zhong, D. Zeng, and F.-Y. Wang, "Social influence and spread dynamics in social networks," *Frontiers of Computer Science*, vol. 6, no. 5, pp. 611–620, 2012.
- [42] Z. Li, X. Fang, and O. R. L. Sheng, "A survey of link recommendation for social networks: Methods, theoretical foundations, and future research directions," *ACM Transactions on Management Information Systems (TMIS)*, vol. 9, no. 1, pp. 1–26, 2017.
- [43] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, pp. 415–444, 2001.
- [44] B. Latané, "The psychology of social impact," *American psychologist*, vol. 36, no. 4, p. 343, 1981.
- [45] G. S. Becker, "A theory of social interactions," *Journal of political economy*, vol. 82, no. 6, pp. 1063–1093, 1974.
- [46] F. Heider, *The psychology of interpersonal relations*. Psychology Press, 2013.
- [47] E. W. Ngai, S. S. Tao, and K. K. Moon, "Social media research: Theories, constructs, and conceptual frameworks," *International journal of information management*, vol. 35, no. 1, pp. 33–44, 2015.
- [48] L. A. McFarland and R. E. Ployhart, "Social media: A contextual framework to guide research and practice," *Journal of applied psychology*, vol. 100, no. 6, p. 1653, 2015.
- [49] J. M. McPherson and J. R. Ranger-Moore, "Evolution on a dancing landscape: organizations and networks in dynamic blau space," *Social Forces*, vol. 70, no. 1, pp. 19–42, 1991.

- [50] C. Sedikides and J. M. Jackson, "Social impact theory: A field test of source strength, source immediacy and number of targets," *Basic and applied social psychology*, vol. 11, no. 3, pp. 273–281, 1990.
- [51] H. C. Kelman, "Compliance, identification, and internalization three processes of attitude change," *Journal of conflict resolution*, vol. 2, no. 1, pp. 51–60, 1958.
- [52] H. H. Chang and S.-S. Chuang, "Social capital and individual motivations on knowledge sharing: Participant involvement as a moderator," *Information & management*, vol. 48, no. 1, pp. 9–18, 2011.
- [53] A. Portes, "Social capital: Its origins and applications in modern sociology," *Knowledge and social capital: Foundations and applications*, pp. 43–67, 2009.
- [54] G. A. Akerlof, "Social distance and social decisions," *Econometrica: Journal of the Econometric Society*, pp. 1005–1027, 1997.
- [55] G. R. Salancik and J. Pfeffer, "A social information processing approach to job attitudes and task design," *Administrative science quarterly*, pp. 224–253, 1978.
- [56] M. J. Lee, B. Ferwerda, J. Choi, J. Hahn, J. Y. Moon, and J. Kim, "Github developers use rockstars to overcome overflow of news," in *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, 2013, pp. 133–138.
- [57] X. Zheng, D. Zeng, and F.-Y. Wang, "Social balance in signed networks," *Information Systems Frontiers*, vol. 17, no. 5, pp. 1077–1095, 2015.
- [58] C. I. Hovland, "The order of presentation in persuasion," 1957.
- [59] J. Deese and R. A. Kaufman, "Serial effects in recall of unorganized and sequentially organized verbal material," *Journal of experimental psychology*, vol. 54, no. 3, p. 180, 1957.

- [60] N. Miller and D. T. Campbell, “Recency and primacy in persuasion as a function of the timing of speeches and measurements.” *The Journal of Abnormal and Social Psychology*, vol. 59, no. 1, p. 1, 1959.
- [61] J. Scott, “Social network analysis: developments, advances, and prospects,” *Social network analysis and mining*, vol. 1, no. 1, pp. 21–26, 2011.
- [62] S. Wasserman, K. Faust *et al.*, “Social network analysis: Methods and applications,” 1994.
- [63] M. O. Jackson, *Social and economic networks*. Princeton university press, 2010.
- [64] M. E. Newman, “Modularity and community structure in networks,” *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.
- [65] E. Otte and R. Rousseau, “Social network analysis: a powerful strategy, also for the information sciences,” *Journal of information Science*, vol. 28, no. 6, pp. 441–453, 2002.
- [66] D. J. Watts, “A twenty-first century science,” *Nature*, vol. 445, no. 7127, pp. 489–489, 2007.
- [67] K. Rohe, S. Chatterjee, and B. Yu, “Spectral clustering and the high-dimensional stochastic blockmodel,” *The Annals of Statistics*, vol. 39, no. 4, pp. 1878–1915, 2011.
- [68] S. Jiang and H. Chen, “Natergm: A model for examining the role of nodal attributes in dynamic social media networks,” *IEEE transactions on knowledge and data engineering*, vol. 28, no. 3, pp. 729–740, 2015.
- [69] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, “Kronecker graphs: an approach to modeling networks.” *Journal of Machine Learning Research*, vol. 11, no. 2, 2010.

