ABSTRACT

SYSTEMS FOR PRIVACY-PRESERVING MACHINE LEARNING

by
Pritam Sen

Machine learning algorithms based on Deep Neural Networks have achieved remarkable results and are currently employed across many application domains. However, these algorithms rely on obtaining raw data, which is often confidential and can create potential security and privacy risks. Several privacy-preserving techniques have been developed to deal with these issues, including differential privacy, homomorphic encryption, secure multi-party computation, and federated learning. These techniques allow for data analysis and machine learning applications by mitigating or eliminating the privacy risks of individuals whose data is being used.

However, privacy is a complex issue, and a single solution typically cannot fully address it across different application domains. Furthermore, the trade-offs between privacy and utility of machine learning models may vary depending on the specific application and context. Therefore, it is essential to carefully consider the privacy implications of machine learning algorithms and choose the most appropriate solution for each specific case. This dissertation presents three privacy-preserving machine learning systems for several application cases.

First, a secure federated learning aggregation technique, called FedMTL, is proposed to handle task heterogeneity across users. FedMTL generates personalized multi-task learning models based on task similarities, which are determined by analyzing the parameters of the task-specific layers in the trained models. To prevent privacy leakage through these model parameters and to protect the privacy of the task types, FedMTL employs low-overhead algorithms that are adaptable to existing techniques for secure aggregation. The FedMTL aggregation algorithm is implemented using secure multi-party computation (SMPC), and it can achieve

the same accuracy as the plain-text version while preserving privacy. Extensive experiments on three datasets demonstrate that FedMTL outperforms state-of-the-art approaches.

Second, a secure and effective inference solution, called CryptGNN, is designed for third-party graph neural network models in the cloud, which are accessed by clients as ML as a service. The main novelty of CryptGNN is its secure message passing and feature transformation layers using SMPC techniques. CryptGNN protects the client's input data and graph structure from the cloud provider and the third-party model owner, and it protects the model parameters from the cloud provider and the clients. CryptGNN works with any number of SMPC parties, does not require a trusted server, and is provably secure even if $\mathcal{P} - 1$ out of $\mathcal{P}$ parties in the cloud collude. Theoretical analysis and empirical experiments demonstrate the security and efficiency of CryptGNN.

Third, this dissertation discusses the privacy threats posed by current indoor localization and prediction approaches, and then proposes GoPlaces, a novel app that fuses inertial sensor data with WiFi-RTT estimated distances to predict the indoor places visited by a user in a privacy-preserving way. GoPlaces works on user phones and protects users' location privacy because user's data never leaves the phone. GoPlaces does not require any infrastructure, except for one cheap off-the-shelf WiFi access point that supports ranging with RTT. GoPlaces uses an attention-based BiLSTM model to detect the user's current trajectory, which is then used together with historical information stored in a prediction tree to infer the user's future places. GoPlaces is implemented in Android and is evaluated in several indoor spaces. The experimental results demonstrate high prediction accuracy in various situations. Furthermore, GoPlaces has been shown to be feasible in real-world scenarios due to its low latency and low resource consumption on the phones.

SYSTEMS FOR PRIVACY-PRESERVING MACHINE LEARNING

by
Pritam Sen

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2025

APPROVAL PAGE

SYSTEMS FOR PRIVACY-PRESERVING MACHINE LEARNING

Pritam Sen

Dr. Cristian Borcea, Dissertation Advisor — Date
Professor, Department of Computer Science, NJIT

Dr. Reza Curtmola, Committee Member — Date
Professor, Department of Computer Science, NJIT

Dr. Shantanu Sharma, Committee Member — Date
Assistant Professor, Department of Computer Science, NJIT

Dr. Tao Han, Committee Member — Date
Associate Professor, Department of Electrical and Computer Engineering, NJIT

Dr. Yao Ma, Committee Member — Date
Assistant Professor, Department of Computer Science,
Rensselaer Polytechnic Institute, Troy, NY

Dr. Qiong Wu, Committee Member — Date
Principal Inventive Scientist, AT&T Chief Data Office, Bedminster, NJ

BIOGRAPHICAL SKETCH

Author: Pritam Sen

Degree: Doctor of Philosophy

Date: May 2025

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, NJ, US, 2025
- Bachelor of Science in Electrical and Electronic Engineering
  Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, 2013

Major: Computer Science

Presentations and Publications:

P. Sen, Y. Ma, C. Borcea, "CryptGNN: Enabling secure inference for graph neural networks", under review by the 45th IEEE International Conference on Distributed Computing Systems (ICDCS), 2025.

A. Alemari, **P. Sen**, C.Borcea, "AdFL: In-browser federated learning for online advertisement", under review by the International AAAI Conference on Web and Social Media (ICWSM), 2025.

X. Jiang, **P. Sen**, C. Borcea, "Federated continual learning using concept matching", under review by the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2025.

P. Sen, C. Borcea, "FedMTL: Privacy-preserving federated multi-task learning", in 27th European Conference on Artificial Intelligence (ECAI), 2024.

M. Smith, A. Torres-Agüero, R. Grossman, **P. Sen**, Y. Chen, C. Borcea, "A study of GDPR compliance under the transparency and consent framework", in Proceedings of the ACM Web Conference (WWW), 2024.

P. Sen, X. Jiang, Q. Wu, M. Talasila, WL. Hsu, C. Borcea, "GoPlaces: An app for personalized indoor place prediction", in 20th IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS), 2023.

P. Sen, X. Jiang, Q. Wu, M. Talasila, WL. Hsu, C. Borcea, "Demo abstract: Indoor place prediction on smart phones", in 20th ACM Conference on Embedded Networked Sensor Systems (SenSys), 2022.

To Those Who Shaped My Journey with Love and Wisdom.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my dissertation advisor, Dr. Cristian Borcea, whose guidance and support have been instrumental in shaping my research. His feedback and insightful suggestions have been invaluable throughout this journey, and I appreciate his mentorship greatly.

I also extend my sincere appreciation to my Committee members, Dr. Reza Curtmola, Dr. Shantanu Sharma, Dr. Tao Han, Dr. Qiong Wu, and Dr. Yao Ma for their time, effort, and thoughtful feedback on my research. Their expertise and constructive criticism have helped me refine and improve my work.

In addition, I would like to thank Dr. Manoop Talasila, Dr. Guy Jacobson, Dr. Wen-Ling Hsu, and Syed Anwar Aftab from AT&T Research Labs, for being my industry collaborators. Their comments and suggestions have greatly helped me improve this dissertation.

I would also like to thank my friends for their support, motivation, and the moments of laughter that kept me going. Their presence has made this journey more meaningful, and I am truly grateful for each of them.

Lastly, I am profoundly grateful to my parents for their unwavering encouragement and belief in me, which have been my greatest sources of strength. I am also deeply thankful to my wife for her patience, love, and constant support - I could not have done this without her. Above all, I am forever grateful for the love and support of my family.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Machine learning (ML) algorithms play a central role in numerous applications that significantly impact our daily lives, including recommendation systems [1], medical diagnosis [2], traffic forecasting [3], autonomous vehicles [4], and more. However, building an accurate and reliable model requires a significant amount of data for training. This process can raise concerns regarding data privacy and security [5], especially when dealing with sensitive personal data such as health records or financial information. Moreover, ML models themselves constitute a threat [6], as it is possible to extract sensitive information from them. Therefore, it is required to protect data privacy in the various phases of the ML-based systems, including data preparation, model training and evaluation, model deployment, and model inference.

The development of systems for privacy-preserving machine learning is an emerging area of research that aims to design techniques for training machine learning models on sensitive data without compromising the privacy of individuals whose data is being used. Various techniques and approaches have been developed to tackle privacy concerns in machine learning. One such technique is differential privacy (DP) [7, 8], which involves adding random noise to training data to safeguard the privacy of individuals. Another approach is to use homomorphic encryption (HE) [9, 10] to perform computations on encrypted data. Secure multi-party communication (SMPC) [11, 12] enables multiple parties to jointly compute a function on their private inputs without revealing their inputs to one another. Despite the availability of these privacy-enhancing techniques, there are still several challenges that need to be addressed. For example, achieving a balance between privacy and utility [13] is a difficult task in DP. The computation and communication overhead [14]

required for complex algorithms can be significant, which may hinder the adoption of techniques such as HE or SMPC in environments with limited resources.

With the advancement of computing power on mobile devices, it has become possible to collect and preprocess data from device sensors, train and evaluate ML models on the device [15, 16], and make real-time predictions while ensuring that user data and models remain on the device to preserve privacy. However, privacy concerns persist due to the risk of data exploitation by external devices used for data collection in the environment [17, 18]. In addition, certain systems require training models to learn heterogeneous trends from multiple users, making it necessary to protect the privacy of their data. Federated learning [19, 20] has been proposed as a solution that enables model training without sharing raw data across different devices. Nevertheless, there is a security risk associated with federated learning, as an adversary may be able to infer information [6] about the training dataset through the model parameters.

The choice of a privacy-preserving ML technique relies on several factors, such as the use case, the assets to be protected, etc. There is no silver-bullet solution; understanding the various factors helps identify the right approach. In addition, we must balance scenario-specific considerations with the need for portable and reusable solutions. In this dissertation, we propose privacy-preserving systems for specific applications, such as designing secure algorithms to support multi-task learning in a federated learning setting, using the SMPC technique for secure inference service of graph neural networks in the cloud, and using mobile computation to predict users' location in an indoor environment.

The rest of this chapter presents an overview of three systems for privacy-preserving machine learning, namely secure multi-task learning in a federated learning setting in Section 1.1, secure graph neural networks in the cloud in Section 1.2 and privacy-preserving indoor place prediction using mobile computation in Section 1.3.

The contributions of this dissertation are presented in Section 1.4. Finally, Section 1.5 details the structure of this dissertation.

1.1 Secure Multi-task Learning in a Federated Learning Setting

The growing volume of data generated by smart phones and IoT devices enables them to train models for various tasks. Multi-task learning (MTL) [21] proves particularly useful for mobile and IoT devices that require local model training to uphold privacy (i.e., avoid sending data to a server for centralized training), as the overhead of MTL is comparatively lower on resource-constrained devices than training individual models for each task. For example, a device can collect audio signals and employ MTL models for tasks such as speech recognition, speaker identification, and emotion detection. Autonomous vehicles, likewise, can capture video data to train MTL models for lane detection, recognizing traffic signs, predicting pedestrian intent, and detecting obstacles. Text data gathered by mobile devices can also be used to train MTL models for sentiment analysis, text summarization, spam detection, and named entity recognition.

Although these models can be trained independently by each device, collaboration among devices can further improve model performance by allowing them to learn from each other's models, which may be trained on similar or different sets of tasks. However, privacy becomes a concern when raw data is gathered from devices for centralized learning or when clients exchange data for distributed learning. Federated learning (FL) [22] allows collaborative training across clients while keeping client data locally. Unfortunately, conventional FL is not applicable to MTL scenarios, wherein clients do not collectively train models for similar sets of tasks. One possible approach is for each client to share information about the tasks (i.e., task IDs) with the FL aggregation server so that it can group the MTL models based on the set of tasks for which the models were trained. However, this raises concerns, as it could reveal

information about the client, such as the tasks performed, which can expose the client's interests, preferences, or client's processing capabilities, such as computational power, storage capacity, or specialized hardware. Furthermore, sharing MTL model parameters with an aggregator server may inadvertently leak task- or dataset-related information.

Prior studies [23, 24] on federated MTL assume each client performs only one task, and tune the local models through methods such as (a) minimizing the parameter differences between models or (b) clustering models into distinct groups to generate average models for each group. However, these approaches face challenges when clients work on different sets of tasks due to: (i) Heterogeneity in model structures arising from differences in the set of tasks, and (ii) Difficulty of clustering models, as a client model with a specific set of tasks may yield a similar score to models trained by others on different subsets of tasks, making it difficult to assign a unique cluster ID. Therefore, it is crucial to design an aggregation algorithm that considers the heterogeneity of tasks executed by each client. Furthermore, the algorithm needs to ensure the privacy of the model parameters to prevent leakage of the training datasets [25] and of the tasks executed by each client. While various privacy-preserving techniques exist for secure aggregation [26, 27], they cannot be applied directly because of the following challenges: (a) The heterogeneity of model structures may leak the number and types of tasks in the dataset, and (b) Analyzing encrypted model parameters and applying complex algorithms on encrypted data incurs high computation and communication overhead.

In this dissertation, we focus on designing an algorithm that allows clients to obtain personalized MTL models in an FL setting while safeguarding the confidentiality of user data. To address these issues, this dissertation introduces **FedMTL**, a novel algorithm to support MTL in the FL setting. This work paves the way for privacy-aware, resource-efficient federated MTL, enabling the deployment of

collaborative intelligence in privacy-sensitive environments such as mobile and IoT applications. FedMTL uses new protocols to enhance the capabilities of each client model by enabling collaboration among models from the participating clients based on similarities in the tasks they execute. It aims to develop a low resource-consumption approach, enabling straightforward integration with established privacy-preserving techniques for secure aggregation.

Targeting privacy assurance and high efficiency, FedMTL employs the secure multi-party computation (SMPC) technique [28, 29, 11], which enables a set of aggregation servers to compute a public function on secret inputs. To protect the data (i.e., model parameters, task IDs), FedMTL splits and distributes the data across multiple servers, each of which is only allowed to access its local data. Thus, neither server is able to combine or reconstruct the original data. To perform computations, each server conducts local calculations using its local copy of the data and coordinates with the other servers to produce the final personalized models. To evaluate the effectiveness of FedMTL, we implement the proposed FL aggregation algorithm and privacy-preserving protocols in SMPC, achieving the same accuracy as the plain text version while preserving client data privacy.

1.2 Secure Graph Neural Networks in the Cloud

Graphs provide a universal representation for a wide range of real-world data, including online social networks [30], transportation networks [31, 32], financial transaction networks [33, 34], integrated circuits [35], and chemical molecules [36, 37]. Graph Neural Networks (GNNs) are a set of prominent neural models that translates the success of deep learning (DL) to graph-structured data [38, 39, 40]. Thanks to their great capability in learning representations for graphs, GNNs have advanced numerous computational tasks on graph-structured data, including drug discovery

[41, 42], fraud detection [43, 44], recommendation systems [45, 46], and traffic flow prediction [47, 48].

As with traditional DL models, unleashing the power of GNNs typically requires large amounts of training data and prohibitive computing resources, which are not available to small businesses or individuals. Hence, a promising way to democratize access to large-scale GNN models is to provide GNN-based predictive tools in the form of machine learning-as-a-service (MLaaS) [49, 50, 51, 52, 53, 54]. In MLaaS settings, the owners of the trained models can monetize these models, while the clients do not need to train or optimize the models. The clients perform inference by uploading their input data to the cloud service. In this way, they benefit from models that they may not be able to build quickly, effectively, and inexpensively. For example, a company may enable researchers and small start-ups to quickly screen out unqualified molecules in a drug discovery process by providing a GNN model in the cloud, trained on a proprietary collection of organic compound data. Similarly, a company may use a large proprietary codebase to represent code components by a program dependence graph (PDG) [55], and train a GNN model that provides an automated code analysis service in the cloud for software developers.

However, privacy in GNN inference in the cloud is a critical challenge, particularly in MLaaS settings, where clients submit sensitive graph-structured data to a cloud-hosted GNN model. In particular, there are two main challenges. First, the sensitive input graph data that the clients submit to the GNN model in the cloud need to be protected from the cloud provider and the GNN model owner. Second, the GNN model in the cloud needs to be protected from the cloud provider and the clients. Therefore, to promote GNNs in MLaaS settings, this dissertation addresses these two privacy challenges for GNN inference in the cloud.[1]

[1]We do not address training privacy, as we assume the model may be trained in the private infrastructure of the model owner.

Several research efforts have proposed protocols for privacy-preserving ML inference [56, 57, 58, 59, 60, 61, 62] using cryptographic techniques such as homomorphic encryption (HE), trusted execution environment (TEE), and secure multi-party computation (SMPC). SMPC is preferred over other approaches, as it is computationally less expensive, does not require specialized hardware, and achieves accurate results while ensuring information-theoretical security [63, 64]. Many recent works [65, 60, 57, 58, 66, 59] have developed SMPC protocols to perform major operations in neural networks, such as activation and convolution [56, 67, 68].

Applying SMPC to GNNs faces major challenges due to the unique structure and operations of GNNs. Specifically, to protect the graph data, no parties are allowed to access or infer the graph structure and the node feature information. Since GNNs typically employ message passing to propagate information through the entire graph, it is particularly difficult to perform this operation without allowing different parties to access the full graph. Furthermore, it is challenging to design an efficient algorithm to secure the computations in the feature transformation layers (FTL) in GNN, which is required to protect the intermediate and final results, thereby safeguarding the model parameters and node features.

In this dissertation, we propose CryptGNN, a secure and efficient inference solution for GNN models, deployed as MLaaS in the cloud. To ensure privacy and high performance, we develop SMPC protocols [28, 29, 11] that enable a set of cloud providers (parties) to perform operations in the MPL and FTL layers of the GNN model on clients' input graphs, where both the model and graph data are encrypted in an additive secret-shared format. We implement the complete CryptGNN system using our novel components and conduct comprehensive theoretical analysis and empirical experiments to demonstrate the security, effectiveness, and efficiency of CryptGNN.

1.3 Privacy-preserving Indoor Place Prediction

Ensuring privacy in indoor place-level prediction systems is a critical challenge, as these systems inherently rely on users' mobility data to infer their future locations. With the development of indoor positioning technology and the widespread availability of mobile and wearable devices, there has been an explosive growth in the amount of indoor mobile trajectory data [69]. While location prediction can enhance various applications, it also raises significant privacy concerns due to the sensitive nature of the data. Depending on the data collection approach, unauthorized external parties may be able to track a user's movements. Montjoye et al. [18] demonstrated that human mobility patterns are highly unique, and it is possible to re-identify the movement traces of specific individuals even in a sparse, large-scale, and imprecise mobility dataset. Moreover, studies have shown that human mobility is highly predictable, with 93% of user movements being predictable [70]. This predictability underscores the need for robust privacy protection in data processing and prediction systems, as knowledge of future locations can enable intrusive tracking.

Knowing the places to be visited by a moving user has positive implications in many application scenarios. For example, an assisted living application may predict the place where an elderly person goes and guide the person to follow the safest path. In a smart home, a door can be unlocked automatically, and the lights can be turned on if a user is predicted to go to that room, or a smart music system can adjust its volume to provide a better user experience as the user is predicted to move to another place. Other applications may warn the user that the WiFi signal strength is weak at the predicted place to improve the customer experience of home Internet services, or the phone can change the settings automatically for user privacy before reaching a common area in a shared living space (e.g., turn off sound notifications). However, without adequate privacy safeguards, these benefits come at the risk of exposing users' movement patterns to unauthorized parties.

The current solutions developed for indoor localization or prediction systems do not ensure privacy protection due to one or more of the following issues. Firstly, they employ complex infrastructure to determine the user's location because GPS accuracy is limited inside buildings. For instance, indoor localization based on visual anchors [71] needs to attach pre-defined image tags at certain known locations in the environment. Other types of indoor localization use multiple ultra-wideband (UWB) anchor nodes [72] with known coordinates, and the user needs to carry a UWB tag to communicate with the anchors to estimate location. In addition, different forms of wireless fingerprinting and multi-lateration [73] have been explored for indoor localization, including WiFi [74], FM radio [75], RFID [76], acoustic [77], GSM [78], light [79], and magnetism [80]. These solutions are dangerous from a privacy point of view because they may be able to collect and store the user's locations or trajectories on systems that are not under the user's control [81]. For instance, an attacker could use RFID or UWB tags attached to the user to track their movement within the building. Additionally, the system can estimate the user's position by measuring their distance from multiple Bluetooth beacons or WiFi access points. Secondly, some studies use a localization server to access the user's mobility data, which allows it to monitor the user's activities within the building, including continuous tracking, storing historical records of the user's location, and sharing these data with third parties without the user's knowledge.

A possible approach to address privacy risks involves using a simple infrastructure that can not track users and performs all localization computations on the user's device. This approach is effective in achieving privacy-by-design since the user's data stays on the phone and is not shared with any external entity. Moreover, this system can also be customized to improve personalization, which provides two benefits for place prediction: (i) allows individual users to name places in a way that makes sense for them (i.e., semantic naming), and (ii) improves prediction accuracy

because different people have different frequently visited places. However, designing an efficient system for data collection, analysis, training, and inference to minimize energy consumption and manual effort poses a significant challenge.