- [70] A. Goldenberg, A. X. Zheng, S. E. Fienberg, E. M. Airoldi *et al.*, “A survey of statistical network models,” *Foundations and Trends® in Machine Learning*, vol. 2, no. 2, pp. 129–233, 2010.
- [71] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs over time: densification laws, shrinking diameters and possible explanations,” in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 177–187.
- [72] L. Wu, Y. Ge, Q. Liu, E. Chen, R. Hong, J. Du, and M. Wang, “Modeling the evolution of users’ preferences and social links in social networking services,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 6, pp. 1240–1253, 2017.
- [73] D. M. Dunlavy, T. G. Kolda, and E. Acar, “Temporal link prediction using matrix and tensor factorizations,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, pp. 1–27, 2011.
- [74] S. Gao, L. Denoyer, and P. Gallinari, “Temporal link prediction by integrating content and structure information,” in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 1169–1174.
- [75] X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang, “A deep learning approach to link prediction in dynamic networks,” in *Proceedings of the 2014 SIAM International conference on data mining*. SIAM, 2014, pp. 289–297.
- [76] W. Pan, W. Dong, M. Cebrian, T. Kim, J. H. Fowler, and A. S. Pentland, “Modeling dynamical influence in human interaction: Using data to make better inferences about influence within social systems,” *IEEE Signal Processing Magazine*, vol. 29, no. 2, pp. 77–86, 2012.

- [77] V. Raghavan, G. Ver Steeg, A. Galstyan, and A. G. Tartakovsky, “Modeling temporal activity patterns in dynamic social networks,” *IEEE Transactions on Computational Social Systems*, vol. 1, no. 1, pp. 89–107, 2014.
- [78] N. Du, X. Jia, J. Gao, V. Gopalakrishnan, and A. Zhang, “Tracking temporal community strength in dynamic networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 11, pp. 3125–3137, 2015.
- [79] C. Gao and J. Liu, “Network-based modeling for characterizing human collective behaviors during extreme events,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 171–183, 2016.
- [80] J. C. L. Pinto, T. Chahed, and E. Altman, “Trend detection in social networks using hawkes processes,” in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, 2015, pp. 1441–1448.
- [81] Y. Zhu, E. Zhong, S. J. Pan, X. Wang, M. Zhou, and Q. Yang, “Predicting user activity level in social networks,” in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 159–168.
- [82] D. Qiu, H. Li, and Y. Li, “Identification of active valuable nodes in temporal online social network with attributes,” *International Journal of Information Technology & Decision Making*, vol. 13, no. 04, pp. 839–864, 2014.
- [83] D. Antoniadou and C. Dovrolis, “Co-evolutionary dynamics in social networks: A case study of twitter,” *Computational Social Networks*, vol. 2, no. 1, pp. 1–21, 2015.
- [84] A. Alexandrov, G. Krastev, and V. Markl, “Representations and optimizations for embedded parallel dataflow languages,” *ACM Transactions on Database Systems (TODS)*, vol. 44, no. 1, p. 4, 2019.

- [85] B. Rao and L. Wang, “A survey of semantics-aware performance optimization for data-intensive computing,” in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/Pi-Com/DataCom/CyberSciTech)*. IEEE, 2017, pp. 81–88.
- [86] A. Y. Zomaya and S. Sakr, *Handbook of Big Data Technologies*. Springer, 2017.
- [87] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.
- [88] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl *et al.*, “The stratosphere platform for big data analytics,” *The VLDB Journal*, 2014.
- [89] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, “Spark sql: Relational data processing in spark,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.
- [90] Z. Guo, X. Fan, R. Chen, J. Zhang, H. Zhou, S. McDirmid, C. Liu, W. Lin, J. Zhou, and L. Zhou, “Spotting code optimizations in data-parallel pipelines through periscope,” in *OSDI*, 2012.
- [91] E. Jahani, M. J. Cafarella, and C. Ré, “Automatic optimization for mapreduce programs,” *Proceedings of the VLDB Endowment*, 2011.

- [92] J. Liu, N. Ravi, S. Chakradhar, and M. Kandemir, “Panacea: towards holistic optimization of mapreduce applications,” in *Proceedings of the Tenth International Symposium on Code Generation and Optimization*. ACM, 2012.
- [93] Z. Liu, H. Zhang, B. Rao, and L. Wang, “A reinforcement learning based resource management approach for time-critical workloads in distributed computing environment,” in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 252–261.
- [94] A. Rheinländer, A. Heise, F. Hueske, U. Leser, and F. Naumann, “Sofa: An extensible logical optimizer for udf-heavy data flows,” *Information Systems*, vol. 52, pp. 96–125, 2015.
- [95] H. Zhang, L. Wang, and H. Huang, “Smarth: Enabling multi-pipeline data transfer in hdfs,” in *2014 43rd International Conference on Parallel Processing*. IEEE, 2014, pp. 30–39.
- [96] L. Wang, S. Lu, X. Fei, A. Chebotko, H. V. Bryant, and J. L. Ram, “Atomicity and provenance support for pipelined scientific workflows,” *Future Generation Computer Systems*, vol. 25, no. 5, pp. 568–576, 2009.
- [97] H. Zhang, H. Huang, and L. Wang, “Mrapid: An efficient short job optimizer on hadoop,” in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 459–468.
- [98] J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou, “Optimizing data shuffling in data-parallel computation by understanding user-defined functions.” in *NSDI*, 2012.
- [99] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, “Scope: easy and efficient parallel processing of massive data sets,” *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1265–1276, 2008.

- [100] D. Garbervetsky, Z. Pavlinovic, M. Barnett, M. Musuvathi, T. Mytkowicz, and E. Zoppi, “Static analysis for optimizing big data queries,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 932–937.
- [101] M. Interlandi, S. D. Tetali, M. A. Gulzar, J. Noor, T. Condie, M. Kim, and T. Millstein, “Optimizing interactive development of data-intensive applications,” in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 510–522.
- [102] A. Roy, A. Jindal, H. Patel, A. Gosalia, S. Krishnan, and C. Curino, “Sparkcruise: Handsfree computation reuse in spark,” *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1850–1853, 2019.
- [103] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, “Apache tez: A unifying framework for modeling and building data processing applications,” in *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*, 2015, pp. 1357–1369.
- [104] L. Xu, M. Li, L. Zhang, A. R. Butt, Y. Wang, and Z. Z. Hu, “Memtune: Dynamic memory management for in-memory data analytic platforms,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2016, pp. 383–392.
- [105] Y. Yu, W. Wang, J. Zhang, and K. B. Letaief, “Lrc: Dependency-aware cache management for data analytics clusters,” in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
- [106] T. B. Perez, X. Zhou, and D. Cheng, “Reference-distance eviction and prefetching for cache management in spark,” in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

- [107] M. Motwani, S. Sankaranarayanan, R. Just, and Y. Brun, “Do automated program repair techniques repair hard and important bugs?” *Empirical Software Engineering*, vol. 23, no. 5, pp. 2901–2947, 2018.
- [108] M. Monperrus, “Automatic software repair: a bibliography,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–24, 2018.
- [109] D. Hovemeyer and W. Pugh, “Finding more null pointer bugs, but not too many,” in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2007, pp. 9–14.
- [110] L. Gazzola, D. Micucci, and L. Mariani, “Automatic software repair: A survey,” *IEEE Transactions on Software Engineering*, vol. 45, no. 1, pp. 34–67, 2017.
- [111] Z. Li and Y. Zhou, “Pr-miner: automatically extracting implicit programming rules and detecting violations in large software code,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, pp. 306–315, 2005.
- [112] J. Toman and D. Grossman, “Taming the static analysis beast,” in *2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA*, ser. LIPIcs, vol. 71. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 18:1–18:14.
- [113] P. Bian, B. Liang, W. Shi, J. Huang, and Y. Cai, “Nar-miner: discovering negative association rules from code for bug detection,” in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 411–422.
- [114] H. Hata, E. Shihab, and G. Neubig, “Learning to generate corrective patches using neural machine translation,” *arXiv preprint arXiv:1812.07170*, 2018.