To solve these challenges, this dissertation proposes GoPlaces, a place prediction smart phone app that does not require any infrastructure, except for one cheap off-the-shelf WiFi AP that supports ranging using WiFi Round Trip Time (WiFi-RTT). GoPlaces enables personalized place naming and place prediction through its on-the-phone data collection, training, and inference algorithms. Additionally, GoPlaces ensures better privacy protection for user locations and trajectories since the data never leaves the device, and the AP cannot track the user. We implemented GoPlaces as an Android app that collects and analyzes data, trains models, and performs inference of destination places with low latency and low resource consumption.

1.4 Contributions of the Proposed Research

1.4.1 Privacy-preserving federated multi-task learning

This dissertation introduces FedMTL, a novel algorithm designed to support privacy-preserving MTL in the FL setting. FedMTL algorithm facilitates collaborative learning among client models while ensuring that sensitive information remains protected. It achieves this by employing a secure aggregation mechanism that enables task-based model collaboration without exposing client-specific model parameters, task distributions, or dataset sizes.

Figure 1.1 illustrates the high-level system architecture of FedMTL. FedMTL employs new protocols to enhance the capabilities of each client model by enabling collaboration among models from the participating clients based on similarities in the tasks they execute. It aims to develop a low resource-consumption approach, enabling straightforward integration with established privacy-preserving techniques for secure aggregation. In particular, the secure version of FedMTL is designed to preserve the

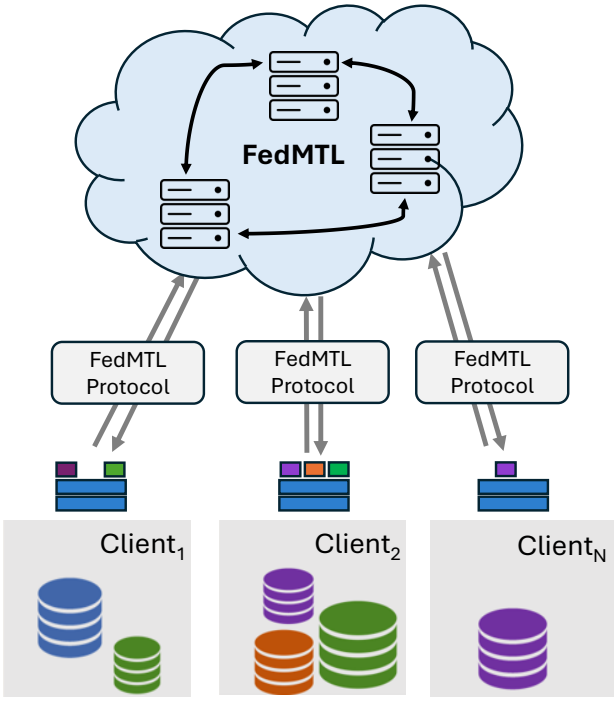


Figure 1.1 FedMTL system architecture.

privacy of (a) the parameters of the clients' models, (b) the types and numbers of tasks executed by each client, and (c) the size of the training dataset. To the best of our knowledge, FedMTL is the first framework that supports privacy-preserving MTL in an FL setting.

FedMTL generates personalized MTL models based on task similarities, which are determined by analyzing the parameters of the task-specific layers in trained models. It assumes that the supported tasks have unique IDs and that clients are aware of the task IDs for which their models have been trained. Furthermore, it employs a hard parameter-sharing architecture [21] for local MTL models, wherein hidden layers are shared across related tasks while task-specific output layers are maintained. This architecture enables the MTL model to understand the overall data structure while gaining expertise in multiple tasks. The aggregator in FedMTL applies layer-wise aggregation policies, computing different sets of weights for clients' model parameters to enhance personalization in MTL models.

The secure version of FedMTL leverages a secure multi-party computation approach, allowing clients to upload their model and task-specific information in an additive secret-sharing format. To prevent task type inference based on the size of the uploaded data, the FedMTL protocol on the client side injects fake data into the task-specific parameters, ensuring that all tasks have the same number of parameters. Additionally, to obscure the actual number of tasks, the client introduces fake task layers. The aggregator servers then execute the secure FedMTL aggregation protocol, generating personalized MTL models while preserving user data privacy. Since the FedMTL algorithm is designed to be computable using standard additive secret-sharing techniques, data privacy remains protected throughout the computation. Finally, the client downloads the personalized model from the aggregator servers and extracts the necessary parameters by removing the injected fake task layers.

To evaluate the effectiveness of FedMTL, we implement the proposed FL aggregation algorithm and privacy-preserving protocols using PyTorch and conduct experiments on benchmark datasets. The results show that FedMTL outperforms baseline approaches when each client trains an MTL model for distinct sets of two tasks. Furthermore, FedMTL demonstrates its adaptability in cases where clients' MTL models are trained on different numbers of tasks — a scenario not directly supported by the state-of-the-art approaches. Additionally, we conduct ablation studies that demonstrate the significance of our aggregation algorithm compared to alternative methods. We use CrypTen [11] to implement a secure version of FedMTL aggregation and achieve the same accuracy performance as the plain text while preserving the privacy of client data.

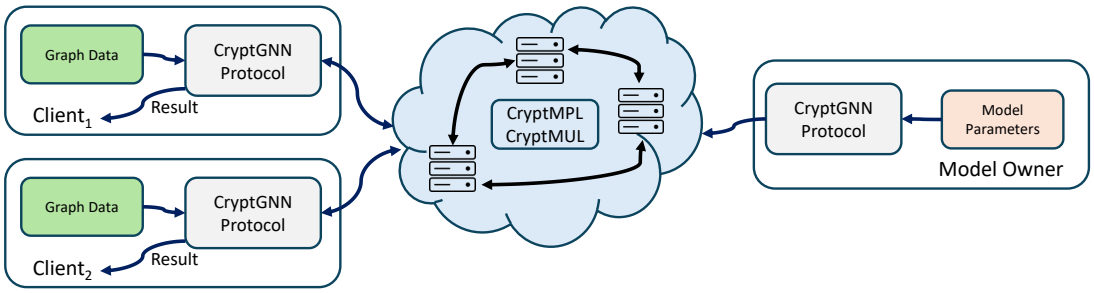


Figure 1.2 CryptGNN system architecture.

1.4.2 Secure inference for graph neural networks

This dissertation presents CryptGNN, a cloud-based ML-as-a-service (MLaaS) that provides a secure and effective inference solution for graph neural networks (GNNs). Figure 1.2 shows the high-level system architecture of CryptGNN. CryptGNN safeguards the user's graph input data from the cloud provider and the model owner, and protects the model from the cloud provider and the clients. It employs SMPC in the cloud, enabling privacy-preserving GNN inference even if P-1 out of P parties collude with each other. The cloud providers compute the forward pass of the model, as the model architecture is known, while CryptGNN protects the model parameters in an additive secret-shared format. To protect the client input graph, CryptGNN encrypts the node features and the graph structure in an additive secret-shared format before uploading the data to the SMPC parties.

CryptGNN consists of two novel protocols to enable privacy-preserving inference of encrypted GNN models on encrypted input graph data in the cloud. **CryptMPL** executes the message-passing layer, while preserving the privacy of input data (i.e., node features and graph structure). It employs novel operations that involve rotation and shifting of input data to securely perform the read and write steps in the MPLs. These operations use a data preprocessing step at the client, which helps the SMPC parties mask the private data, thereby eliminating the need for any trusted servers. **CryptMUL** executes the secure multiplication operations required for evaluating the linear and nonlinear FTLs in GNN models. In this protocol, the SMPC parties

conduct offline preprocessing to generate auxiliary data that allows them to execute matrix and element-wise multiplications without expensive cryptograpic operations or relying on a trusted server. Both protocols can work with any number of SMPC parties. CryptMPL and CryptMUL are invoked from the GNN models to guarantee the cloud providers do not learn partial results of functions executed over secure inputs or the final inference results.

To implement a full CryptGNN system, we leverage several privacy-preserving components from CrypTen [11] in addition to the novel components mentioned earlier. The CryptGNN system can perform privacy-preserving inference using a wide variety of GNN models. We conducted comprehensive theoretical analyses and empirical experiments to demonstrate the security, effectiveness, and efficiency of CryptGNN. Notably, CryptGNN protects both the model and user input data while producing the same inference results as a non-secure version.

1.4.3 Privacy-preserving personalized indoor place prediction

In this dissertation, we present GoPlaces, a place prediction smart phone app. Through its privacy-first design, GoPlaces empowers users with a secure and personalized place prediction system, built on a simple, privacy-preserving infrastructure, mitigating privacy risks while utilizing the advantages of mobility-based predictive services. This work demonstrates that privacy and functionality can coexist, offering a robust solution to the growing concerns surrounding indoor mobility data security.

GoPlaces does not require any infrastructure, except for one cheap off-the-shelf WiFi AP that supports ranging using WiFi Round Trip Time (WiFi-RTT). Due to this feature, such APs can become common in houses, shops, or workplaces in the near future. However, knowing the distance from a single AP is not enough to localize the user. GoPlaces' idea is to detect the user's walking trajectories by augmenting the WiFi-RTT distance measurements with mobile sensor measurements, specifically

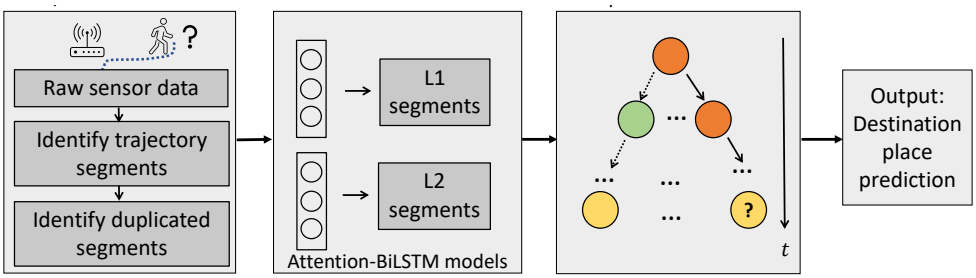


Figure 1.3 Framework of GoPlaces.

accelerometer and magnetometer data. Although the data collected from sensors are usually noisy, GoPlaces finds similar patterns for the sequence of data collected along a trajectory, and it identifies a trajectory by analyzing the walking direction and the series of WiFi-RTT distance measurements.

GoPlaces enables personalized place naming and place prediction through its on-the-phone data collection, training, and inference algorithms. By design, GoPlaces also leads to better privacy protection of user's locations and trajectories because the user's data never leaves the phone. Furthermore, the single WiFi-RTT AP cannot localize the users accurately to detect their trajectories. The high-level software framework of GoPlaces is shown in Figure 1.3. GoPlaces divides the trajectories into smaller segments, which are automatically identified and labeled based on changes of direction in the trajectories. We designed an attention-based bidirectional long short-term memory (attention-BiLSTM) model that learns and classifies the segments traversed by the user. This model effectively learns the trends of walking direction and WiFi-RTT distance features and finds the correlation between them. The trajectories, as sequences of segments, are stored in a prediction tree, and used to infer the user's destination place. During inference, GoPlaces checks possible combinations of segments, assigns weights to each place, and predicts the place with the highest confidence value.

We implemented GoPlaces in Android and evaluated it in several indoor spaces, using a Google WiFi-RTT AP and commodity smart phones. The place prediction

accuracy depends on the percentage of progression by the user toward the destination. The higher this percentage, the higher the accuracy. The experimental results demonstrate prediction accuracy as high as 86% when 90% of the trajectory is traveled, and as high as 74% when 75% of the trajectory is traveled. Based on the characteristics of the WiFi-RTT distance and walking direction patterns, we designed a technique to collect and label trajectory data automatically, which substantially reduces the manual effort required to collect training data. Furthermore, we developed a data augmentation process to enrich the training dataset with synthetic data that improves the prediction accuracy by as much as 140% when compared to a model that uses only user-collected data. We also demonstrate that GoPlaces is feasible in real life because it has low latency and low resource consumption on the phones. With a full battery, a Google Pixel 4 phone can execute 0.5 million predictions, and each inference takes 142 ms. For application scenarios that require lower latency, we demonstrate that GoPlaces' parameters can be configured to have faster inference, with a slight drop in prediction accuracy.

1.5 Structure of the Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 provides a review of the literature related to this study. Chapter 3 introduces FedMTL - a privacy-preserving system designed for multi-task learning within a federated learning framework. Chapter 4 describes CryptGNN - a secure inference service for graph neural networks. Chapter 5 presents GoPlaces - an app for personalized indoor place prediction. Finally, Chapter 6 concludes the dissertation.

CHAPTER 2

LITERATURE REVIEW

Machine learning is a data-driven technology that requires vast amounts of data to build accurate models. Large-scale data collection and use pose significant privacy concerns as a result of recent data breaches [82, 83]. Many techniques and frameworks have been proposed to address the challenge of privacy preservation in machine learning. Some popular frameworks for privacy-preserving machine learning include Google's TF Encrypted [84], Microsoft's SEAL [85], OpenMined's PySyft [86] and Meta's CrypTen [87]. Through the exploitation of these frameworks and the development of new techniques, several research works are being conducted to build systems that preserve privacy in machine learning.

For the training of a generalized model, it may be necessary to collect data from many users at a single location. In such cases, it is required to ensure that sensitive data remains protected while still enabling machine learning algorithms to be trained and deployed. To preserve data privacy, several techniques such as differential privacy [8, 7], homomorphic encryption [9, 10], trusted processors (SGX) [88], secure multi-party computation [89, 12] are employed by different research works. However, those techniques face several challenges such as achieving a balance between privacy and utility [13], computation and communication overhead [14] etc.

With the increasing use of mobile devices and the amount of sensitive data that they collect, it has become more common to train machine learning models on-device [90] which can be tailored to each consumer. There are several machine learning frameworks that are specifically designed for mobile devices. For instance, Google's TensorFlow Lite [16] is a mobile-friendly version of TensorFlow that can be used to train machine learning models on mobile devices. Additionally, by leveraging

hardware capabilities and optimization techniques, Apple's Core ML [91] framework supports machine learning model training on iOS devices, helping models stay relevant to user behavior without compromising privacy. However, machine learning on mobile devices presents unique challenges, such as limited computational resources and limited battery life.

Although mobile computation enables personalization, it does not utilize knowledge sharing among users, thus losing the benefit of generalization. To overcome this limitation, federated learning (FL) [20, 92, 93] is a popular approach that enables multiple devices or nodes to collaborate and train a model without sharing their data with a central server. In FL, the training process takes place locally on each device, and only the updated model parameters are sent to a central server for aggregation and updating of the global model. However, malicious attackers can exploit an ML model through various attacks such as membership inference attacks [94, 95], model inversion attacks [96, 97], and privacy leakage from gradients [6, 98, 99] exchanged in distributed ML scenarios. Hence, it is crucial to adopt appropriate measures to ensure the privacy and security of user data and model parameters.

In this dissertation, we focus on the design, implementation, and evaluation of systems for privacy-preserving machine learning for several application cases. This chapter provides a review of related works in the areas of federated multi-task learning, secure machine learning as a service, secure inference of graph neural networks, indoor localization and place prediction.

2.1 Privacy-Preserving Federated Multi-task Learning

Multi-task learning (MTL) [21, 100, 101, 102] enhances task performance and reduces training and inference time by simultaneously learning related tasks. To learn the generic patterns of data, as well as task-specific features, several MTL architectures [103, 104] adopt hard parameter sharing, which utilizes the same hidden

layers for all tasks and incorporates task-specific output layers. However, these approaches typically assume a centralized setup, where the model trainer has full access to all tasks and data. In this dissertation, we present FedMTL, which addresses scenarios where each client has a small amount of data for training an MTL model focused on a subset of all the supported tasks in the system, exploring collaboration among clients to enhance the performance of the MTL models.

Federated learning (FL) [22] enables collaboration among clients without sharing private datasets. In FL, each client trains a model using its private data, and uploads the model to a central server for aggregation with models from other clients. To address data heterogeneity, several works [105, 106] aim to learn a personalized model for each client, enhancing compatibility with highly non-IID clients. FedFomo [107] allows each client to compute personalized aggregation weights by minimizing validation loss using model information from other clients, leading to high computational and communication costs, as well as privacy concerns. Ghosh et al. [106] introduces a clustering algorithm to represent relationships among clients and aggregates models within client groups. FedMSplit [108] enables federated training on multi-modal distributed data, taking into account modality incongruity across clients while assuming the task remains consistent. Despite addressing data heterogeneity, all these methods overlook the task heterogeneity across clients.

To address task heterogeneity, MOCHA [23] employs an optimization algorithm to fit related models trained for separate tasks. FedU [109] formulates the federated MTL problem using Laplacian regularization to explicitly leverage the relationships among the models. Since these approaches update models for all clients simultaneously, it is necessary to re-run the algorithms to support a newly added client, leading to waste in bandwidth and computational resources. FedEM [110] enables each client to learn personalized models as mixtures of several component models. Cai et al. [111] employs a clustering-based training approach to address task

heterogeneity by enabling each client to infer its similarity with others by comparing their layer-wise weight updates sent to the server, and then determining how to aggregate weights for selected similar clients. In contrast to our work, all these prior studies assume each client executing a single task.

Although MAS [112] enables clients to train local models for multiple tasks, it assumes the clients have data and labels for all possible tasks. MAS allows the server to pick a subset of tasks, and the clients train models for that specific subset, ensuring task consistency across all clients. In contrast, FedMTL assumes that each client holds data for only a subset of tasks, which may differ among clients, making it a more realistic scenario. Furthermore, FedMTL does not require collecting all clients' models for aggregation, and it enhances personalized MTL models by leveraging the similarities in task-specific parameters among participating clients' models.

To avoid data leakage [113, 114] from uploaded model parameters, several studies [115, 116, 117] propose secure aggregation of local models. These studies employ techniques such as secure multi-party computation (SMPC) [118], homomorphic encryption (HE) [119], or a combination of both. However, none of these works are designed to support MTL where clients' model architectures are different, each working on a different set of tasks. FedMTL introduces an algorithm designed for computing personalized MTL models with minimal overhead, which has the flexibility to integrate with established privacy-preserving techniques for secure aggregation.

2.2 Secure Machine Learning as a Service

With the increasing growth of cloud services, it is now possible to train and deploy machine learning models on the infrastructure of cloud providers. These types of Machine Learning as a Service (MLaaS) are currently offered by different industries including Google [120], Microsoft [121], IBM [122] and Amazon [123].

Recently, there has been an increased interest in developing methods for the training and inference of neural networks in a secure environment. There are several existing works [56, 57, 58, 59, 60, 61, 62] that provide a secure inference service for a model trained on a proprietary dataset. They use different cryptographic techniques, such as homomorphic encryption [59], garbled circuits [56, 124], and SMPC/secret sharing [68]. In comparison with SMPC, homomorphic encryption and garbled circuits are relatively expensive and usually incur large performance overheads. All of these studies work with unstructured data, such as images and text.

2.3 Secure Inference of Graph Neural Networks

Graph Neural Networks (GNNs) is a popular type of neural model that apply deep learning techniques to graph-structured data [38, 39, 40]. However, providing secure inference for graph-structured data using SMPC is challenging, as it requires protecting both the data and the graph structure. A major challenge is to protect the graph structure in GNNs that use a message passing layer, where each node aggregates the feature vectors of its neighbors to compute its new feature vector [125, 38, 39, 40, 126]. Preserving the privacy of the graph structure in this layer is challenging because message exchange through edges requires exploiting the graph structure without revealing edge source and destination nodes.

To address this challenge, Wang et al. proposed SecGNN [127], an SMPC-based solution, that encrypts graph data using 2-out-of-2 additive secret sharing and a trusted server to assist in the computations. This design has several problems. First, it relies on a trusted server, which is a strong assumption in practice. Second, it works only for two parties, which cannot collude with each other or with the trusted server. In this dissertation we propose CryptGNN, which works for an arbitrary number of

parties, even when P-1 out of the P parties collude with each other. Thus, CryptGNN offers a significantly stronger security guarantee.

One intuitive approach for secure message-passing layer (MPL) computation is to represent node features in an encrypted feature matrix and the graph structure in an encrypted adjacency matrix, and then employ state-of-the-art matrix multiplication methods [128, 11]. However, this incurs high communication overhead, may require an additional trusted party, and results in unnecessary computations for sparse real-world graphs. CryptGNN represents the graph structure as source/destination arrays and achieves substantially lower overhead. To reduce the computation in the matrix representation, CryptoGCN [129] proposed an efficient matrix multiplication using homomorphic encryption (HE). However, it does not protect the graph structure and assumes the GNN model parameters are in plain-text format. In fact, it is challenging to protect the client input graph and model parameters with HE for two reasons: (a) Since the data are encrypted by two different entities (client and model owner), there is an additional overhead for bootstrapping [130] and key relinearization [131]; (b) It requires complex operations to create an adjacency matrix-aware formatting [129] of the graph and execute MPL using that matrix. CryptGNN uses a much cheaper A-SS approach in SMPC, reducing the overall overhead.

ORAM techniques [132, 133, 134] could enable a client to access graph data without leaking the access patterns, and thus the graph structure; however, they require the client to download and decrypt data, making it more expensive to compute at potentially resource-constrained clients. In contrast, CryptGNN's secure message-passing protocol CryptMPL offers a more efficient approach that reduces client involvement in GNN inference computations.

Several approaches exist for secure element-wise and matrix multiplication for FTLs in SMPC settings. CrypTen [11] requires the parties to communicate with a

trusted third party. This is not required in CryptGNN. Protocols using HE [135] and oblivious transfer [136] tend to have a high overhead, making them impractical, especially for numerous inference requests. CryptGNN's secure protocol CryptMUL employs HE or OT-based techniques only during preprocessing, enabling the design of FTLs with very low overhead for computing multiple inference requests.

2.4 Indoor Localization and Place Prediction

Location prediction systems [137, 138, 139, 140, 141, 142, 143, 144], which predict location points, have been widely studied. More recently, place prediction has become an active research field [145, 146, 147, 148]. The quality of the prediction depends on the localization accuracy of users' historical data and current position, which is difficult to determine indoors where GPS cannot be used effectively. The remaining of this subsection places our work in the context of related work on indoor localization and indoor place prediction.

There has been considerable interest in developing accurate indoor localization [149, 150, 151]. Radio technology, especially RSSI measurements [152, 153], is the most widely employed. In typical indoor environments, RSSI is affected by dense multipath fading [154] effects and its overall accuracy is low. As signal strength from a single AP is not enough to estimate distance, signals from several APs are recorded at each position, and the user's location is estimated by finding the best match between the current RSSIs and the historical RSSIs. The major drawbacks of this method are the requirements to (i) have 3 or more APs in the transmission range of the mobile devices, and (ii) build a fingerprint database [155].

Another popular approach for indoor localization is CSI [156, 157, 158], which provides more information compared to RSSI. Most CSI-based solutions require a fingerprinting approach, and thus have the same disadvantages as the RSSI-based solutions. Chronos [159] does not use fingerprinting, as it emulates a wideband radio

using CSI captured by multiple packets and processes the CSI to infer time-of-flight and device locations. One problem with this solution is that mobile operating systems do not make physical layer information, such as CSI, accessible to apps.

Several studies [151, 160, 161, 162, 163] attempt to localize users using a single WiFi access point by applying techniques that leverage multipath effects to extract signal metrics such as angle-of-arrival (AoA), angle-of-departure (AoD), and time-of-flight (ToF). However, all of these techniques require WiFi access points equipped with antenna arrays, typically consisting of three or more antennas, whereas most commodity routers are equipped with only two antennas. Although access points with antenna arrays can potentially enhance distance measurements by integrating multiple techniques, they are affected by multipath fading and require precise clock synchronization, which increases both cost and complexity.

Other solutions for indoor localization involve anchors with known locations (visual or RF). Xiao et al. [164] propose a system based on static objects as anchors and estimates the position of the user based on features extracted from photos. Its main drawbacks include the requirement for good lighting conditions and the high computational cost on the phone. If the execution is at the server-side, privacy risks become a drawback. Furthermore, the user needs to hold the phone to take photos. Other studies [165, 166, 167] use artificial markers (e.g., barcodes), which are more robust and work well under varying lighting conditions. However, these solutions require the deployment of markers in the indoor space.

Yu et al. [168] use an Ultra-wideband (UWB) protocol for indoor localization. In this approach, users carry tags that transmit UWB signals to fixed sensors (i.e., anchors), and the user's position is estimated using the time-of-arrival method. The main disadvantage of this approach is that it requires multiple anchors to be placed in the indoor space. Furthermore, it requires users to carry additional tags, and the user location is tracked by the infrastructure.

There are several studies on indoor localization systems that use acoustic signals beyond the audible range. Mobile devices can work as acoustically passive [169] or active [170, 171]. Similar to other anchor-based systems, these solutions require substantial infrastructure support and can also track the users.

To mitigate the need for expensive or difficult to deploy infrastructure, GoPlaces takes advantage of WiFi-RTT in the IEEE 802.11-2016 standard, which created a WiFi fine timing measurement (FTM) protocol, commonly known as WiFi-RTT. This protocol defines a WiFi-based two-way ranging approach that makes WiFi ranging more robust and accurate (e.g., meter-level positioning accuracy). Therefore, a smart phone can estimate its distance from APs that support WiFi-RTT [172, 173]. This technology has been incorporated into commercial products and is currently supported by different smart phones and WiFi AP manufacturers.

WiFi-RTT alone, however, cannot provide a solution for the data needed for place prediction, due to two reasons. First, the WiFi-RTT estimated distance places the user on a circle around the AP, not at an exact location. Second, the WiFi-RTT measurements are noisy and lead to significant errors in the distance estimation [174, 175, 176]. Therefore, GoPlaces leverages sensors in smart phones to solve these problems. There are several studies [177, 178] that use only inertial sensors to track the path of a user from a known initial position. A significant drawback of these solutions is the error propagation of sensor readings; even a small error is magnified through integration, and the localization error accumulates with increasing walking distance. GoPlaces is unique in its approach to fuse data from WiFi-RTT distance estimation and inertial sensors. These data together with our algorithms for segment classification and trajectory matching lead to high accuracy place prediction. Furthermore, these data can be collected and preprocessed efficiently on resource-constrained phones.

Existing place prediction algorithms, based on Markov chains [179], Hidden Markov Models [180], or Bayesian Networks [143], rely on coordinate-level locations of the users. Such algorithms can work well only with highly accurate location data, which requires substantial localization infrastructure. Since our aim is to use minimal infrastructure (i.e., WiFi-RTT AP), the location data will not be very accurate, and therefore existing solutions cannot be applied in our settings. Therefore, GoPlaces identifies trajectories and segments based on the fusion of WiFi-RTT data and inertial sensor data. By splitting the trajectories in a few segments, instead of many locations such as in Markov chains or Bayesian networks, GoPlaces avoids problems related to dimensionality (e.g., state space explosion). Another advantage of GoPlaces compared to existing place prediction algorithms is its personalization for place naming and training/inference, which improves prediction accuracy and makes the results more meaningful to individual users.

In its use of DL models, GoPlaces shares ideas and methods with work on indoor localization [181, 182, 183], which demonstrated that deep neural networks perform well despite signal fluctuations and noise effects. DL has also been used to predict outdoor locations [144], motion mode detection [184], trajectory construction [183], etc. The main difference between these works and GoPlaces consists in the problem we address and its specific settings: to perform accurate place prediction with fused data from a single WiFi-RTT supported access point and inertial sensors, we had to come up with new algorithms for segment detection, segment classification, and place prediction.

In terms of privacy, the WiFi-RTT protocol may leak location-sensitive information, as an attacker can determine the distance between itself and the user [185]. However, an attacker cannot localize a user without placing multiple observers in the environment. In addition, as GoPlaces identifies trajectories instead of coordinate-level locations, it does not require multiple APs, making it more

Table 2.1 Comparison Between GoPlaces and Related Work

	Papers	Does Not Need to Know the Initial Point	Deep Learning	Off-the-shelf Infrastructure	No Fingerprint (Less Manual Effort)	Privacy Protection	Indoor Place Prediction	Implementation on Mobile OS
RSSI Based	[73, 74, 81, 137, 142, 150, 152, 153, 155, 181]	○	○	×	○	○	×	○
CSI Based	[149, 151, 156, 157, 158, 159, 182]	○	○	○	○	○	×	×
GPS Based	[143, 144, 145, 146, 146]	○	○	○	○	○	×	○
Inertial Sensors based	[186, 183, 177, 178]	×	○	○	○	○	×	○
Multipath Based	[151, 160, 161, 162, 163]	○	○	○	○	○	×	○
GSM, FM Radio	[75, 78]	○	○	○	○	×	×	○
Others (Visual, UWB, BLE, Acoustic, Light, Magnetism)	[71, 72, 76, 77, 79, 80, 164, 165, 166, 167, 169, 170, 171, 168]	○	○	×	○	○	×	○
WiFi-RTT	[172, 173, 174, 175, 176]	○	○	○	○	○	×	○
Sequence-based (Sequence of place names or APs)	[137, 138, 139, 140, 141]	○	○	×	–	–	○	–
GoPlaces		✓	✓	✓	✓	✓	✓	✓

✓ Supported × Not Supported ○ Supported by some papers in the group – Not Applicable

secure from external attacks. Furthermore, unlike most indoor localization systems, GoPlaces stores the data and performs all computations locally on the phone, which helps reduce the privacy risks.

Table 2.1 summarizes the studies related to our work. While there are studies on indoor localization, the indoor place prediction has been explored by only a limited number of works. These studies often rely on sequences of place names or access points (APs), require complex infrastructure, and are not implemented on mobile OS. Other studies, which utilize various signals (RSSI, CSI, visual data, inertial sensors, etc) aim to localize the user but cannot be directly applied to the place prediction task using off-the-shelf infrastructure while ensuring user location privacy.

2.5 Chapter Summary

This chapter discussed the existing studies related to privacy requirements in the machine learning pipeline and techniques used to develop systems preserving data privacy. Firstly, we explored existing works on multi-task learning, as well as approaches for addressing task heterogeneity and privacy challenges in federated learning. Next, we presented existing works concerning privacy in ML-as-a-Service (MLaaS) in the cloud and discussed the challenges associated with supporting graph neural networks in that setting. Finally, we discussed existing solutions for indoor place prediction, which suffer from both low privacy and personalization issue.

CHAPTER 3

FEDMTL: PRIVACY-PRESERVING FEDERATED MULTI-TASK LEARNING

This chapter presents **FedMTL**, a novel algorithm to support multi-task learning (MTL) in the federated learning (FL) setting. FedMTL uses new protocols to enhance the capabilities of each client model by enabling collaboration among models from the participating clients based on similarities in the tasks they execute. FedMTL generates personalized MTL models based on task similarities, which are determined by analyzing the parameters for the task-specific layers of the trained models. FedMTL uses novel protocols to enhance performance by encouraging collaborations among clients based on similarities in the tasks they execute. FedMTL assumes that the supported tasks have unique IDs and the clients are aware of the task IDs for which the model has been trained. Furthermore, it uses a hard parameter sharing-based architecture [21] for local MTL models. This architecture involves sharing hidden layers for the related tasks while maintaining task-specific output layers. The architecture enables the MTL model to understand the overall data structure while gaining expertise in multiple tasks. The aggregator of FedMTL applies layer-wise aggregation policies and computes different sets of weights on clients' model parameters to improve the personalization of the MTL models. FedMTL aims to develop a low resource-consumption approach, enabling straightforward integration with established privacy-preserving techniques for secure aggregation. To the best of our knowledge, FedMTL is the first framework that supports privacy-preserving MTL in FL settings.

In this chapter, Section 3.1 defines the FedMTL aggregation problem, and Section 3.2 describes the FedMTL aggregation algorithm. Section 3.3 presents the privacy-preserving protocols for FedMTL. Section 3.4 shows the system and security

analysis of FedMTL, and Section 3.5 presents the experimental evaluation of FedMTL. The chapter is summarized in Section 3.6.

3.1 FedMTL Aggregation Problem Definition

The goal of FedMTL is to design FL aggregation protocols that enhance personalized MTL models, without imposing heavy computational burdens on devices. We consider a set of tasks $\mathbf{U} = \{T_1, \ldots, T_K\}$ supported by FedMTL, where the total number of supported tasks, $K = |\mathbf{U}|$. Each task T_k, $k \in \{1, \ldots, K\}$, has a unique integer ID. We consider a hard parameter sharing architecture for the multi-task models, with the model parameters represented as $\mathbf{W} = \{\mathbf{W}^S, \mathbf{W}^T\}$, where $\mathbf{W}^S$ and $\mathbf{W}^{T_k}$ represent the parameters for the shared layers and task T_k, respectively (shown in Figure 3.1).

In each round, FedMTL picks a set of clients $\mathbf{C}$, where the number of clients is $N = |\mathbf{C}|$. Each client $C_i \in \mathbf{C}$ trains an MTL model for a subset of tasks $\mathbf{U}_i \subseteq \mathbf{U}$ using data $D_i = \{\mathbf{X}_i, \mathbf{Y}_i\}$, where $\mathbf{X}_i$ represents the input features and $\mathbf{Y}_i$ denotes the list of labels for $K_i = |\mathbf{U}_i|$ number of tasks, as $\mathbf{Y}_{ik}$ is the label for task $T_k \in \mathbf{U}_i$. The set of tasks executed by each client can be different. Training on local data, each client obtains the local model $\mathbf{W}_i$, which consists of the parameters of the shared layer $\mathbf{W}_i^S$ and the parameters of the task-specific layers (task layers) $\mathbf{W}_i^T$. We use $\mathbf{W}_i^T[j]$ to represent the parameters for j-th task and $\mathbf{W}_i^{T_k}$ to represent the parameters for the task with ID k, i.e., T_k.

This work focuses on the aggregation protocol employed by the server to generate personalized MTL models for each client. As shown in Figure 3.1, the aggregator $\mathbf{A}$ collects the model parameters from N clients $\mathbf{W}_i, i \in \{1, \ldots, N\}$ and aggregates these parameters to get the personalized model parameters $\mathbf{W}_i^* = \{\mathbf{W}_i^{S^*}, \mathbf{W}_i^{T^*}\}$, where $\mathbf{W}_i^{S^*}$ is the aggregated model parameters of the shared layer and $\mathbf{W}_i^{T_k^*}$ is the aggregated layer parameters for task T_k.

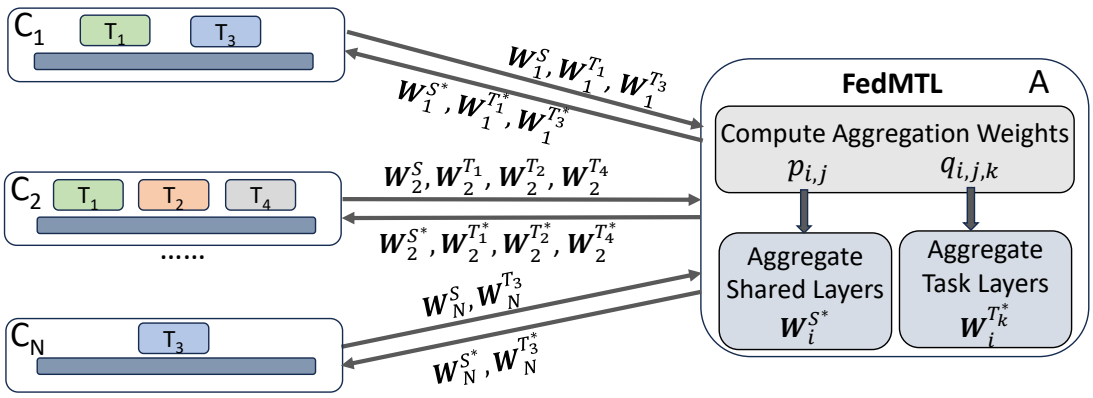


Figure 3.1 FedMTL problem setup.

In the case of personalized FL, the optimization problem can be represented as in Equation (3.1), where the function $f_i : \mathbb{R}^d \to \mathbb{R}$ denotes the expected loss over the data distribution of C_i.

$$\forall i, \mathbf{W}_i^* = \arg\min_{w_i} f_i(\mathbf{W}_i) \tag{3.1}$$

To achieve this, we define $p_{i,j}$ as the weights of client C_i to aggregate the parameters of the shared layers from the local model of client C_j, as shown in Equation (3.2). Additionally, we define $q_{i,j,k}$ as the weights to aggregate the parameters of task layers of C_i using the parameters of the local model from C_j for task T_k, as shown in Equation (3.3). For conventional FL that supports only single-task learning, $\mathbf{W}_i^S$ contains $\mathbf{W}_i^{T_k}$ and Equation (3.2) is similar to the vanilla FedAvg if $p_{i,j} = 1$.

$$\mathbf{W}_i^{S^*} = \frac{\sum_{j=1}^{N} p_{i,j} \mathbf{W}_j^S}{\sum_{j=1}^{N} p_{i,j}} \tag{3.2}$$

$$\mathbf{W}_i^{T_k^*} = \frac{\sum_{j=1}^{N} q_{i,j,k} \mathbf{W}_j^{T_k}}{\sum_{j=1}^{N} q_{i,j,k}} \tag{3.3}$$

The FedMTL aggregation problem is to determine the weights $p_{i,j}$ and $q_{i,j,k}$ that enhance personalized MTL models in an FL setting.

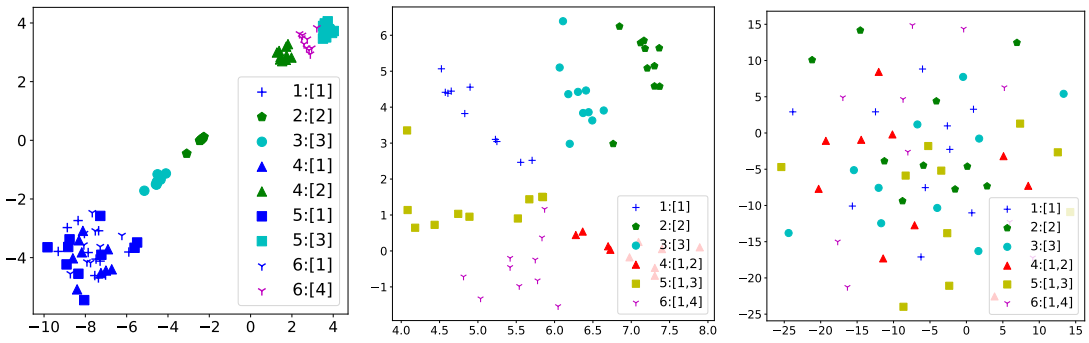
3.2 FedMTL Algorithm

This section first presents our analysis (Subsection 3.2.1) of the characteristics of the parameters across different layers of the local MTL models and the similarities of the models trained by the users executing different sets of tasks. This analysis provides insights into the rationale behind our algorithm's design by highlighting the similarities in parameters at different layers of the local models. Then, we introduce the aggregation algorithm (Subsection 3.2.2) of FedMTL that improves the personalized MTL models by leveraging similarities between task-specific parameters to aggregate local models.

3.2.1 MTL model parameter analysis

To inform the design of FedMTL, we consider at MTL model running over the CelebA dataset [187] and randomly partition it among 60 clients using the symmetric Dirichlet distribution. We consider four tasks, where each task T_k, $k \in \{1, \ldots, 4\}$ consists of detecting five different face attributes. The clients are grouped into six groups of equal size, where each group G_i, $i \in \{1, \ldots, 6\}$, trains an MTL model for the i-th set of tasks from the list: $[[T_1], [T_2], [T_3], [T_1, T_2], [T_1, T_3], [T_1, T_4]]$. As local MTL models, we use the LeNet [188] architecture as the base, with a task-specific linear layer for each task. We then collect the trained model parameters after executing 20 epochs and visualize those weight updates by t-SNE [189] in Figure 3.2.

We analyze the parameters of the MTL models in terms of (i) task layers and (ii) shared layers. Figure 3.2(a) shows the parameters of the task layers mapped into two dimensions, where the label $G_i : [T_k]$ represents the parameters $\mathbf{W}^{T_k}$ for the task T_k of the clients in group G_i. We observe that the parameters of the task layers are very similar for the clients performing the same set of tasks. Such similarity can even be across task groups; for example, parameters for task T_1 from all groups of clients are very close, even though the task is executed in conjunction with different tasks.



(a) (b) (c)

Figure 3.2 t-SNE visualization of 60 MTL model parameters for: (a) task layers, (b) top shared layers, (c) bottom shared layers.

We analyze the shared layers $\mathbf{W}^{S_l}$, $1 \leq l \leq L$, of the MTL models, where L is the number of shared layers. For a linear layer in the LeNet model, which is closer to the task layer (i.e., $l \to L$), the parameters exhibit similarity among clients within the same group (as shown in Figure 3.2(b)), as these models are trained for the same set of tasks. For some clients in groups G_4, G_5, G_6, these parameters are close to the parameters of the clients in group G_1, since there is a common task T_1 in all of these groups. Figure 3.2(c) shows the parameters of the convolution layer, which is the bottom layer (i.e., $l \to 1$) in the shared layers of the LeNet model. For these bottom layers, the parameters do not show any specific pattern. This analysis leads us to the following insight for the design of FedMTL: although the shared layers learn generic patterns of data, some layers are influenced more by the tasks, as these parameters are significantly updated by the gradients computed based on the losses associated with each task. Our experimental results in Section 3.5 further demonstrate that the parameters in task-specific layers are similar across clients performing the same task. Therefore, FedMTL should treat the shared layers and task-specific layers separately, prioritizing the parameters in the task-specific layers to evaluate similarity across clients' models.