- [115] S. Chakraborty, M. Allamanis, and B. Ray, “Codit: Code editing with tree-based neural machine translation,” *arXiv preprint arXiv:1810.00314*, 2018.
- [116] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk, “On learning meaningful code changes via neural machine translation,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 25–36.
- [117] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
- [118] S. Bhatia, P. Kohli, and R. Singh, “Neuro-symbolic program corrector for introductory programming assignments,” in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, IEEE. ACM, 2018, pp. 60–70.
- [119] M. White, M. Tufano, M. Martinez, M. Monperrus, and D. Poshyvanyk, “Sorting and transforming program repair ingredients via deep learning code similarities,” in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 479–490.
- [120] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.
- [121] Y. Pu, K. Narasimhan, A. Solar-Lezama, and R. Barzilay, “sk_p: a neural program corrector for moocs,” in *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*. ACM, 2016, pp. 39–40.
- [122] Y. Li, S. Wang, and T. N. Nguyen, “Dlfix: Context-based code transformation learning for automated program repair,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 602–614.

- [123] E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, and K. Wang, “Hoppity: Learning graph transformations to detect and fix bugs in programs,” in *International Conference on Learning Representations (ICLR)*, 2020.
- [124] D. Tarlow, S. Moitra, A. Rice, Z. Chen, P.-A. Manzagol, C. Sutton, and E. Aftandilian, “Learning to fix build errors with graph2diff neural networks,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 19–20.
- [125] —, “Learning to fix build errors with graph2diff neural networks,” *arXiv preprint arXiv:1911.01205*, 2019.
- [126] P. G. Savage, “Strapdown inertial navigation integration algorithm design part 2: Velocity and position algorithms,” *Journal of Guidance, Control, and dynamics*, vol. 21, no. 2, pp. 208–221, 1998.
- [127] S. Shen, M. Gowda, and R. Roy Choudhury, “Closing the gaps in inertial motion tracking,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 429–444.
- [128] Q. Tian, Z. Salcic, I. Kevin, K. Wang, and Y. Pan, “An enhanced pedestrian dead reckoning approach for pedestrian tracking using smartphones,” in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 2015, pp. 1–6.
- [129] D. Titterton, J. L. Weston, and J. Weston, *Strapdown inertial navigation technology*. IET, 2004, vol. 17.
- [130] E. Foxlin, “Pedestrian tracking with shoe-mounted inertial sensors,” *IEEE Computer graphics and applications*, vol. 25, no. 6, pp. 38–46, 2005.

- [131] A. Brajdic and R. Harle, “Walk detection and step counting on unconstrained smartphones,” in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, 2013, pp. 225–234.
- [132] C. Chen, X. Lu, A. Markham, and N. Trigoni, “Ionet: Learning to cure the curse of drift in inertial odometry,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [133] S. Herath, H. Yan, and Y. Furukawa, “Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3146–3152.
- [134] S. Sun, D. Melamed, and K. Kitani, “Idol: Inertial deep orientation-estimation and localization,” *arXiv preprint arXiv:2102.04024*, 2021.
- [135] H. Yan, Q. Shan, and Y. Furukawa, “Ridi: Robust imu double integration,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 621–636.
- [136] B. Wagstaff, V. Peretroukhin, and J. Kelly, “Robust data-driven zero-velocity detection for foot-mounted inertial navigation,” *IEEE Sensors Journal*, vol. 20, no. 2, pp. 957–967, 2019. [Online]. Available: <https://arxiv.org/abs/1910.00529>
- [137] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, “Robust visual inertial odometry using a direct ekf-based approach,” in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.
- [138] D. Ahmetovic, C. Gleason, C. Ruan, K. Kitani, H. Takagi, and C. Asakawa, “Navcog: a navigational cognitive assistant for the blind,” in *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 2016, pp. 90–99.

- [139] J. Li, M. Guo, and S. Li, “An indoor localization system by fusing smartphone inertial sensors and bluetooth low energy beacons,” in *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*. IEEE, 2017, pp. 317–321.
- [140] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time.” in *Robotics: Science and Systems*, vol. 2, 2014.
- [141] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-based visual–inertial odometry using nonlinear optimization,” *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.
- [142] V. Usenko, J. Engel, J. Stückler, and D. Cremers, “Direct visual-inertial odometry with stereo cameras,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1885–1892.
- [143] D. Tayouri, “Social media as an intelligence goldmine,” *Cyber security review*, pp. 27–30, 2016.
- [144] M. Chui, J. Manyika, and J. Bughin, “The social economy: Unlocking value and productivity through social technologies,” McKinsey Global Institute, Tech. Rep., 2012.
- [145] E. N. Gilbert, “Random graphs,” *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.
- [146] B. Bollobás, “Random graphs,” in *Modern graph theory*. Springer, 1998, pp. 215–252.
- [147] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *science*, vol. 286, no. 5439, pp. 509–512, 1999.
- [148] L. C. Freeman, “A set of measures of centrality based on betweenness,” *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

- [149] M. O. Jackson, *Social and Economic Networks*. Princeton, NJ: Princeton University Press, 2008.
- [150] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 23, pp. 8577–8582, 2006. [Online]. Available: <GotoISI>://WOS:000238278400002
- [151] S. P. Borgatti, "Centrality and network flow," *Social networks*, vol. 27, no. 1, pp. 55–71, 2005.
- [152] J. Bush, T. F. McLarty III, and E. H. Alden, *U.S. immigration policy*, ser. Independent Task Force Report No. 63. New York: Council on Foreign Relations, 2009.
- [153] J. Gans, E. M. Replogle, and D. J. Tichenor, *Debates on U.S. Immigration*. Sage Publications, Inc., 2012.
- [154] M. C. LeMay, *U.S. immigration: A reference handbook*, ser. ABC-CLIO's contemporary world issues series. Santa Barbara, Calif.: ABC-CLIO, 2004. [Online]. Available: <http://www.loc.gov/catdir/toc/ecip049/2003021522.html><http://www.loc.gov/catdir/description/abcclio041/2003021522.html>
- [155] U. C. on Immigration Reform, *U.S. immigration policy - Restoring credibility: A report to Congress*. Washington, DC (1825 Connecticut Ave., NW, Suite 511, Washington 20009): U.S. Commission on Immigration Reform, 1994.
- [156] P. Constable and C. Morello, "Marchers urge congress to pass immigration reform; several congressmen arrested," *The Washington Post*, 2013.
- [157] C. Tofallis, "A better measure of relative prediction accuracy for model selection and model estimation," *Journal of the Operational Research Society*, vol. 66, no. 8, pp. 1352–1362, 2015.

- [158] S. Palkar, F. Abuzaid, P. Bailis, and M. Zaharia, “Filter before you parse: Faster analytics on raw data with sparser,” *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1576–1589, 2018.
- [159] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas, “Opening the black boxes in data flow optimization,” *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1256–1267, 2012.
- [160] A. Rheinländer, U. Leser, and G. Graefe, “Optimization of complex dataflows with user-defined functions,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–39, 2017.
- [161] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, “Log-based abnormal task detection and root cause analysis for spark,” in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 389–396.
- [162] S. Lu, X. Wei, Y. Li, and L. Wang, “Detecting anomaly in big data system logs using convolutional neural network,” in *IEEE Cyber Science and Technology Congress (CyberSciTech)*. IEEE, 2018, pp. 151–158.
- [163] S. Lu, X. Wei, B. Rao, B. Tak, L. Wang, and L. Wang, “Ladra: Log-based abnormal task detection and root-cause analysis in big data processing with spark,” *Future Generation Computer Systems*, vol. 95, pp. 392–403, 2019.
- [164] S. Chiba, “Javassist—a reflection-based programming wizard for java,” in *Proceedings of OOPSLA’98 Workshop on Reflective Programming in C++ and Java*, vol. 174, 1998, p. 21.
- [165] A. Shinnar, D. Cunningham, V. Saraswat, and B. Herta, “M3r: increased performance for in-memory hadoop jobs,” *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1736–1747, 2012.