3.2.2 FedMTL aggregation

When dealing with MTL in FL settings, the model parameters $\mathbf{W}_i$, cannot be directly aggregated due to the heterogeneity in model architecture arising from variations in the number and types of tasks executed by the clients. Therefore, we consider aggregating the shared layers and task layers separately. Although the shared layers from all clients have the same structure, identical weights $p_{i,j}$ should not be assigned for aggregation in Equation (3.2), as the shared layers are learned differently by clients executing different sets of tasks. Regarding the task layers, the i-th client's task layers, denoted as $\mathbf{W}_i^{T_k}$ for task T_k, can be improved by learning from the task layers $\mathbf{W}_j^{T_k}$, $j \in \mathcal{N}(C)$, where $\mathcal{N}(C)$ is the subset of clients executing T_k. Based on the observations presented in Subsection 3.2.1, we propose employing the FedMTL algorithm to aggregate the local models, thereby enhancing both the task layers and the shared layers to create a personalized MTL model for each client. In each round, FedMTL first computes the similarity score for each pair of clients by analyzing the task layers of each client's MTL model, and then uses the similarity scores to compute the aggregation weights $p_{i,j}$ and $q_{i,j,t}$ to aggregate the shared and the task layers as in Equations (3.2) and (3.3), respectively.

Computation of similarity scores. To compute the similarity score for a pair of clients (C_i, C_j), FedMTL compares the parameters of the task layers and identifies the one-to-one mapping between the task layers of the two clients that maximizes the overall score. The overall similarity score also captures the similarity in the shared layers; if the task layers are similar, there is a high likelihood that the shared layers are as well. FedMTL analyzes the parameters of the task layers instead of relying on matching task IDs, as data from different domains can be used for the same task, and thus, does not guarantee similarities in the model parameters.

For a pair of clients, FedMTL first computes the cosine similarities of all pairs of tasks between these clients, denoted as $A_{i,j} \in \mathbb{R}^{K_i \times K_j}$, using Equation (3.4). Then,

FedMTL uses a standard Hungarian algorithm [190], H_{Score}, to find the one-to-one task mapping with maximum cumulative score $H_{i,j}$ (Equation (3.5)). The similarity score for client C_i with respect to clients C_j, denoted as $S_{i,j} \in [0, 1]$, is computed by dividing the score $H_{i,j}$ by the number of tasks K_i, as shown in Equation (3.6). Here, $S_{i,j}$ might not necessarily be the same as $S_{j,i}$, since the number of tasks executed by the clients can differ.

$$\begin{aligned} A_{i,j}[m][n] &= \cos(\mathbf{W}_i^T[m], \mathbf{W}_j^T[n]); \\ m &\in \{1, \dots K_i\}, n \in \{1, \dots K_j\} \end{aligned} \tag{3.4}$$

$$H_{i,j} = H_{Score}(A_{i,j}) \tag{3.5}$$

$$S_{i,j} = H_{i,j}/K_i \tag{3.6}$$

Aggregation of the MTL models. To obtain the personalized MTL model for a client C_i, FedMTL assigns varying weights to all clients' MTL models based on the similarity score, as calculated in Equation (3.6). To mitigate adverse effects arising from highly dissimilar models belonging to another client C_j, FedMTL sets $S_{i,j} = 0$ if the score $S_{i,j}$ falls below a threshold value Z_i, effectively excluding models that differ significantly from aggregation. The threshold value Z_i is adaptable, $0 \leq Z_{min} \leq Z_i \leq Z_{max} \leq 1$, allowing clients to select their threshold based on the current model's ($\mathbf{W}_i$) performance. In our experiments, initially, the threshold value is set low, $Z_i = Z_{min}$, to encourage learning from a diverse set of clients. As a client participates in multiple rounds and achieves improved model performance, Z_i is programmatically increased in each round by $(1 - Z_{min})/R$, where R is the total number of rounds. This adjustment facilitates learning from more similar models, thereby reducing the need for substantial changes in the parameters, and can be continued until the model's

accuracy converges on the test dataset. The updated score is expressed as in Equation (3.7).

$$S_{i,j} = \begin{cases} 0 & \text{if } S_{i,j} < Z_i \\ S_{i,j} & \text{otherwise} \end{cases} \tag{3.7}$$

For C_i, FedMTL computes the aggregation weights $p_{i,j}$, using Equation (3.8), and then uses $p_{i,j}$ in Equation (3.2) to compute $\mathbf{W}_i^{S^*}$ to aggregate the shared layers from all clients $C_j, j \in \{1, \ldots, N\}$. Here, D_j denotes the size of the training dataset of client C_j, and $S_{i,j}$ represents the pairwise similarity scores of the client models, which are used to compute the aggregation weights. Equation (3.8) is equivalent to the aggregation weights for FedAvg, if $S_{i,j} = 1$.

$$p_{i,j} = S_{i,j} \times D_j \tag{3.8}$$

Similarly, FedMTL computes the weights $q_{i,j,k}$ using $S_{i,j}$ as in Equation (3.9) and uses $q_{i,j,k}$ in Equation (3.3) to obtain the parameters of the task layers $\mathbf{W}_i^{T_k}$ for task T_k of C_i, by aggregating the parameters of the corresponding task $\mathbf{W}_j^{T_k}$ from C_j.

$$q_{i,j,k} = \begin{cases} S_{i,j} \times D_j & \text{if } T_k \in \mathbf{U}_j \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

FedMTL uses the same aggregation weights for all parameters in a model, such that the scaled model captures a similar parameter distribution as the local model. Following this approach, the personalized model of a client is the linear combination of all the models of users executing a similar subset of tasks.

3.3 Privacy-Preserving FedMTL

Improved performance through personalization in FedMTL comes with a trade-off in privacy, as the aggregator needs access to information about the tasks executed by each client and their local model parameters. To preserve the privacy of client data in FL, we can use an established cryptographic technique $\mathcal{E}$, such as SMPC or HE, for secure aggregation. We assume that fundamental operations (e.g., addition, multiplication, comparison, etc.) are supported in $\mathcal{E}$. However, it is required to design secure protocols which the clients invoke to protect the numbers/types of the tasks and the model parameters before uploading them to the server. Furthermore, server-side protocols must be developed to implement the aggregation algorithm within the specified cryptographic domain. In this section, we specifically explore secret sharing [191, 192] in SMPC, as the cryptographic technique to design a secure FedMTL system. The secret-sharing scheme allows sharing of a secret x among P servers, such that the servers can use their shares to compute a function, while each server learns nothing about the secret. Next, we describe the threat model, the overview of the privacy-preserving FedMTL, and the secure protocols to be followed at the client and server sides.

3.3.1 Threat model

We consider an honest-but-curious adversary in a P-party SMPC settings, where each of the P servers honestly follows the protocols, but may individually attempt to learn clients' private data. We assume that the application developer is responsible for setting up the distributed trust, and the parties (client and servers) will communicate using a secure channel.

In our threat model, we consider that at most $P-1$ servers can collude to learn clients' sensitive data and global statistics. A secure version of FedMTL guarantees the following privacy properties:

- Individual model parameters of the clients are not revealed to anyone other than the source client itself.
- IDs of the tasks executed by each client are protected.
- Size of the training dataset used by each client is protected.
- Secure aggregation of encrypted models produces correct personalized models, achieving the same level of accuracy as if the aggregation were done using plain text data.

3.3.2 System overview

Figure 3.3 shows the secure FedMTL system with P aggregators and N clients. The aggregators communicate with each other to compute the required parameters encrypted in secret-shared format, ensuring privacy against the threat model. In additive (or arithmetic) secret sharing (A-SS) [191, 192], P values $(x_1, \ldots, x_P)$ are chosen uniformly at random, subject to the requirement that $\sum_{i=1}^{P} x_i = x \pmod{L}$, where $L = 2^l$ represents l-bit integers. This can be done by choosing $x_1, \ldots, x_{P-1} \in \mathbb{Z}_L$ uniformly at random, and then setting $x_P = x - \sum_{i=1}^{P-1} s_i \pmod{L}$. The reconstruction algorithm simply adds all the shares modulo L, that is, $x = (\sum_{p \in P}[x]_p) \pmod{L}$. We denote the sharing of x across the parties $p \in P$ by $\langle\langle x \rangle\rangle = \left\{ \langle\langle x \rangle\rangle_p \right\}_{p \in P}$, where $\langle\langle x \rangle\rangle_p$ indicates p 's share of x. Fundamental operations are already supported in A-SS; certain functions, such as addition, can be performed locally, while others, such as multiplication and comparison, require communication between servers.

To preserve the data privacy in FedMTL, each client C_i uploads the parameters of the shared layers, task layers, and size of the dataset in encrypted format to the aggregator servers as $\langle\langle \mathbf{W}_i^S \rangle\rangle$, $\langle\langle \mathbf{W}_i^T \rangle\rangle$ and $\langle\langle D_i \rangle\rangle$ respectively. The servers invoke secure protocols to compute $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle$ and $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle$ which are downloaded and decrypted by C_i to get the personalized model $\mathbf{W}_i^* = \{\mathbf{W}_i^{S^*}, \mathbf{W}_i^{T^*}\}$ in plain text.

Since each of the P aggregator servers can monitor the computation's control flow, it is essential to use oblivious operations that avoid data-dependent control flow.

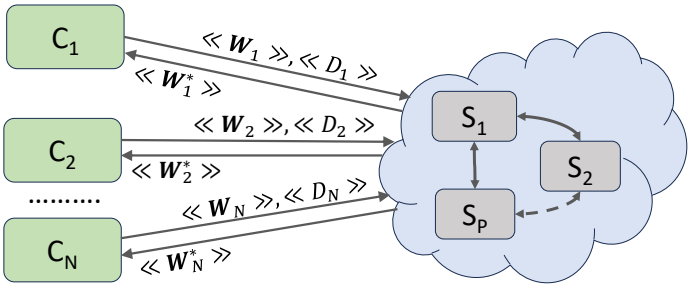


Figure 3.3 Secure FedMTL system using SMPC.

This ensures the security of the input, output, and intermediate results throughout the aggregation process. The secure aggregation protocols in FedMTL provides the security guarantees described in Subsection 3.3.1 to ensure that honest clients no longer have to put their complete trust in all servers for privacy. As long as one server is functioning correctly, privacy is guaranteed.

3.3.3 Secure aggregation protocols

FedMTL employs secure protocols designed to ensure the privacy of client data during the FL workflow. Next, we present the workflow (Algorithm 1) and the protocols for the clients and the servers.

Model initialization. To participate in the privacy-preserving FedMTL workflow, a client requests the aggregator to get the list of supported tasks $\mathbf{U}$, and the complete MTL model architecture with the shared layers $\mathbf{W}^S$ and task layers $\mathbf{W}^{T_k}$, $1 \leq k \leq K = |\mathbf{U}|$, initialized with $\mathbf{W}^{S^{(0)}}$ and $\mathbf{W}^{T_k{}^{(0)}}$. The maximum number of parameters for the task layers is $dt_m = \max_{T_k \in \mathbf{U}} |\mathbf{W}^{T_k}|$. Since these parameters are not confidential, a client can connect to any aggregator to receive this data in plain text.

The client C_i selects a set of tasks $\mathbf{U}_i \subseteq \mathbf{U}$, and create the local MTL model $\mathbf{W}_i^{(0)} = \{\mathbf{W}_i^{S^{(0)}}, \mathbf{W}_i^{T^{(0)}}\}$, where $\mathbf{W}_i^{T^{(0)}}$ contains the initial parameters $\mathbf{W}_i^{T_k{}^{(0)}}$ for $T_k \in \mathbf{U}_i$.

Client side data preparation. At each round, the aggregator randomly selects a subset of clients $\mathbf{C}$, $N = |\mathbf{C}|$, as in traditional FL. Then, it invokes the protocol $\mathcal{F}_{SU}$ (defined below) for each client C_i, initiating the client to train the MTL model using its training dataset of size D_i. After training the model for a predefined number of epochs as in traditional FL, C_i gets the model $\mathbf{W}_i = \{\mathbf{W}_i^{S^{(r)}}, \mathbf{W}_i^{T^{(r)}}\}$ and sets the threshold Z_i in it's r-th communication round. Then, C_i follows the steps below to upload its private data for secure aggregation.

- $\mathcal{F}_{SU}$ encrypts the parameters of the shared layers as $\langle\langle \mathbf{W}_i^S \rangle\rangle$, thereby securing all the values in the vector $\mathbf{W}_i^S$.
- As $\mathbf{W}_i^{T_k}$ may vary in size for different tasks $T_k \in \mathbf{U}_i$, $\mathcal{F}_{SU}$ ensures uniform size dt_m by padding zeros at the end, thereby protecting the type of the task.
- To protect the number of tasks, $\mathcal{F}_{SU}$ generates fake task-specific vectors, each of size dt_m with zero value, and appends those vectors to $\mathbf{W}_i^T$ to generate $\mathbf{W}_i^{T'} \in \mathbb{R}^{K_i' \times dt_m}$, where $K_i' = |\mathbf{W}_i^{T'}|$, $K_i \leq K_i' \leq K$.
- $\mathcal{F}_{SU}$ generates a mapping $\mathbf{M}_i \in \{0,1\}^{K_i' \times K}$ that maps the index of each task-specific parameter $\mathbf{W}_i^{T'}$ to the ID of that task. Thus, $\mathbf{M}_i[j][k] = 1$, if $\mathbf{U}_i[j] = T_k$, otherwise 0.
- $\mathcal{F}_{SU}$ uploads $\langle\langle \mathbf{W}_i^S \rangle\rangle$, $\langle\langle \mathbf{W}_i^{T'} \rangle\rangle$, $\langle\langle \mathbf{M}_i \rangle\rangle$, $\langle\langle D_i \rangle\rangle$, $\langle\langle Z_i \rangle\rangle$.

In this way, $\mathcal{F}_{SU}$ protects the dataset size, the actual number and type of the executed tasks, and the associated model parameters for both the shared and task layers.

Server side aggregation. At each round, FedMTL computes the task-specific parameter from C_i for all T tasks, $\mathbf{W}_i^{T''} \in \mathbb{R}^{K \times dt_m}$ by performing matrix multiplication of $\mathbf{M}_i^\intercal$ and $\mathbf{W}_i^{T'}$, where $\mathbf{M}_i^\intercal$ is the transpose of matrix $\mathbf{M}_i$. Using matrix multiplication in the secret-sharing scheme, the aggregator servers get the share of $\langle\langle \mathbf{W}_i^{T''} \rangle\rangle$ for all clients (Line 5). Then, for each pair of clients (i,j), $1 \leq i, j \leq N$, it uses the secret-sharing version of the Hungarian algorithm $\mathcal{F}_H$ and computes $\langle\langle \mathbf{S}_{i,j} \rangle\rangle$ following Equations (3.5), (3.6), (3.7). Next, the aggregator servers invoke: (i) $\mathcal{F}_S$, to securely compute Equations (3.8) and (3.2) to get the parameters of the shared

Algorithm 1 Secure FedMTL Aggregation

Input: Communication Round R, number of tasks K, initial model parameters $\mathbf{W}^{(0)} = \{\mathbf{W}^{S^{(0)}}, \mathbf{W}^{T_k{}^{(0)}}\}$, $1 \leq k \leq K$

Output: MTL models $\mathbf{W}_i^*$ for C_i

1: **for** r = 1, ..., R **do**

2: Randomly selects a subset of clients $\mathbf{C}$, $N = |\mathbf{C}|$

3: **for** client $C_i \in \mathbf{C}$ **do**

4: $\langle\langle \mathbf{W}_i^S \rangle\rangle, \langle\langle \mathbf{W}_i^{T'} \rangle\rangle, \langle\langle \mathbf{M}_i \rangle\rangle, \langle\langle D_i \rangle\rangle, \langle\langle Z_i \rangle\rangle = C_i.\mathcal{F}_{SU}()$

5: $\langle\langle \mathbf{W}_i^{T''} \rangle\rangle = \langle\langle \mathbf{M}_i \rangle\rangle^\intercal \times \langle\langle \mathbf{W}_i^{T'} \rangle\rangle$

6: **end for**

7: $\langle\langle \mathbf{S} \rangle\rangle = \mathcal{F}_H(\langle\langle \mathbf{W}^{T''} \rangle\rangle)$, where $\langle\langle \mathbf{W}^{T''}[i] \rangle\rangle = \langle\langle \mathbf{W}_i^{T''} \rangle\rangle$

8: **for** client $C_i \in \mathbf{C}$ **do**

9: $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle = \mathcal{F}_S(\langle\langle \mathbf{W}^S \rangle\rangle, \langle\langle \mathbf{S} \rangle\rangle, \langle\langle \mathbf{D} \rangle\rangle)$

10: $\langle\langle \mathbf{W}_i^{T^{**}} \rangle\rangle = \mathcal{F}_T(\langle\langle \mathbf{W}^{T''} \rangle\rangle, \langle\langle \mathbf{S} \rangle\rangle, \langle\langle \mathbf{D} \rangle\rangle)$

11: $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle = \langle\langle \mathbf{M}_i \rangle\rangle \times \langle\langle \mathbf{W}_i^{T^{**}} \rangle\rangle$

12: $\mathbf{W}_i^* = C_i.\mathcal{F}_{SR}(\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle, \langle\langle \mathbf{W}_i^{T^*} \rangle\rangle)$

13: **end for**

14: **end for**

layers $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle$ (Line 9), (ii) $\mathcal{F}_T$, to securely compute Equations (3.9) and (3.3) to get the parameters of the task layers $\langle\langle \mathbf{W}_i^{T^{**}} \rangle\rangle$ and mapped into the parameters for the required tasks $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle$ for C_i using the matrix $\langle\langle M_i \rangle\rangle$ (Lines 10-11). $\mathcal{F}_H$, $\mathcal{F}_S$ and $\mathcal{F}_T$ are computed securely using standard techniques for addition, multiplication and comparison operations in A-SS domain.

Retrieving personalized models. Client C_i invokes $\mathcal{F}_{SR}$ (Line 12) to download $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle$ and $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle$ from the aggregator and decrypts them to obtain the updated model $\mathbf{W}_i^* = \{\mathbf{W}_i^{S^*}, \mathbf{W}_i^{T^*}\}$. Subsequently, the client can proceed with training the model for the upcoming round.

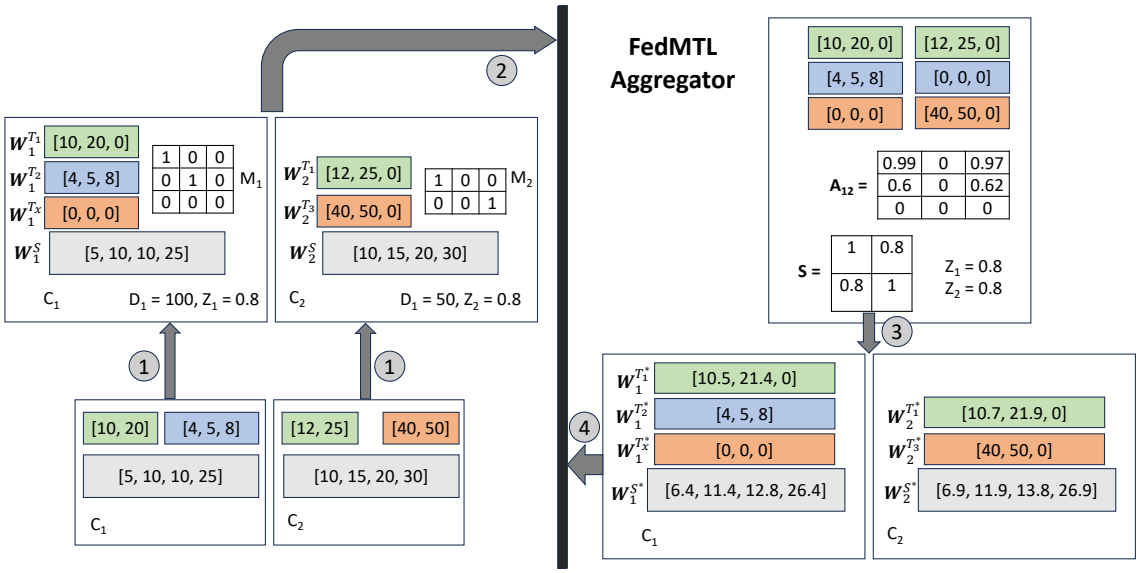


Figure 3.4 Example of privacy-preserving FedMTL aggregation of the MTL models from two clients C_1 and C_2.