- [166] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, “Storm@ twitter,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.* ACM, 2014.
- [167] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.
- [168] S. Ioannidis and E. Yeh, “Adaptive caching networks with optimality guarantees,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1, pp. 113–124, 2016.
- [169] Z. Yang, D. Jia, S. Ioannidis, N. Mi, and B. Sheng, “Intermediate data caching optimization for multi-stage and parallel big data frameworks,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD).* IEEE, 2018, pp. 277–284.
- [170] G. Optimization, “Inc., “gurobi optimizer reference manual,” 2015,” 2014.
- [171] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of program analysis.* Springer, 2015.
- [172] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna, “Stage aware performance modeling of dag based in memory analytic platforms,” in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD).* IEEE, 2016, pp. 188–195.
- [173] J. McAuley and A. Yang, “Addressing complex and subjective product-related queries with customer reviews,” in *Proceedings of the 25th International Conference on World Wide Web.* International World Wide Web Conferences Steering Committee, 2016, pp. 625–635.
- [174] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *arXiv preprint arXiv:1609.08144*, 2016.
- [175] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *arXiv preprint arXiv:1508.07909*, 2015.

- [176] V. J. Hellendoorn and P. Devanbu, “Are deep neural networks the best choice for modeling source code?” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 763–773.
- [177] R.-M. Karampatsis and C. Sutton, “Maybe deep neural networks are the best choice for modeling source code,” *arXiv preprint arXiv:1903.05734*, 2019.
- [178] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, “Big code!= big vocabulary: Open-vocabulary models for source code,” *arXiv preprint arXiv:2003.07914*, 2020.
- [179] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *arXiv preprint arXiv:1310.4546*, 2013.
- [180] Y. Goldberg and O. Levy, “word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method,” *arXiv preprint arXiv:1402.3722*, 2014.
- [181] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, “Codebert: A pre-trained model for programming and natural languages,” *arXiv preprint arXiv:2002.08155*, 2020.
- [182] W. Chen, E. Matusov, S. Khadivi, and J.-T. Peter, “Guided alignment training for topic-aware neural machine translation,” *arXiv preprint arXiv:1607.01628*, 2016.
- [183] P. Koehn and R. Knowles, “Six challenges for neural machine translation,” *arXiv preprint arXiv:1706.03872*, 2017.
- [184] J.-T. Peter, A. Nix, and H. Ney, “Generating alignments using target foresight in attention-based neural machine translation,” *The Prague Bulletin of Mathematical Linguistics*, vol. 108, no. 1, pp. 27–36, 2017.
- [185] E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum, “Linguistically-informed self-attention for semantic role labeling,” *arXiv preprint arXiv:1804.08199*, 2018.

- [186] S. Garg, S. Peitz, U. Nallasamy, and M. Paulik, “Jointly learning to align and translate with transformer models,” *arXiv preprint arXiv:1909.02074*, 2019.
- [187] K. Song, K. Wang, H. Yu, Y. Zhang, Z. Huang, W. Luo, X. Duan, and M. Zhang, “Alignment-enhanced transformer for constraining nmt with pre-specified translations,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 8886–8893.
- [188] S. Wiseman and A. M. Rush, “Sequence-to-sequence learning as beam-search optimization,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. The Association for Computational Linguistics, 2016, pp. 1296–1306.
- [189] U. Z. Ahmed, P. Kumar, A. Karkare, P. Kar, and S. Gulwani, “Compilation error repair: for the student programs, from the student programs,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. ACM, 2018, pp. 78–87.
- [190] M. Vasic, A. Kanade, P. Maniatis, D. Bieber, and R. Singh, “Neural program repair by jointly learning to localize and repair,” *arXiv preprint arXiv:1904.01720*, 2019.
- [191] C. Maddison and D. Tarlow, “Structured generative models of natural source code,” in *International Conference on Machine Learning*. JMLR.org, 2014, pp. 649–657.
- [192] Z. Tu, Z. Su, and P. Devanbu, “On the localness of software,” in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. IEEE Computer Society, 2014, pp. 269–280.
- [193] P. Gage, “A new algorithm for data compression,” *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

- [194] T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing,” *arXiv preprint arXiv:1808.06226*, 2018.
- [195] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *arXiv preprint arXiv:2005.14165*, 2020.
- [196] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, “Graphcodebert: Pre-training code representations with data flow,” *arXiv preprint arXiv:2009.08366*, 2020.
- [197] Z. S. Harris, “Distributional structure,” *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.
- [198] N. Smith, D. van Bruggen, and F. Tomassetti, “Javaparser: visited,” *Leanpub, oct. de*, 2017.
- [199] J. Saraiva, C. Bird, and T. Zimmermann, “Products, developers, and milestones: how should i build my n-gram language model,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 998–1001.
- [200] M. Odersky, L. Spoon, and B. Venners, *Programming in scala*. Artima Inc, 2008.
- [201] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “A general path-based representation for predicting program properties,” *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 404–419, 2018.
- [202] U. Alon, S. Brody, O. Levy, and E. Yahav, “code2seq: Generating sequences from structured representations of code,” *arXiv preprint arXiv:1808.01400*, 2018.
- [203] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: Learning distributed representations of code,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

- [204] Y. David, U. Alon, and E. Yahav, “Neural reverse engineering of stripped binaries using augmented control flow graphs,” *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–28, 2020.
- [205] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, “Fine-grained and accurate source code differencing,” in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 2014, pp. 313–324.
- [206] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.*, “A survey on visual transformer,” *arXiv preprint arXiv:2012.12556*, 2020.
- [207] X. Li, G. Li, L. Liu, M. Meng, and S. Shi, “On the word alignment from neural machine translation,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 1293–1303.
- [208] R. Müller, S. Kornblith, and G. E. Hinton, “When does label smoothing help?” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 2019, pp. 4696–4705.
- [209] T. Van Erven and P. Harremoës, “Rényi divergence and kullback-leibler divergence,” *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.
- [210] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. ACL, 2002, pp. 311–318.
- [211] N. Tran, H. Tran, S. Nguyen, H. Nguyen, and T. Nguyen, “Does bleu score work for code migration?” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 165–176.

- [212] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [213] G. Klein, Y. Kim, Y. Deng, V. Nguyen, J. Senellart, and A. M. Rush, “Opennmt: Neural machine translation toolkit,” in *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas, AMTA 2018, Boston, MA, USA, March 17-21, 2018 - Volume 1: Research Papers*. Association for Machine Translation in the Americas, 2018, pp. 177–184.
- [214] R. L. Russell and C. Reale, “Multivariate uncertainty in deep learning,” *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [215] X. Wang, R. Girshick, A. Gupta, and K. He, “Non-local neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.
- [216] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, “A simple neural network module for relational reasoning,” *arXiv preprint arXiv:1706.01427*, 2017.
- [217] H. Hu, Z. Zhang, Z. Xie, and S. Lin, “Local relation networks for image recognition,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3464–3473.
- [218] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, “Stand-alone self-attention in vision models,” *arXiv preprint arXiv:1906.05909*, 2019.
- [219] H. Zhao, J. Jia, and V. Koltun, “Exploring self-attention for image recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10076–10085.

- [220] Y. Li, T. Yao, Y. Pan, and T. Mei, “Contextual transformer networks for visual recognition,” *arXiv preprint arXiv:2107.12292*, 2021.
- [221] J. Yao, D. Wang, H. Hu, W. Xing, and L. Wang, “Adcnn: Towards learning adaptive dilation for convolutional neural networks,” *Pattern Recognition*, vol. 123, p. 108369, 2022.
- [222] A. Kendall and Y. Gal, “What uncertainties do we need in bayesian deep learning for computer vision?” *Advances in neural information processing systems*, vol. 30, 2017.
- [223] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.
- [224] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *ICLR (Poster)*, 2015.
- [225] L. Prechelt, “Early stopping-but when?” in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.
- [226] D. Wang, Y. Li, L. Wang, and B. Gong, “Neural networks are more productive teachers than human raters: Active mixup for data-efficient knowledge distillation from a black-box model,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1498–1507.
- [227] C. Chen, P. Zhao, C. X. Lu, W. Wang, A. Markham, and N. Trigoni, “Oxiod: The dataset for deep inertial odometry,” *arXiv preprint arXiv:1809.07491*, 2018.
- [228] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, “Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4304–4311.

- [229] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, D. Cremers, R. Siegwart, and W. Burgard, “Towards a benchmark for rgb-d slam evaluation,” in *Rgb-d workshop on advanced reasoning with depth cameras at robotics: Science and systems conf.(rss)*, 2011.