3.3.4 Secure FedMTL example

Figure 3.4 illustrates the privacy-preserving FedMTL aggregation for 2 clients in a single round. Client C_1 trains the model for tasks T_1 and T_2, while client C_2 trains the model for tasks T_1 and T_3. Subsequently, for each client C_i, FedMTL invokes the $\mathcal{F}_{SU}$ protocol to prepare the data for upload, which includes the parameters of the shared layers $\mathbf{W}_i^S$, parameters of the task layers $\mathbf{W}_i^{T_k}$, task-mapping $\mathbf{M}_i$, dataset size D_i, and threshold value Z_i, as shown in step ① (Figure 3.4). For each task layer, clients pad the parameter vector with zero values such that all task layers have the same number of parameters, thereby protecting the types of the tasks. Additionally, a client can add one or more fake tasks to protect the actual number of tasks for which the MTL model is trained. For instance, C_1 adds a fake task $\mathbf{W}_1^{T_x}$ in. Figure 3.4. In step ②, $\mathcal{F}_{SU}$ encrypts all the values and uploads the encrypted data to the aggregator servers.

The FedMTL aggregator invokes $\mathcal{F}_H$ to calculate the similarities $\mathbf{A}_{i,j}$ between the task layers of C_i and C_j. It then employs the Hungarian assignment algorithm and uses the threshold value Z_i to compute the similarity matrix $\mathbf{S}$. In step ③, the similarity matrix $\mathbf{S}$ is employed by protocols $\mathcal{F}_S$ and $\mathcal{F}_T$ to calculate the aggregation

weights $p_{i,j}$ and $q_{i,j,t}$ for aggregating the shared and task layers of the clients' models, respectively. Finally, in step ④, the aggregated model parameters are downloaded by the clients.

3.4 System Overhead and Security Analysis

This section presents first the overhead analysis of FedMTL without privacy protection, and then the overhead and security analysis of the privacy-preserving version of FedMTL.

3.4.1 Overhead analysis of FedMTL without privacy

The FedMTL aggregation incurs similar computation and communication overhead as state-of-the-art approaches that analyze model parameters at the server to provide personalized models. We consider N number of clients participating in training, with the model parameters denoted as $d = ds + K \times dt_m$, where ds is the number of parameters in the shared layer, dt_m is the maximum number of parameters in the task-specific layer and K is the number of tasks. In FedMTL, there is no additional computational overhead at the client side other than the training of the MTL models. For aggregation, the computational complexity at the server side is $O(N^2(d + K^2 \times dt_m + K^3))$. Typically, $d \gg dt_m$ and when T is small, the complexity becomes $O(N^2 d)$, which is similar to FedAMP [193]. For FedFomo [107] and MOCHA [23], the complexity at the server is $O(Nd)$; however, both approaches offload additional computation onto clients. Other state-of-the-art approaches, such as FedAvg [22], FedProx [93], and pFedMe [194], have a lower computational cost of $O(Nd)$, but they do not perform well in the case of task-heterogeneity.

In FedMTL, each client uploads the model parameters, dataset size, task-mapping, and threshold value to the aggregator. The dataset size, task-mapping, and threshold value are very small and can be represented in a few bytes. Therefore, the amount of data (including upload and download) that each client needs to

transmit per communication round in FedMTL is approximately $2 \cdot \Delta$, where Δ represents the size of the model. FedMTL's overhead is lower compared to state-of-the-art approaches, such as FedFomo, MOCHA, FedAMP, FedDWA, etc. The overhead of FedMTL is the same as that of FedAvg, FedProx, pFedMe. However, FedMTL outperforms them in terms of accuracy while demonstrating the capability of aggregation even when clients train MTL models on different sets of tasks.

3.4.2 Overhead of secure FedMTL

The overhead associated with the secure aggregation of FedMTL depends on the cryptographic technique used to protect the privacy of the data. We present the overhead analysis in the case of using SMPC for privacy-preserving FedMTL as discussed in Section 3.3.

In secure FedMTL, each client C_i incurs negligible computational overhead in preparing the data for uploading, as it only involves preparing the task-map of size $K_i' \times K$ and updating the K_i' number of vectors of size dt_m that contain the parameters of the task-layers. At the server side, the computational complexity is $O(N^2 \times (d + K^2 \times dt_m + K^3))$, which is the same as the plain text in asymptotic notation. However, in a secret-shared domain that utilizes Beaver triples [128] to evaluate multiplication operations, one multiplication in plain text is equivalent to four multiplications and three additions in A-SS format.

In secure FedMTL using SMPC, each client needs to transmit approximately $2 \times P \times \Delta$ of data (including upload and download) per communication round. Here, Δ represents the size of the model, and P is the number of aggregator servers. The same amount of data transmission is required for other state-of-the-art approaches, such as FedAvg, when using SMPC for secure aggregation.

To execute the secure protocols in FedMTL, aggregator servers need to communicate with each other. For example, when comparing two B-bit numbers,

the standard implementation requires $log(B)$ rounds of communication. FedMTL protocols, such as, $\mathcal{F}_S$ and $\mathcal{F}_T$, involve only multiplication operations, and can be computed in a single round. The computation of similarity scores using the $\mathcal{F}_H$ protocol involves calculating the score for each pair of clients; thus, the simple implementation requires a large number of communication rounds. However, by processing data in batches, the number of communication rounds can be reduced.

3.4.3 Security analysis of secure FedMTL

FedMTL employs the standard additive secret-shared (A-SS) approach to upload the clients' private data to the aggregator servers. Thus, data at rest (model parameters, dataset size, task-mapping threshold value) is information-theoretically secure [195] against the threat model following Axiom 1. For aggregation, FedMTL uses existing A-SS protocols, which are secure following Axiom 2. After the axioms, we present two theorems to show that Secure FedMTL preserves the privacy of client data throughout the FL workflow.

Axiom 1. A value x is information-theoretically secure in additive secret-shared format even if $P-1$ out of P parties collude.

Axiom 2. There exist secure protocols for fundamental operations (arithmetic operations, comparisons, sorting) that preserve the privacy of the input and output of each operation.

Theorem 1. $\mathcal{F}_{SU}$ secures each client's uploaded data to the aggregator servers, protecting the parameters of the trained MTL model and the number and types of executed tasks from attacks described in the threat model.

Proof. In FedMTL, each value in the vectors representing the parameters of the shared layers and task-specific layers is encrypted in A-SS format. Therefore, the parameters of the trained MTL model are secure, following Axiom 1. Since the size of the vectors for each task-layer is the same, the adversary cannot identify the type

of the task by observing the volume of data. Additionally, since each client can add fake task-layers, the adversary cannot learn the actual number of executed tasks by observing the number of task-layers. Thus, the parameters, as well as the number and types of tasks, are protected against the threat model.

Theorem 2. The private data of each client is protected against the threat model throughout the execution of the secure protocols ($\mathcal{F}_H$, $\mathcal{F}_S$, $\mathcal{F}_T$) for FedMTL aggregation.

Proof. In FedMTL aggregation, $\mathcal{F}_H$ computes the similarity score for each pair of clients, involving addition, element-wise multiplication, matrix-multiplication, and comparison operations in the secret-shared domain. Since the similarity score is calculated over all possible pairs of tasks, one from each client, it does not leak the number or types of tasks. Finally, $\mathcal{F}_S$ and $\mathcal{F}_T$ compute the aggregated parameters using the matrix-multiplication operation. Since these operations are secure in the secret-shared domain following Axiom 2, the clients' private data is protected against the threat model throughout the FedMTL workflow.

3.5 Evaluation

The evaluation has several goals: demonstrate the feasibility of FedMTL when clients execute different numbers and types of tasks, compare the performance of FedMTL with state-of-the-art approaches, conduct ablation studies to assess the performance of the proposed algorithm compared to alternative approaches, and demonstrate the performance of the privacy-preserving version of FedMTL in terms of accuracy and overhead.

3.5.1 Experimental setup

Dataset description, data and task distribution. We evaluate FedMTL on two face attribute datasets (CelebA and LFWA) [187] and one indoor scene dataset, NYUD2 [196], which are commonly used to evaluate MTL models. For

Table 3.1 Dataset Parameters

Dataset	D	N	K	L
CelebA	12,000	60	8	5
LFWA	9,000	60	8	5
FaceA	16,000	80	8	5
NYUD2	1,449	20	4	3

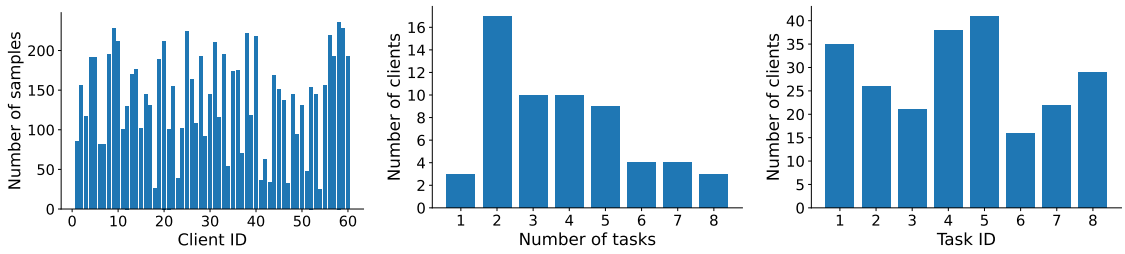
CelebA/LFWA, we consider $K = 8$ tasks, each involving the classification of $L = 5$ face attributes. For NYUD2, we consider $K = 4$ tasks, each involving $L = 3$ categories for pixel-wise semantic segmentation. Additionally, to assess FedMTL's effectiveness in scenarios with both data and task heterogeneity, we merge CelebA and LFWA to form a unified face attribute dataset, referred to as FaceA.

For each dataset, we select a number of samples D and distribute the samples among N clients using a symmetric Dirichlet distribution. We set $N = 60, 80$ and 20 for CelebA/LFWA, FaceA, and NYUD respectively. Each client C_i splits its dataset into training, validation, and test sets with a distribution of 70%, 15%, and 15%, respectively. In FedMTL, C_i can select any number of tasks to participate in the FL workflow. Then, to choose tasks from the set $\mathbf{U}$, where $K = |\mathbf{U}|$, within the dataset, each client C_i (where $i \in 1, \ldots, N$) selects a random number of tasks K_i such that $1 \leq K_i \leq K$. Subsequently, the client randomly picks K_i tasks from the task-set $\mathbf{U}$. The values for N, D, and K for the datasets are shown in Table 3.1.

We consider two different task-distributions: (a) D_1: where each client selects $K_i = 2$ tasks, (b) D_2: each client selects K_i tasks where $1 \leq K_i \leq K$.

To provide an illustration for one dataset, Figure 3.5 shows the number of samples per client and task distributions for the CelebA dataset using a random seed.

Implementation details. For the CelebA/LFWA/FaceA datasets, we use LeNet [188] and MobileNetV2 [197] model architectures. For NYUD2 we use SegNet model architecture [198]. For client C_i, we replace the last layer of each model with



(a) (b) (c)

Figure 3.5 Sample distribution of tasks among $N = 60$ clients: (a) number of samples per client, (b) distribution of clients across different numbers of tasks, (c) distribution of clients across different tasks.

Table 3.2 Number of Parameters in MTL Models

Model Architecture	#Parameters in the Shared Layers	#Parameters in a Task Layer
LeNet	1,628,210	2,505
MobileNetV2	2,223,872	6,405
SegNet	24,943,296	37,123

K_i task layers, where each task layer is a linear layer with L number of neurons for CelebA/LFWA/FaceA and a sequence of convolution layers with L output channels for NYUD2. Table 3.2 shows the number of shared parameters and task-layer parameters in each model.

We evaluate the performance of FedMTL by comparing it with state-of-the-art approaches, implemented using the open source library PFLlib[1] and use the default hyper-parameters. To implement secure FedMTL aggregation, we employ CrypTen [11], which provides APIs for creating arithmetic shares of private data and supports $\mathcal{P}$-party SMPC computations. We use 64 bits ($B = 64$) to represent values in A-SS format. The experiments are conducted on a 3.4GHz Intel Core i7, with parties running in separate processes. We implemented the Secure FedMTL prototype to demonstrate the feasibility of the proposed system using CrypTen's basic APIs, without focusing on latency optimizations. It is possible to reduce both the latency and the overall overhead through pre-computation and parallel data processing.

[1]https://github.com/TsingZ0/PFLlib

Comparison methods. We compare FedMTL with the following approaches: (i) FedAvg [22], which is the vanilla FL technique, (ii) FedProx [93], which employs a proximal term to formulate the clients' optimization objectives to mitigate the adverse influence of heterogeneity on FL, (iii) MOCHA [23], which considers each client as a separate task and applies MTL with model similarity penalization, (iv) pFedMe [194], which uses a regularized loss function to optimize the personalized model w.r.t. each client's local data distribution, (v) FedFomo [107], which computes personalized aggregation weights via minimizing the validation loss on each client based on the model information collected from other clients, and (vi) FedAMP [193], which employs federated attentive message passing to facilitate collaboration among similar clients.

Training and aggregation settings. For the experiments, we assume 100% client participation, although FedMTL would still work in the case of partial participation. The number of local training epochs is set to 1, and the number of global communication rounds is set to 20. We employ mini-batch SGD as the local optimizer in all approaches. The batch size for each client is set to 32 for CelebA/LFWA/FaceA, and 2 for NYUD2. For aggregation, we vary the threshold Z from 0.75 to 0.95 linearly. We conduct tests for all methods over three runs and report the average results.

Evaluation metrics. For CelebA/LFWA/FaceA, we report the average test accuracy of the personalized MTL models across all participant devices. For NYUD2, we report mIoU and pixel accuracy for the semantic segmentation task.

3.5.2 Results and analysis

Comparison with baselines. We compare FedMTL with state-of-the-art approaches when clients are executing the same number of tasks (using task-distribution D_1). The model architecture remains the same across clients, which is required for the

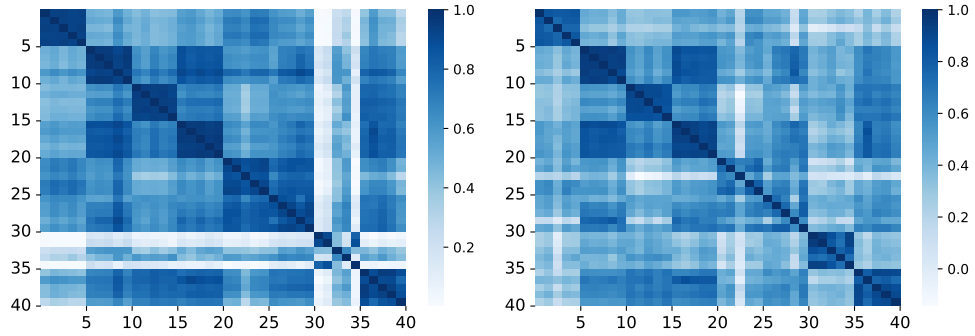
Table 3.3 Comparison of FedMTL with Baselines

Dataset	**CelebA**	**LFWA**	**FaceA**	**NYUD2**	
	Acc (%)	Acc (%)	Acc (%)	mIoU (%)	PixAcc (%)
FedMTL	**83.3**	**70.9**	**78.3**	**37.2**	**74.4**
FedAvg	76.0	64.7	68.5	28.9	62.0
FedProx	76.4	64.7	69.3	30.4	63.9
MOCHA	81.7	67.9	76.9	34.9	71.1
pFedMe	76.2	64.7	69.2	30.3	64.2
FedFomo	80.9	68.9	76.1	35.1	72.0
FedAMP	81.9	68.4	77.1	35.4	71.3

baseline approaches. The results presented in Table 3.3 show that FedMTL achieves better performance compared to other methods for all datasets. FedAvg, FedProx, and pFedMe achieve low accuracy because they aggregate local models without considering task heterogeneity, and the accuracy is impacted by the aggregation of model parameters from conflicting tasks. FedFomo and FedAMP manage to tackle this adverse effect by applying regularization or personalized aggregation weights. In contrast, FedMTL assigns aggregation weights based on similarities in task layers, enabling clients to achieve higher accuracy. FedMTL avoids the complex client-side analysis in FedFomo and FedAMP, reducing computation and communication overhead in resource-constrained devices.

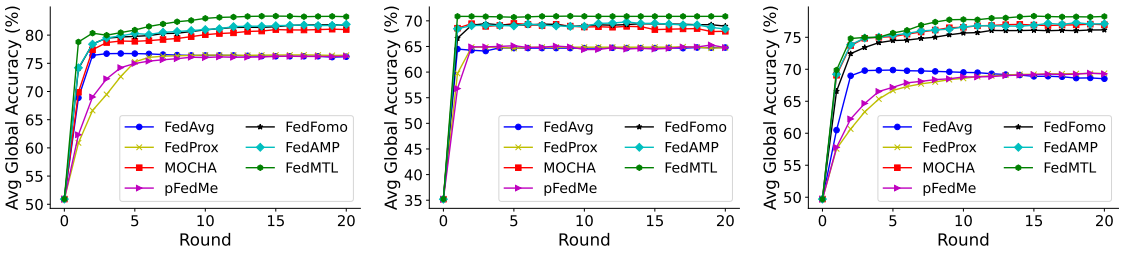
Data and task heterogeneity. We consider $N = 80$ clients, where C_i can use data samples from either CelebA or LFWA within FaceA. The tasks $\mathbf{U}$, $K = |\mathbf{U}| = 8$, are distributed among the N clients according to D_1. The results for the FaceA dataset, presented in Table 3.3, indicate that FedMTL outperforms other approaches by effectively handling both data and task heterogeneity.

To analyze in more detail, we create two groups, where the group G_1 and G_2 have data from CelebA and LFWA respectively. There are 20 clients in each group. Each group is divided into four sub-groups where the clients in each sub-group G_{ig}, $i \in \{1, 2\}$, $g \in \{1, 2, 3, 4\}$ train MTL model locally for two tasks: $[T_a, T_b]$, where



(a) (b)

Figure 3.6 Similarity scores: (a) task-layers (b) shared-layers.



(a) (b) (c)

Figure 3.7 Test accuracy during training for three datasets: (a) CelebA, (b) LFWA, (c) FaceA.

$a = 2g - 1, b = 2g, 1 \leq a, b \leq 8$. As shown in Figure 3.6(a), the parameters of the task layers of the clients in the same group G_{ig} are very similar. Although G_{1g} and G_{2g} work on the same tasks, the task layers are not very similar since the datasets are different. Since FedMTL assigns weights based on the similarities of task-specific parameters, it enables the clients to aggregate models considering task and domain heterogeneity, thus performing better compared to other approaches.

Efficiency. To evaluate the efficiency of FedMTL aggregation algorithm, we record the evolution of average test accuracy over global communication rounds for CelebA, LFWA, and FaceA datasets, with tasks distributed according to the D_1 distribution. As illustrated in Figure 3.7, FedMTL achieves higher accuracy than other state-of-the-art approaches and converges within a few rounds.

Different numbers of tasks. We evaluate the effectiveness of FedMTL in the case of task heterogeneity, where each client works on a different number of tasks (D_2 task-distribution). The task distributions are presented in Figure 3.5. We do not compare the results with existing techniques, as they cannot support this scenario directly. FedMTL works well in addressing task heterogeneity and achieves 82.5% and 70.5% accuracy for CelebA and LFWA respectively, and 36.7% mIoU for NYUD2.

Ablation studies. We conduct ablation studies to compare different versions of FedMTL in scenarios where clients execute varying sets of tasks. Specifically, we consider the following versions:

- FedMTL-X: To aggregate the shared layers from all clients, it computes the aggregation weights $p_{i,j}$ for C_i by setting $S_{i,j} = 1$ in Equation (3.8), and ignores the task layers from other clients by setting $q_{i,j,t} = 1$ if $j = i$, otherwise $q_{i,j,t} = 0$.
- FedMTL-G: To aggregate the shared layers and task layers of task T_t for C_i, it computes the aggregation weights by setting $S_{i,j} = 1$ in Equations (3.8) and (3.9).
- FedMTL-T: To aggregate the shared layers from all clients, it computes $p_{i,j}$ for C_i by setting $S_{i,j} = 1$ in Equation (3.8), and aggregates the task-layers by computing the similarity scores following Equation (3.4) to Equation (3.7) and setting $q_{i,j,t}$ values using Equation (3.9) as in FedMTL.
- FedMTL-J: It computes the similarity between two client's models based on the IDs of tasks, instead of task-specific parameters. The similarity score for a pair of clients (C_i, C_j) is computed by $J_{i,j} = \frac{|\mathbf{U}_i \cap \mathbf{U}_j|}{|\mathbf{U}_i \cup \mathbf{U}_j|}$, $J_{i,j} \in [0, 1]$, where $\mathbf{U}_i$ and $\mathbf{U}_j$ are the set of tasks executed by C_i and C_j respectively. It computes the aggregation weights using Equations (3.8) and (3.9), where $S_{i,j} = J_{i,j}$.
- FedMTL-E: For C_i, it only considers the clients executing the same set of tasks. The similarity score for a pair of clients (C_i, C_j) is computed based on the types of tasks as $E_{i,j} = (\mathbf{U}_i == \mathbf{U}_j)$, $E_{i,j} \in \{0, 1\}$. It computes the aggregation weights using Equations (3.8) and (3.9), where $S_{i,j} = E_{i,j}$.
- FedMTL-Z: It applies our proposed algorithm to aggregate the models based on the similarity scores from the task-specific parameters and uses a fixed threshold value Z.

Comparison with alternative approaches. Figure 3.8 shows the average accuracy per round for FaceA with different task distributions. It illustrates that our approach,

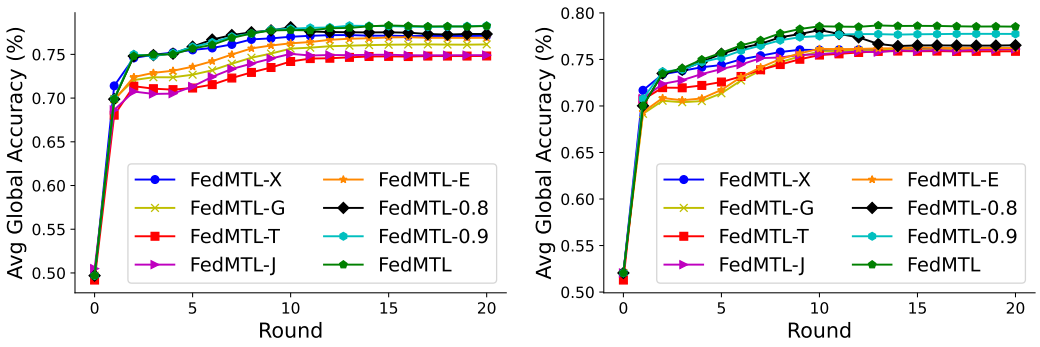


Figure 3.8 Average global accuracy for FaceA dataset with (a) D_1 task distribution (b) D_2 task distribution.

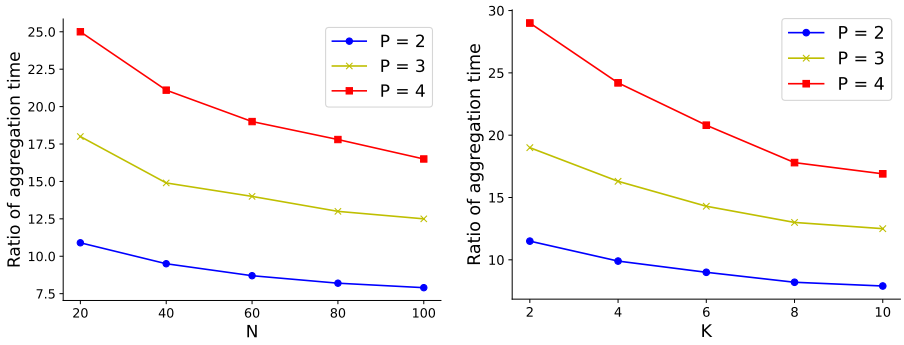
FedMTL, utilizing the similarities of task layers, achieves the highest accuracy, 78.3% and 78.5% for D_1 and D_2 task distribution, respectively. FedMTL outperforms FedMTL-X and FedMTL-G since these approaches fail to learn the task layers from the participating clients effectively. FedMTL-T aggregates the task-layers based on the similarity score computed using parameters of the task-layers, and aggregates the shared layers as in conventional FedAvg. However, this approach fails to perform well because shared layers are generic for all clients and may be adversely affected by client models trained on dissimilar sets of tasks. While FedMTL-J and FedMTL-E take into account task heterogeneity, they are adversely affected when clients execute the same task from a different domain. This occurs because the similarity scores are measured based on task ID, potentially assigning identical weights to model parameters from different domains.

Effect of hyper-parameter. We also examine the impact of the threshold value Z used as a hyper-parameter in our similarity score algorithm. Figure 3.8 shows the average global test accuracy for FaceA dataset using FedMTL-Z with $Z = 0.8$ (FedMTL-0.8) and $Z = 0.9$ (FedMTL-0.9). By setting a constant threshold value, FedMTL-0.8 and FedMTL-0.9 either allow learning from many clients, potentially affecting personalization negatively, or limit learning from a few highly similar models.

Figure 3.8 also shows that FedMTL achieves the highest accuracy as it increases the threshold Z in each round.

Secure version of FedMTL. In the privacy-preserving version of FedMTL, we encrypt the model parameters, dataset size, and task mapping before uploading the data to the P-aggregators. We evaluate the accuracy of the secure FedMTL for CelebA, LFWA, FaceA and NYUD2 datasets using D_1 task-distribution. For each dataset, we get the same accuracy performance as the plain text version. As shown in Figure 3.9, the overhead of privacy-preserving FedMTL w.r.t. the plain text version decreases as the number of clients N or number of tasks T increases. This is due to the fact that, while both versions incur increased computation overhead as N or T increases, the number of communication rounds among the computing parties in the secure version of FedMTL remains constant. For the aggregation of MTL models using the LeNet architecture, the amount of data transferred from a client to each aggregator server is around 12.5 MB. However, as the number of aggregator servers P increases, the communication overhead also rises due to increased communication among the computing parties. Given that FedMTL aggregation does not require real-time processing, it remains feasible for real-world scenarios. For example, with $P = 3$, $N = 80$, $K = 8$, the secure aggregation of MTL models using the LeNet architecture takes around 45 seconds. About 71% of the total time is spent on aggregating shared parameters ($\mathcal{F}_S$) because there are significantly more shared parameters compared to parameters in the task layers. However, it is possible to reduce this overhead through pre-computation and parallel data processing.

To evaluate secure FedMTL considering network delay and bandwidth restrictions, we performed an experiment with three AWS instances (t2.micro, US-East-1 region) as the computing parties. For the MTL models using the LeNet architecture, FedMTL takes around 52 seconds to complete the secure aggregation of models from $N = 80$



(a) (b)

Figure 3.9 Ratio of aggregation time between the secure and plain-text versions of FedMTL in P-party setting for varying (a) number of participating clients N and (b) number of tasks K.

clients. This indicates that network latency has minimal impact on the overall aggregation time.

3.6 Chapter Summary

This chapter presents FedMTL, a novel FL aggregation technique that enables clients to collaboratively enhance their personalized multi-task learning (MTL) models. FedMTL determines aggregation weights for each client by examining the parameters of task-specific layers in MTL models and employs a layer-wise aggregation strategy across participating clients. The FedMTL algorithm can seamlessly integrate with existing privacy-preserving techniques to ensure the security of clients' sensitive data during aggregation. The experimental results demonstrated that FedMTL outperforms state-of-the-art FL aggregation approaches and can work in cases where clients are involved in different sets of tasks. Additionally, we implemented a secure version of FedMTL using secret-sharing SMPC, which achieves the same accuracy performance as plain text while preserving the privacy of client data.

CHAPTER 4

CRYPTGNN: SECURE INFERENCE FOR GRAPH NEURAL NETWORKS

This chapter presents the design, implementation and evaluation of a secure inference service for Graph Neural Networks. Our solution uses novel secure multi-party computation (SMPC) techniques, works with any number of SMPC parties, does not require a trusted server, and is provably secure even if P-1 out of P SMPC parties in the cloud collude with each other. CryptGNN consists of two novel protocols to enable privacy-preserving inference of encrypted GNN models on encrypted input graph data in the cloud. **CryptMPL** executes the message-passing layer (MPL), while preserving the privacy of input data (i.e., node features and graph structure). **CryptMUL** executes the secure multiplication operations required for evaluating the linear and nonlinear feature transformation layers (FTLs) in GNN models. CryptMPL and CryptMUL are invoked from the GNN models to guarantee the cloud providers do not learn partial results of functions executed over secure inputs or the final inference results. Our theoretical analysis proves that CryptGNN is correct and secure, as models using CryptGNN achieve the same accuracy as plain-text models while protecting the input graph and the model parameters. Our experiments demonstrate that CryptGNN and its protocols achieve lower latency and overhead than the baseline approaches. In this chapter, Section 4.1 presents the background on GNNs and the cryptographic primitives used in CryptGNN. Section 4.2 introduces the threat model and gives an overview of CryptGNN. Sections 4.3 and 4.4 describe CryptGNN's novel protocols, CryptMPL and CryptMUL, respectively. Section 4.5 presents the analysis of CryptGNN. The implementation and evaluation of CryptGNN are detailed in Section 4.6. The chapter is summarized in Section 4.7.

4.1 Preliminaries

This section covers background information on GNN and Cryptographic Primitives. As a matter of notation: (i) $x \xleftarrow{\$} \mathbb{Z}_L$ denotes that x is uniformly randomly sampled from $\mathbb{Z}_L$, where $L = 2^l$ represents l-bit values; (ii) regular and bold characters represent a scalar and matrix, respectively.

Message-passing layer (MPL) in GNN. This key operation is executed on graph data $\mathcal{G} = (\mathbf{X}, \mathbf{S}, \mathbf{D})$. $\mathbf{X} \in \mathbb{R}^{N \times K}$ represents the node features as a matrix, where N is the number of nodes in the graph and K is the number of features for each node. The structure of a graph is often stored via edges, represented as source/destination indexes $\mathbf{S}$ and $\mathbf{D}$, where $(\mathbf{S}[j], \mathbf{D}[j])$ represents the j-th edge. We consider the most common MPL, where the features of neighboring nodes are aggregated at each node. For the i-th node, the MPL processing is expressed as in Equation (4.1), where $\mathcal{N}(i)$ is the neighbor set of node i, $\mathbf{x'}_i$ is the aggregated feature vector, and $\mathbf{x}_j$ is the current feature vector of node j. A GNN model often consists of multiple MPLs, with FTLs in between.

$$\mathbf{x'}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \tag{4.1}$$

Feature transformation layers (FTLs) in GNN. A GNN model architecture incorporates several FTLs, which can involve linear operations, non-linear activations, and other operations that modify the feature representations of the nodes. Below, we describe some common types of FTLs in GNNs. We use $\times$ and $\otimes$ symbols for scalar and matrix multiplication, respectively.

Linear layers. A linear layer uses learned parameters (weight matrix $\mathbf{H}$ and bias matrix $\mathbf{B}$) to transform intermediate feature matrices during inference. Mathematically, it involves matrix multiplication and addition operations:

$$\mathbf{X'} = \mathbf{X} \otimes \mathbf{H} + \mathbf{B} \tag{4.2}$$

Non-linear layers. A non-linear layer modifies the input representation by employing non-linear functions to each element of the input. For instance, a sigmoid layer applied to the input vector $\mathbf{X}$ computes the sigmoid function for each element in $\mathbf{X}$. Non-linear functions can be implemented using standard approximations [11]. These non-linear layers do not require any trained parameters to transform the values.

Batch normalization layer. During inference, a batch normalization layer utilizes learned parameters, including mean and variance calculated from the training data, along with model-specific parameters (e.g., ϵ, γ, and β), to normalize an input value x to the value y as defined in Equation (4.3).

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta \tag{4.3}$$

Cryptographic primitives. A secret sharing [199] scheme shares a secret x among $\mathcal{P}$ parties, s.t. the parties can collectively reconstruct the secret, while learning nothing about the secret. We use $\mathcal{P}$-out-of-$\mathcal{P}$ secret sharing schemes, which require the shares of all $\mathcal{P}$ parties to reconstruct the data. We denote the parties by CP_i, $i \in \{1, \ldots, \mathcal{P}\}$.

Additive secret sharing (A-SS): In our work, we primarily use A-SS approach. In A-SS, the secret value and its shares are defined over the ring $\mathbb{Z}_L$. A real value $x_R \in \mathbb{R}$ is represented using a fixed-point encoding with a scaling factor B to obtain $x = \lfloor Bx_R \rfloor \in [-2^{l-1}, 2^{l-1})$, where $B = 2^f$ for a given precision of f bits. x can be decoded as $x_R \approx \frac{x}{B}$. We denote the shares of x across the parties by $[\![x]\!] = \{[\![x]\!]_p\}_{p \in \mathcal{P}}$, where $[\![x]\!]_p$ indicates CP_p 's share of x. In A-SS, $\mathcal{P}$ shares are chosen s.t. $\sum_{i=1}^{\mathcal{P}} x_i = x \bmod L$. This can be done by choosing $x_1, \ldots, x_{\mathcal{P}-1} \xleftarrow{\$} \mathbb{Z}_L$, and setting $x_{\mathcal{P}} = (x - \sum_{i=1}^{\mathcal{P}-1} x_i) \bmod L$. The reconstruction algorithm simply adds all the shares as $x = (\sum_{p \in \mathcal{P}} [x]_p) \bmod L$.

Multiplicative secret sharing (M-SS): We define M-SS over real field $\mathbb{R}$, where $\mathcal{P}$ values are chosen uniformly at random, such that $x = \prod_{i=1}^{\mathcal{P}} x_i$, $x_i \in \mathbb{R}$ and $x_i > 0$. We denote the M-SS of x across the parties $p \in \mathcal{P}$ by $\langle\langle x \rangle\rangle = \left\{ \langle\langle x \rangle\rangle_p \right\}_{p \in \mathcal{P}}$, where $\langle\langle x \rangle\rangle_p$ indicates party CP_p 's share of x.

Beaver triples: Given the additive secret shares of values $X, Y \in \mathbb{Z}_L$, computing the shares of $X \times Y$ requires interaction between the parties. A commonly used approach for this secure multiplication is using a Beaver triple [128], which consists of three elements (A, B, C) such that $C \leftarrow A \times B$, and $A, B \xleftarrow{\$} \mathbb{Z}_L$. The additive secret shares of a Beaver triple (A, B, C) can be used to compute the shares of $Z \leftarrow X \times Y$ by following the protocol $\mathcal{F}_{BeaverMul}([\![X]\!], [\![Y]\!], [\![A]\!], [\![B]\!], [\![C]\!])$ shown below:

- Each party gets the share of triples as $([\![A]\!], [\![B]\!], [\![C]\!])$.
- Each party computes $[\![U]\!] \leftarrow ([\![X]\!] - [\![A]\!])$ and $[\![V]\!] \leftarrow ([\![Y]\!] - [\![B]\!])$.
- All parties interact with each other to reveal $U \leftarrow (X - A)$ and $V \leftarrow (Y - B)$.
- Each party computes the shares of Z as $[\![Z]\!] \leftarrow U \times [\![B]\!] + V \times [\![A]\!] + [\![C]\!] + U \times V$.

Matrix multiplication can also be performed using Beaver triples following the above steps, just by replacing $\times$ with $\otimes$ to represent the multiplication of different matrices.

4.2 Threat Model and System Overview

Threat model. In our system, there are three key entities:

- Model owner (MO): Its primary concern is safeguarding the parameters of the trained GNN model, while ensuring accurate results for each inference request.
- Data owners (DO): The system can accommodate multiple DOs (clients), each making numerous inference requests and concerned about ensuring privacy of the input graph data for each request.
- Cloud servers (referred to as parties or CP): We consider an honest-but-curious adversary in the $\mathcal{P}$-party SMPC settings, where each of the $\mathcal{P}$ cloud servers honestly follows the protocols, but may attempt to learn the private data of the MO or DO individually or through collusion.

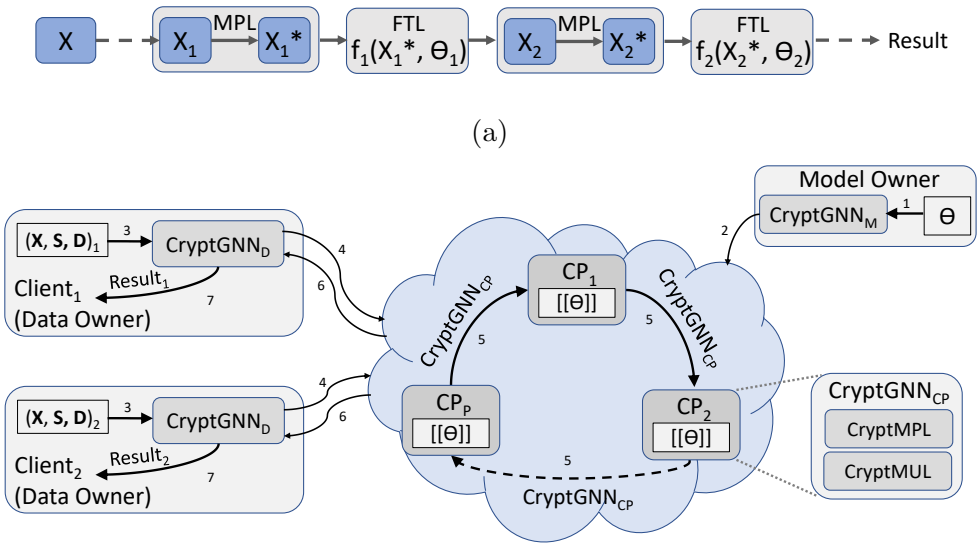


Figure 4.1 (a) GNN computation (b) CryptGNN architecture.

Our threat model TM assumes that at most $\mathcal{P}-1$ parties may collude to learn DO's input data or MO's model parameters. Within TM, we also consider the cases where $\mathcal{P}-1$ colluding parties may collude either with the MO to gain access to the DO's input data or with a DO that they control, DO_{fake}, to access the MO's model parameters or the input graph data of other DOs. We assume that parties communicate using a secure channel. As the colluding parties can monitor the computation's control flow and analyze data access patterns, we must use oblivious operations to ensure the input, output, and intermediate results are secured.

System overview. Figure 4.1(a) shows the flow of a typical GNN, where the initial node features $\mathbf{X}$ are passed through GNN layers to get the intermediate node features $\mathbf{X_1}$. An MPL takes the current node features $\mathbf{X_1}$ as input and exchanges messages between the nodes through the edges to compute new node features $\mathbf{X_1^*}$. The FTL transforms $\mathbf{X_1^*}$ into $\mathbf{X_2} = f_1(\mathbf{X_1^*}; \Theta_1)$, where Θ_1 summarizes the parameters in an FTL. In GNN, based on the model architecture, after executing multiple MPLs and FTLs, the final node features are computed to generate the inference result.

The CryptGNN system architecture, shown in Figure 4.1(b), has components at the SMPC parties, the MO, and the DOs. The components at the SMPC parties execute most of the secure inference protocols. The component at MO uploads the proprietary GNN model to the $\mathcal{P}$ SMPC parties in A-SS format, such that model parameters Θ are protected (Steps 1 and 2 in Figure 4.1(b)). Θ comprises of the parameters $\{\Theta_1, \Theta_2, \cdots\}$, where Θ_i is associated with the i-th FTL of the model. Following prior work [62, 127], we consider the architecture of the GNN model (i.e., type, sequence, and number of layers) to be shared by the MO with the parties and the clients. Thus, the parties know to invoke the secure versions of the insecure layers of the model.

CryptGNN's client-side component allows DOs to upload graph input data to the cloud as $([\![\mathbf{X}]\!], [\![\mathbf{S}]\!], [\![\mathbf{D}]\!])$, such that the node features $\mathbf{X}$ and the graph structure, i.e., the list of source indices $\mathbf{S}$ and destination indices $\mathbf{D}$, are protected (Steps 3 and 4 in Figure 4.1(b)). We consider directed, unweighted graphs as input, although CryptGNN protocols can be extended in a straightforward way for weighted graphs. During each inference request, the parties execute the secure protocols of CryptGNN to compute the output of each layer of the GNN model (Step 5 in Figure 4.1(b)). Finally, the client receives (Step 6 in Figure 4.1(b)) the shares of the final output from all parties to reconstruct the result (Step 7 in Figure 4.1(b)). CryptGNN comprises of the following two novel protocols:

CryptMPL: This protocol is used for secure message-passing in GNN in a $\mathcal{P}$-party A-SS setting. Executing MPL in the A-SS domain is difficult, as the encrypted features need to be passed through edges, while the source and destination nodes of each edge are encrypted. Our goal is to take the graph structure $(\mathbf{S}, \mathbf{D})$ and the feature matrix $\mathbf{A}$ (e.g., $\mathbf{X_1}$ in Figure 4.1(a)) as input in A-SS format, and compute the feature matrix $[\![\mathbf{A}^*]\!] = MPL([\![\mathbf{A}]\!], [\![\mathbf{S}]\!], [\![\mathbf{D}]\!])$ after the execution of an MPL layer,

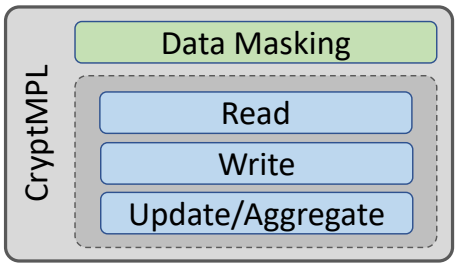


Figure 4.2 CryptMPL protocol stack.

while preserving the privacy of the input graph, intermediate results, and model parameters.

CryptMUL: The FTLs are computed using additions, multiplications, and comparisons. In A-SS, addition is cheap and can be computed locally, and comparison can be implemented using state-of-the-art techniques [200]. To eliminate the need for a trusted server and the costly online processing step in secure multiplications, CryptMUL employs a preprocessing step to generate auxiliary data for each client, which is used for multiple inference requests from the same client.

4.3 CryptMPL

This section presents the CryptMPL stack of protocols, shown in Figure 4.2, for privacy-preserving message-passing in GNN using a $\mathcal{P}$-party SMPC setting. To privately exchange messages through edges of a graph, represented as source/destination arrays, we develop novel protocols enabling the SMPC parties to read the feature vector of a source node, write the feature vector at the destination node, and update the node features by aggregation of intermediate feature vectors. CryptMPL also uses a novel data masking technique, where the client collaborates with the SMPC parties to protect the data against the threat model.

Algorithm 2 presents the pseudo-code for CryptMPL, which consists of invoking secure read, write, and aggregate functions (Lines 3-5). The details of the protocols behind these operations are presented in Subsections 4.3.1, 4.3.2, and 4.3.3, respectively. Read and write require each party to communicate with the other parties

in a ring-like structure, where the p-th party receives data from the $(p-1)$-th and sends data to the $(p+1)$-th party (the $\mathcal{P}$-th party transfers data to the first party). While executing the read and write protocols, CryptMPL uses a novel data masking technique to protect the transferred feature matrices, and the indices of source and destination nodes. To facilitate data masking, the client preprocesses a noise matrix and helps the parties mask their data with noise (Subsection 4.3.4). The accuracy of computation remains unaffected because the noise is eliminated from the final result (Line 7).

Algorithm 2 Secure Message Passing Layer, $\mathcal{F}_{CryptMPL}$

Input: $[\![\mathbf{A}]\!]$ (Feature Matrix), $[\![\mathbf{S}]\!]$ (Source Indices), $[\![\mathbf{D}]\!]$ (Destination Indices), $[\![\boldsymbol{\xi}^*]\!]$ (Noise)

Output: Output Feature Matrix, $[\![\mathbf{A}^*]\!]$

1: $[\![\mathbf{A}^*_\xi]\!] \leftarrow \mathcal{F}_{InitMatrix}(N, K)$

2: **for** $i \leftarrow 1, \ldots, M$ **do**

3: $\quad [\![\mathbf{Y}]\!] \leftarrow \mathcal{F}_{SR}([\![\mathbf{A}]\!], [\![\mathbf{S}[i]]\!])$

4: $\quad [\![\mathbf{G}]\!] \leftarrow \mathcal{F}_{SW}([\![\mathbf{Y}]\!], [\![\mathbf{D}[i]]\!])$

5: $\quad [\![\mathbf{A}^*_\xi]\!] \leftarrow \mathcal{F}_{SA}([\![\mathbf{A}^*_\xi]\!], [\![\mathbf{G}]\!])$

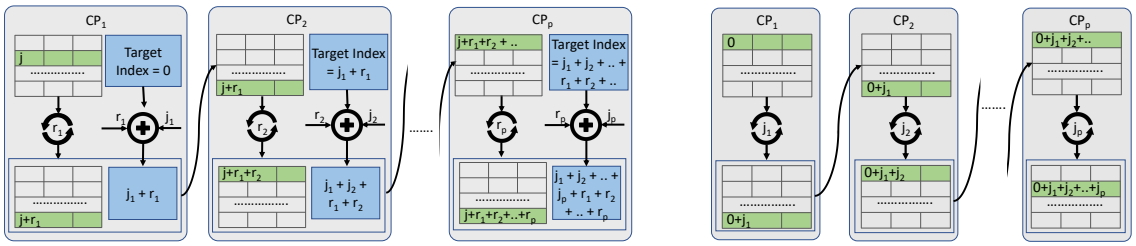
6: **end for**

7: $[\![\mathbf{A}^*]\!] \leftarrow [\![\mathbf{A}^*_\xi]\!] - [\![\boldsymbol{\xi}^*]\!]$

8: **return** $[\![\mathbf{A}^*]\!]$

4.3.1 Reading the feature vector of a source node

The index in the source nodes' array and the feature vector of a source node for an edge are stored in the A-SS domain. The secure read ($\mathcal{F}_{SR}$ in Algorithm 2, Line 3) accesses the feature vector without leaking the index and the features of the source node. The main idea of $\mathcal{F}_{SR}$ requires each party to rotate their share of the current feature matrix $\mathbf{A}$ and shift their share of the source index by the same random amount, and



(a) Secure Read (b) Secure Write

Figure 4.3 Flow of a share of a matrix for read (a) and write (b) in CryptMPL with $\mathcal{P}$ SMPC parties.

then share the updated matrix and index with the next party. The random amount is different at each party. After all the parties have rotated the feature matrix and shifted the source index, each party reads the vector at the updated index of the rotated matrix. Since both the rotation and the shift are performed by the same total amount, each party receives a correct share of the source feature vector.

This procedure is illustrated in Figure 4.3(a) and detailed in the following. For a source node $j \in \mathbf{S}$, its corresponding feature vector is $\mathbf{A}[j]$. The secret-shared versions of the index and the feature vector are $[\![j]\!]$ and $[\![\mathbf{A}[j]]\!]$, respectively. In $\mathcal{F}_{SR}$, party CP_p securely retrieves $[\![\mathbf{A}[j]]\!]_p$. For example, to retrieve $[\![\mathbf{A}[j]]\!]_1$, the parties execute these steps:

1. CP_1 initializes two variables: a target index $j' = 0$ and a target matrix $[\![\mathbf{A}']\!]_1 = [\![\mathbf{A}]\!]_1$. The target index and the target matrix pass through the parties in the ring and are updated by the parties (Steps 2-5). In Step 6, CP_1 reads the vector at the updated index of the updated matrix.
2. To protect the share of the source index, CP_p adds a random integer r_p to $[\![j]\!]_p$, and updates the target index j' as, $j' \leftarrow j' + [\![j]\!]_p + r_p$.
3. To align the target matrix, CP_p rotates the rows of $[\![\mathbf{A}']\!]_1$ by r_p, i.e., $[\![\mathbf{A}']\!]_1 \leftarrow rotate([\![\mathbf{A}']\!]_1, r_p)$.
4. CP_p transfers $[\![\mathbf{A}']\!]_1$ and j' to CP_{p+1}, which repeats Steps 2 & 3 to update $[\![\mathbf{A}']\!]_1$ and j'.
5. After the operations at the $\mathcal{P}$-th party, the information is transferred to the first party CP_1.

6. CP_1 gets $j' = \sum_{p=1}^{\mathcal{P}} [\![j]\!]_p + r_p = j + \sum_{p=1}^{\mathcal{P}} r_p$. Correspondingly, $[\![\mathbf{A}']\!]_1$ is rotated for $\sum_{p=1}^{\mathcal{P}} r_p$ times, i.e, $[\![\mathbf{A}[j]]\!]_1 = [\![\mathbf{A}'[j + \sum_{p=1}^{\mathcal{P}} r_p]]\!]_1$. Thus, CP_1 gets $[\![\mathbf{A}'[j']]\!]_1 = [\![\mathbf{A}[j]]\!]_1$.

All parties follow the same procedure in parallel to retrieve the $\mathcal{P}$ shares of $[\![\mathbf{A}[j]]\!]$. This procedure protects each party's share of the source index through the random shifting of its value. However, this is insufficient to safeguard the graph data, because: (a) It is possible to reconstruct the feature matrix, as each party gets all of the shares of $\mathbf{A}$, and (b) Each party can determine the actual value of the source index by searching the accessed feature vector in $\mathbf{A}$. To solve these problems, each party adds a random noise (a matrix containing random values) to mask the shares of $\mathbf{A}$ before transferring them to the other parties. Thus, $\mathcal{F}_{SR}$ addresses the aforementioned issues, since: (a) Due to the presence of noise, the feature matrix cannot be reconstructed correctly; (b) Since the feature matrix is modified by all parties, the parties cannot determine the source index. Therefore, the parties can securely access the source feature vector. Eliminating noise from the final result is discussed in Subsection 4.3.4.

4.3.2 Writing messages to the destination node

This secure protocol ($\mathcal{F}_{SW}$ in Algorithm 2, Line 4) creates an intermediate matrix $\mathbf{G}$ of the same dimensions as the output feature matrix, and writes the feature vector $\mathbf{Y}$ at the index in $\mathbf{G}$ corresponding to the destination node's index in the destination nodes' array $\mathbf{D}$. Unlike read, which can use rotation operations to preserve index privacy, write must know the destination index to write the vector at the correct position. As the destination node of each edge is encrypted, to write a feature vector $\mathbf{Y}$ at the destination index $j \in \mathbf{D}$ of a matrix $\mathbf{G}$, the parties need to coordinate. In the secret-shared domain, each party CP_p initializes the share of $\mathbf{G}$ as $[\![\mathbf{G}]\!]_p = \mathbf{0}$ (i.e., initialize $\mathbf{G}$ with all entries zero). If party CP_p has the shares of $\mathbf{Y}$ and j, i.e., $[\![\mathbf{Y}]\!]_p$

and $[\![j]\!]_p$, our goal is to get $[\![\mathbf{G}[j]]\!]_p = [\![\mathbf{Y}]\!]_p$, without leaking the target index and the feature vector.

The main idea of the write protocol requires each party to write its share of the feature vector $\mathbf{Y}$ at the 0-th index of its share of $\mathbf{G}$ and transfer it to the next party in the ring. Each party rotates the share of the matrix $\mathbf{G}$ by its share of the destination index. Thus, the feature vector reaches the correct destination index of $\mathbf{G}$. For example, Figure 4.3(b) shows the following steps to write $[\![\mathbf{Y}]\!]_1$ at $[\![\mathbf{G}[j]]\!]_1$.

1. CP_1 writes the vector $[\![\mathbf{Y}]\!]_1$ at index 0 of $[\![\mathbf{G}]\!]_1$ as $[\![\mathbf{G}[0]]\!]_1 = [\![\mathbf{Y}]\!]_1$. The matrix $[\![\mathbf{G}]\!]_1$ will pass through the parties in the ring and be updated by other parties in Steps 2-4. In Step 5, CP_1 gets the updated matrix $[\![\mathbf{G}]\!]_1$, where $[\![\mathbf{Y}]\!]_1$ is written at the correct destination index.
2. CP_p rotates the matrix $[\![\mathbf{G}]\!]_1$ by $[\![j]\!]_p$.
3. CP_p transfers $[\![\mathbf{G}]\!]_1$ to $p+1$-th party for $1 \leq p \leq \mathcal{P}$. CP_{p+1} repeats Steps 2 to update $[\![\mathbf{G}]\!]_1$.
4. After the operations at the $\mathcal{P}$-th party, the matrix $[\![\mathbf{G}]\!]_1$ is transferred to the first party.
5. CP_1 gets $[\![\mathbf{G}]\!]_1$ which is rotated by $\sum_{p=1}^{\mathcal{P}} j_p = j$ times. Due to the overall rotation, $[\![\mathbf{Y}]\!]_1$ is moved to j-th index of $[\![\mathbf{G}]\!]_\mathbf{1}$, equivalent to $[\![\mathbf{G}[j]]\!]_1 = [\![\mathbf{Y}]\!]_1$.

All parties follow the same procedure in parallel to write their shares of $\mathbf{Y}$ at the destination index of the matrix $\mathbf{G}$. During write, each party's share of the destination index is protected. However, the actual destination index j is revealed from the final matrix, since the values in the final matrix are zero for all indices other than j. To solve this problem, CP_p adds random noise to mask its share of $[\![\mathbf{G}]\!]_p$ while sharing it with the other parties. Thus, the final matrix is masked by all parties, and the destination index cannot be determined by observing the values in the matrix. The procedure to remove the noise from the final result is discussed in Subsection 4.3.4.

4.3.3 Updating the feature matrix

Unlike read and write, secure aggregation ($\mathcal{F}_{SA}$ in Algorithm 2, Line 5) can be implemented using standard SMPC techniques. The output feature matrix of the same size as the input feature matrix is initialized by each party as $[\![\mathbf{A}^*]\!] = [\![\mathbf{0}]\!]$. Executing one round of read and write protocols process one edge, where the intermediate result matrix $[\![\mathbf{G}]\!]$ contains the feature vector of node $[\![\mathbf{S}[i]]\!]$ at index $[\![\mathbf{D}[i]]\!]$ after processing the i-th edge. CP_p updates the feature matrix $[\![\mathbf{A}^*]\!]$ with the result: $[\![\mathbf{A}^*]\!]_p = [\![\mathbf{A}^*]\!]_p + [\![\mathbf{G}]\!]_p$.

4.3.4 Putting things together with preprocessing

The protocols $\mathcal{F}_{SR}$, $\mathcal{F}_{SW}$ and $\mathcal{F}_{SA}$ process all the edges in the graph. During read and write, each party masks the shares of the feature matrices to protect the graph data. Masking a matrix with noise involves adding random values to the original matrix. Here, we describe the preprocessing stage executed at the client side to help the parties generate the noise matrices to mask the original data and eliminate the noise from the output for the correct result of the MPL.

As the client has the graph data structure, it can execute the message-passing on a noise matrix $\boldsymbol{\xi} \in \mathbb{R}^{N \times K}$ to get a feature matrix $\boldsymbol{\xi}^*$ and share both $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^*$ with the SMPC parties in a secret-shared manner. Each party can mask its feature matrix with the share of $\boldsymbol{\xi}$ and calculate the feature matrix $\mathbf{A}^*_\xi$ by executing MPL on the masked feature matrix. Finally, it removes the noise $\boldsymbol{\xi}^*$ from the $\mathbf{A}^*_\xi$ to generate the actual result $\mathbf{A}^*$. The steps of this process are as follows:

- The client calculates the effect of noise on each node after executing an MPL round (Equation (4.4)).
- Each party executes the MPL to get the output feature matrix on the masked node features (Equation (4.5)).
- Each party removes the effect of noise to obtain the correct feature matrix (Equation (4.6)).

$$\boldsymbol{\xi}^*[j] = \sum_{j \in \mathcal{N}(i)} \boldsymbol{\xi}[j] \tag{4.4}$$

$$\mathbf{A}^*_{\xi}[i] = \sum_{j \in \mathcal{N}(i)} \mathbf{A}[j] + \boldsymbol{\xi}[j] \tag{4.5}$$

$$\mathbf{A}^*[i] = \mathbf{A}^*_{\xi}[i] - \boldsymbol{\xi}^*[i] \tag{4.6}$$

To mask the feature matrices, CP_p creates a noise matrix $\boldsymbol{\xi}_p$ of the same dimensions as the feature matrix, and adds $\boldsymbol{\xi}_p$ to the share of the feature matrix $[\![\mathbf{A}]\!]_p$. However, if the same noise is used to mask $[\![\mathbf{A}]\!]_p$ while processing each edge, an attacker can identify the node degree based on the number of times the same value is accessed by a party. To prevent this issue, each party needs to generate different noise matrices $\boldsymbol{\xi}_r$ at each round r of the read and write operation. In read and write (Subsections 4.3.1 and 4.3.2), noise is added to the party's own share, and to the matrices received from the other parties, such that the matrices cannot be recognized at the end of the process, and the source and destination indices cannot be detected by observing the values.

During the initialization stage, the client shares different integer values as seeds to each party, which are used in a pseudo-random function (PRF) to generate all the rotation amounts and noise matrices. At the client side, similar noise matrices are used to compute the effect of noise $\boldsymbol{\xi}^*$. The client shares $\boldsymbol{\xi}^*$ in secret-shared manner with each party. After processing all edges, each party removes the noise $\boldsymbol{\xi}^*$ to retrieve the correct feature matrix.

Processing edges in batches. To execute an MPL layer, CryptMPL needs to process all edges in the graph, which involves M rounds of read and write executions, where M is the number of edges. To reduce the number of rounds, and consequently the computation and communication overhead, CryptMPL processes the edges in batches. Our batching technique executes MPL with low overhead while preserving the privacy of the graph structure. The number of edges in a batch is configurable. For

example, a batch of three nodes in secret-shared format $[\![a_1, a_2, a_3]\!]$ can be represented as relative indices $[0, a_2 - a_1, a_3 - a_1]$ in plain text with respect to a_1, which is still represented in secret-shared format as $[\![a_1]\!]$. Both source and destination indices can be represented in this way.

For batching, the client divides the edges into batches and calculates the relative indices with respect to the first index of a batch. The first indices of all batches from $\mathbf{S}$ and $\mathbf{D}$ are stored as two vectors $[\![\mathbf{S_f}]\!]$ and $[\![\mathbf{D_f}]\!]$. The relative indices for all batches are concatenated to create two vectors $\mathbf{S_r}$ and $\mathbf{D_r}$ in plain text. The client uploads $[\![\mathbf{X}]\!]$, $[\![\mathbf{S_f}]\!]$, $[\![\mathbf{D_f}]\!]$, $[\![\boldsymbol{\xi}^*]\!]$, $\mathbf{S_r}$, $\mathbf{D_r}$ and the seed to the SMPC parties.

To further reduce the number of communication rounds, each party concatenates masked feature matrices for all batches to create a matrix of size (R, N, K), where the dimension of the feature matrix is (N, K) and the number of batches is R. Each sub-matrix of size $(1, N, K)$ can be rotated by a different amount and the concatenated version can be passed to the other parties for read operation. In this way, the read operations for all batches can be executed in a single round. Similarly, write operations can be executed in a single round by concatenating $\mathbf{G}$ matrices for all batches.

Finally, let us note that there is a trade-off between performance (fewer batches) and security (more batches). Using the relative order of the indices in each batch, the parties may infer the graph structure. The analysis in Subsection 4.5.2 shows that the probability of correct reconstruction of the graph structure by the parties is N^{-2R}, where N is the number of nodes in the graph and R is the number of batches.

4.3.5 A simple example

In this example, we consider three computing parties CP_1, CP_2 and CP_3 take an input feature matrix $\mathbf{A}$ and compute the message-passing layer to generate the output feature matrix $\mathbf{A}^*$. Here, the matrix $\mathbf{A}$ represents $K = 2$ features for each of the

$N = 4$ nodes in the graph $\mathcal{G}$. For simplicity, we consider a simple graph and illustrate the protocols for computing the message passing through an edge from node ① to node ②. We consider the indices to be 1-indexed. Therefore, for this edge, the source index $S = 1$ and the destination index $D = 2$. In the A-SS domain, CP_p has the shares of node features, source index, and destination index as $[\![\mathbf{A}]\!]_p$, $[\![S]\!]_p$, and $[\![D]\!]_p$ respectively. Here, $\mathbf{A} = \sum_{p=1}^{\mathcal{P}} [\![A]\!]_p$, $S = (\sum_{p=1}^{\mathcal{P}} [\![S_p]\!]) \mod N + 1$ and $D = (\sum_{p=1}^{\mathcal{P}} [\![D_p]\!]) \mod N + 1$.

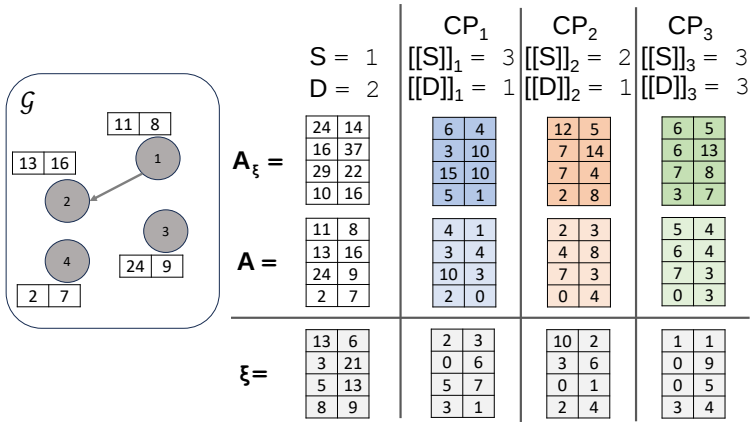


Figure 4.4 Input feature matrix $\mathbf{A}$, noise matrix $\boldsymbol{\xi}$, and masked feature matrix $\mathbf{A}_{\boldsymbol{\xi}}$, for a graph $\mathcal{G}$ with $N = 4$ nodes and $K = 2$ features.

The owner of the graph, the client, has the information about the graph structure, i.e., it knows the source index S and the destination index D. However, the client doesn't have any information about the current feature matrix $\mathbf{A}$. Nonetheless, the client can assist the computing parties in masking their shares and executing the message-passing layer, ensuring that the node features and graph structure (S and D) remain protected throughout the execution of CryptMPL.

To preserve the node features, the client shares the seeds s_p with the computing parties CP_p. Each party uses its seed to create matrix $\boldsymbol{\xi}_p$ which is used to mask its share $[\![\mathbf{A}]\!]_p$ as $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]_p = [\![\mathbf{A}]\!]_p + \boldsymbol{\xi}_p$ (shown in Figure 4.4). The parties follow CryptMPL read protocol $\mathcal{F}_{SR}$ and write protocol $\mathcal{F}_{SW}$ to obtain $[\![\mathbf{A}^*_{\boldsymbol{\xi}}]\!]$ by executing

the message-passing on on $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]$. Finally, the computing parties remove the effect of noise from $[\![\mathbf{A}^*_{\boldsymbol{\xi}}]\!]$ to obtain the correct result $[\![\mathbf{A}^*]\!]$.

During the read operation (Subsection 4.3.1), using the seed value s_p, CP_p generates matrix $\mathbf{X}_{pi}$, which is used to mask the share of the feature matrix from CP_i. Additionally, CP_p generates random values r_{pi} and rotates the share of the feature matrix from party CP_i by r_{pi}. During the write operation (Subsection 4.3.2), CP_p generates $\mathbf{Y}_{pi}$ which is used to mask the share of the feature matrix from CP_i. Additionally, CP_p rotates the share of the feature matrix from party CP_i by D_p. We use $\mathbf{X}_{pimn}$ and $\mathbf{Y}_{pimn}$ to represent the value $\mathbf{X}_{pi}[m][n]$ and $\mathbf{Y}_{pi}[m][n]$ respectively.

Client-side preprocessing. As described in Subsection 4.3.4, the client executes the message-passing step on the noise matrix $\boldsymbol{\xi}$ to generate $\boldsymbol{\xi}^*$. $\boldsymbol{\xi}^*$ is shared with the computing parties in A-SS format, so that they can remove the effect of noise from $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]_p$. We illustrate the client-side preprocessing step to generate a share $[\![\boldsymbol{\xi}^*]\!]_1$ in Figure 4.5. During the read operation, the client simulates the effect of rotations r_{p1} and data masking $\mathbf{X}_{p1}$ on $[\![\boldsymbol{\xi}^*]\!]_1$. Figure 4.5 shows the final state of $[\![\boldsymbol{\xi}^*]\!]_1$ after passing the share through all parties, assuming $r_{11} = 2, r_{21} = 3$ and $r_{31} = 2$. Client completes the read operation by reading the vector at index $(\sum_{p=1}^{\mathcal{P}}([\![S]\!]_p + r_{p1}))$ mod $N + 1 = ((3 + 2) + (2 + 3) + (3 + 2))$ mod $4 + 1 = 4$ to get the vector $[\![\mathbf{Y}_{\boldsymbol{\xi}}]\!]_1 = [2 + Z_{11}, 3 + Z_{12}]$, where the cumulative noises, $Z_{11} = \mathbf{X}_{1111} + \mathbf{X}_{2131} + \mathbf{X}_{3121}$, $Z_{12} = \mathbf{X}_{1112} + \mathbf{X}_{2132} + \mathbf{X}_{3122}$. Similarly, the client computes other shares, $[\![\mathbf{Y}_{\boldsymbol{\xi}}]\!]_2$ and $[\![\mathbf{Y}_{\boldsymbol{\xi}}]\!]_3$.

For the write operation, the client creates a feature matrix $\mathbf{G}_{\boldsymbol{\xi}} = \mathbf{0}$ and writes the read result at index 0 as $[\![\mathbf{G}_{\boldsymbol{\xi}}[0]]\!]_1 = [\![\mathbf{Y}_{\boldsymbol{\xi}}]\!]_1$. Then, it simulates the effect of rotations D_p and data maskings $\mathbf{Y}_{p1}$ on $[\![\mathbf{G}_{\boldsymbol{\xi}}]\!]_1$. Finally, it gets the share $[\![\mathbf{G}_{\boldsymbol{\xi}}]\!]_1$ where $[\![\mathbf{Y}_{\boldsymbol{\xi}}]\!]_1$ is moved to the target index $\sum_{p=1}^{\mathcal{P}} D_p = D$. Here, all values in $[\![\mathbf{G}_{\boldsymbol{\xi}}]\!]_1$ are masked with noise added by all parties. Similarly, client gets other shares $[\![\mathbf{G}_{\boldsymbol{\xi}}]\!]_2$ and $[\![\mathbf{G}_{\boldsymbol{\xi}}]\!]_3$.

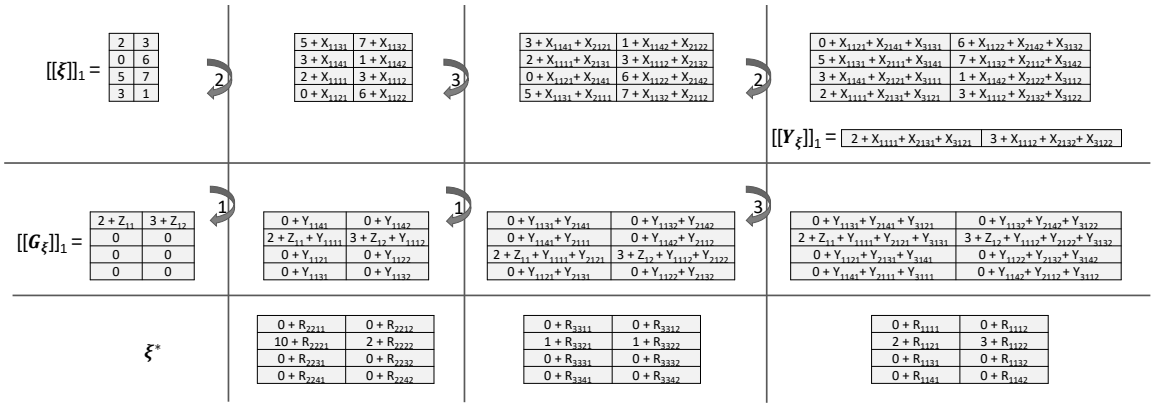


Figure 4.5 CryptMPL: client side preprocessing step.

Thus, the output of message passing on $\boldsymbol{\xi}$ is masked with noise as $[\![\boldsymbol{\xi}]\!]_p^* = [\![\mathbf{G}]\!]_p + \mathbf{R}_{pp}$, where $[\![\mathbf{G}]\!]_p$ represents the p-th party's share of the final result if noise were not added and $\mathbf{R}_{pp}$ are the accumulated noises. To upload the result $\boldsymbol{\xi}^*$, client creates different shares of $\boldsymbol{\xi}^*$ as $[\![\boldsymbol{\xi}^*]\!]_p = [\![\mathbf{G}]\!]_p + \mathbf{T}_{pp}$, where $\sum_{p=1}^{\mathcal{P}} \mathbf{T}_{pp}[m][n] = \sum_{p=1}^{\mathcal{P}} \mathbf{R}_{pp}[m][n]$.

CryptMPL protocols in the cloud. The computing parties CP_p initialize a output feature matrix as $[\![\mathbf{A}^*_{\boldsymbol{\xi}}]\!]_p = \mathbf{0}$. CP_p perform similar operations on the shares of $\mathbf{A}_{\boldsymbol{\xi}}$ as the clients do on $\boldsymbol{\xi}$ during the preprocessing step.

Figure 4.6 shows the operations on $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]_1$ to obtain $[\![\mathbf{Y}]\!]_1$ after read operation and $[\![\mathbf{G}]\!]_1$ after the write operation. After executing the read operation on $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]_1$ for the source index $[\![S]\!]$, CP_1 reads the vector at the updated target index $(\sum_{p=1}^{\mathcal{P}}([\![S]\!]_p + r_{p1}))\mod N+1 = ((3+2)+(2+3)+(3+2)) \mod 4+1 = 4$ to get $[\![\mathbf{Y}]\!]_1$. Since $[\![\mathbf{Y}]\!]_1$ is obtained by rotation and masking of $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]_1$ with noise from all parties, the computing parties cannot learn the original index or values, even if there is collusion among $\mathcal{P}-1$ parties. Similarly CP_2 and CP_3 obtain $[\![\mathbf{Y}]\!]_2$ and $[\![\mathbf{Y}]\!]_3$ respectively.

During the write operation, each party CP_p sets $[\![\mathbf{G}[0]]\!]_p = [\![\mathbf{Y}]\!]_p$. Then, CP_p adds noise Y_{pi} to i-th party's share, rotates it by $[\![D]\!]_p$, and passes it to the next party. As shown in Figure 4.6, CP_1 obtains $[\![\mathbf{G}]\!]_1$ which is modified by all parties. Due to rotations by $\sum_{p=1}^{\mathcal{P}} D_p \mod N+1 = (1+1+3) \mod 4+1 = 2$, $[\![\mathbf{Y}]\!]_1$ reaches at the destination index of $[\![\mathbf{G}]\!]_1$. Due to the rotations and noises, the computing

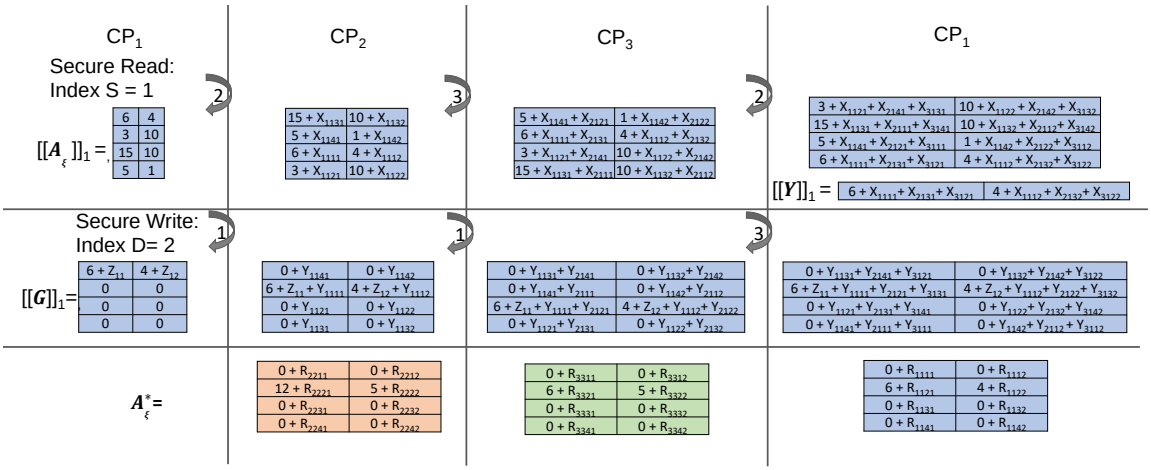


Figure 4.6 CryptMPL flow at the computing parties.

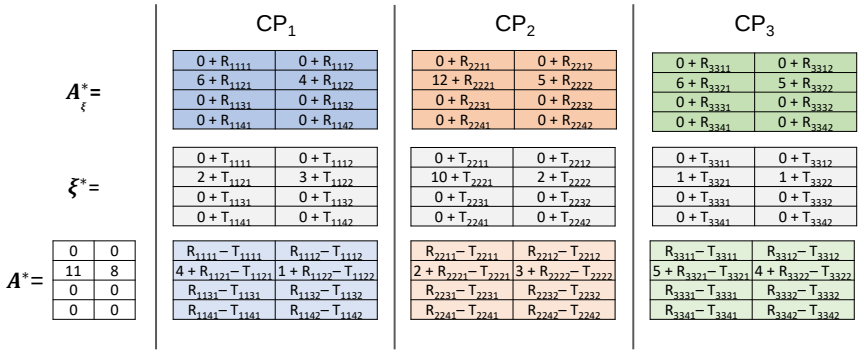


Figure 4.7 CryptMPL - removal of noise to get the output feature matrix.

parties cannot learn the original index or values, even if there is collusion among $\mathcal{P}-1$ parties. In parallel, CP_2 and CP_3 obtain $[\![\mathbf{G}]\!]_2$ and $[\![\mathbf{G}]\!]_3$ respectively. Then, the matrix $[\![\mathbf{G}]\!]_p$ is used to update the output feature matrix for the MPL layer as $[\![\mathbf{A}^*_{\boldsymbol{\xi}}]\!]_p = [\![\mathbf{A}^*_{\boldsymbol{\xi}}]\!]_p + [\![\mathbf{G}]\!]_p$.

However, since $[\![\mathbf{A}^*_{\boldsymbol{\xi}}]\!]$ is computed on the matrix $[\![\mathbf{A}_{\boldsymbol{\xi}}]\!]$, it is required to remove the noise. As shown in Figure 4.7, CP_p removes the effect of noise by computing $[\![\mathbf{A}^*]\!] = [\![\mathbf{A}_{\boldsymbol{\xi}}]\!] - [\![\boldsymbol{\xi}^*]\!]$. Thus, the shares constitute the correct $[\![\mathbf{A}^*]\!]$ in A-SS domain as $[\![\mathbf{A}]\!]^* = \sum_{p=1}^{N} [\![\mathbf{A}]\!]^*_p$, since $\sum_{p=1}^{\mathcal{P}} (\mathbf{R}_{pp} - \mathbf{T}_{pp}) = \mathbf{0}$. Nevertheless, the computing parties can not determine the actual values from $\mathcal{P}-1$ shares of $[\![\mathbf{A}^*]\!]$.

Following this approach, we can compute the message-passing for each edge of the graph. We can process multiple edges in a batch, since secure read and write operation does not reveal the actual source and destination indices of the edges. For

each batch, the client and parties use different sets of noises to protect the node features and source-destination indices by observing data in different batches. To obtain the correct result, the client only needs to share the initial noise $[\![\boldsymbol{\xi}]\!]$, overall noise $[\![\boldsymbol{\xi}^*]\!]$ and the seeds (to generate rotation amount and noise matrices for each batch) to the computing parties.

4.3.6 Optimizations

This section explores potential extensions of CryptMPL, including support for weighted edges in graphs, protection of the number of edges in input graphs, and advanced message-passing operations in popular GNNs.

Supporting weighted edges. To support the weighted edges, client needs to upload the list of weights for each edge as $\mathbf{W}$ along with the list of edges, where $\mathbf{W}[i]$ is the weight for the edge from source $\mathbf{S}[i]$ to destination $\mathbf{D}[i]$. To protect the weights, the $\mathbf{W}$ is list in A-SS format as $[\![\mathbf{W}]\!]$.

To apply the weights, the computing parties multiply the weight $[\![\mathbf{W}[i]]\!]$ with the read result for i-th edge $[\![\mathbf{Y}[i]]\!]$. CryptMPL can use CryptMUL's element-wise multiplication protocol to execute this operation. This can be done with one additional round of communication among the computing parties.

In the data-processing step, client also mulitplies the weights to the read result to compute the overall noise $\boldsymbol{\xi}$. Considering the weights, the Equations (4.4), (4.5) and (4.6) can be modified as follows. Here, Equation (4.7) considers the weights to compute the overall noise, Equation (4.8) computes the message-passing on the masked feature matrix and applies the edge weights, and finally Equation (4.8) removes the overall noise to obtain the correct output feature matrix.

$$\boldsymbol{\xi}^*[j] = \sum_{j \in \mathcal{N}(i)} \mathbf{W}[j] \times \boldsymbol{\xi}[j] \tag{4.7}$$

$$\mathbf{A}^*_{\xi}[i] = \sum_{j \in \mathcal{N}(i)} \mathbf{W}[j] \times (\mathbf{A}[j] + \boldsymbol{\xi}[j]) \tag{4.8}$$

$$\mathbf{A}^*[i] = \mathbf{A}^*_{\xi}[i] - \boldsymbol{\xi}^*[i] = \sum_{j \in \mathcal{N}(i)} \mathbf{W}[j] \times \mathbf{A}[j] \tag{4.9}$$

Protecting the number of edges. If a client wants to protect the number of edges in the graph, it can use any number of fake edges (S_f, D_f), $1 \leq S_f, D_f \leq N$, with weight $W_f = 0$. In A-SS format, the parties can not learn if the weight is zero or not. And, these fake edges do not affect the overall result, as the weights are zero.

Supporting complex message-passing. CryptMPL provides secure read and write protocols that can be used to extend CryptGNN to support complex message-passing in GNN architectures by incorporating additional operations (e.g., node sampling, concatenation, etc.) using standard SMPC techniques or by designing efficient protocols. For example, in GraphSAGE [39], instead of performing message-passing on the entire graph, nodes are randomly sampled, and message-passing is carried out only through the adjacent edges. In the case of secure inference, this sampling operation must be executed in a way that does not reveal the graph structure. While oblivious sampling operations in an SMPC setting may be feasible, they are computationally expensive and significantly increase overhead. An alternative solution is to perform the sampling operation on the client side, where the graph structure is already known. After sampling, the client can upload the list of edges following the CryptMPL protocol, allowing secure message-passing to be executed on the server side.

Similarly, CryptGNN can support Graph Attention Networks (GATs) [201], which require the computation of attention coefficients for each edge in the message-passing layer. Since CryptMPL can securely read the feature vector of any source node, it can be extended to concatenate the feature vectors of the two nodes connected by each edge and compute pairwise attention coefficients using standard SMPC techniques.

4.4 CryptMUL

Typically, a GNN comprises multiple FTLs, along with MPLs. In A-SS, the primary bottleneck in evaluating FTLs is the multiplication operation. Linear layers need matrix multiplications, and non-linear layers require element-wise multiplications. To generate Beaver Triples (discussed in Section 4.1) for multiplications in A-SS, previous studies [11] have relied on techniques such as HE, oblivious transfer (OT), a trusted third party (TTP), or a combination thereof. However, using HE and OT is costly in terms of both computation and communication. Since CryptGNN is designed for MLaaS, it must scale to support numerous inference requests from each client. Consequently, repeatedly employing HE or OT is impractical. While using a TTP is less resource-intensive, it requires an additional third party that must not collude with the computing parties. Moreover, communication with the TTP is necessary for each multiplication operation.

To execute the multiplication operations in GNN, we design CryptMUL, which offers two benefits: (i) performing multiplications without a trusted server, and (ii) lower overhead due to preprocessing. Our CryptMUL conducts offline preprocessing to generate auxiliary data, which can be used to easily create a set of Beaver triples [128] required for multiplication operations in multiple inference requests from the same client, thereby improving performance while preserving data privacy. Although our protocols use existing HE- or OT-based techniques in the offline phase to generate auxiliary data for a client, our contribution lies in efficiently reusing this data for multiple inference requests from the same client with only one round of communication among the parties.

In the case of matrix multiplication, CryptMUL generates a new set of triples considering the properties of matrix operations, where one matrix holds the model parameters that remain the same for all inference requests. For element-wise multiplication, CryptMUL generates the auxiliary data in a M-SS format, which can be easily

transformed to create a new set of triples for each inference request. In this section, we describe the two CryptMUL protocols for: (a) secure matrix-multiplication, and (b) secure element-wise multiplication, along with an analysis of these protocols. While we discuss these protocols in the context of GNN inference, they are generic and can be applied in similar scenarios for other types of model architectures.

4.4.1 Secure matrix-multiplication

In order to compute the linear layers in GNNs, we must conduct matrix multiplication between the intermediate state matrix $[\![\mathbf{X}]\!] \in \mathbb{R}^{N \times K}$ and the parameter matrix of the linear layer $[\![\mathbf{Y}]\!] \in \mathbb{R}^{K \times K'}$ to transform K features into K' features. While the values of K and K' remain constant in GNN inference, the number of nodes in the graph, denoted as N, can vary with each inference request for the same GNN model.

To perform matrix multiplication efficiently in the A-SS setting, we employ the Beaver triple technique, as explained in Section 4.1. By utilizing a Beaver triple ($[\![\mathbf{A}]\!]$, $[\![\mathbf{B}]\!]$, $[\![\mathbf{C}]\!]$), we can calculate $[\![\mathbf{Z}]\!] = [\![\mathbf{X}]\!] \otimes [\![\mathbf{Y}]\!]$. However, it is crucial to note that using the same triple to compute the same linear layer for two different inference requests may lead to a privacy risk. This is because such a scenario reveals the differences ($\mathbf{U} = \mathbf{X} - \mathbf{A}$) and ($\mathbf{V} = \mathbf{Y} - \mathbf{B}$), and using the same $\mathbf{A}$ and $\mathbf{B}$ could disclose the relative changes in $\mathbf{X}$ and $\mathbf{Y}$ across different requests. Unfortunately, generating a new Beaver triple for each inference request using state-of-the-art techniques such as HE or OT is impractical due to their high overhead.

The following observations help us to solve this problem:

- The intermediate feature matrix $\mathbf{X}$ is derived from the input feature matrix of the graph. Revealing $\mathbf{U}$ does not disclose $\mathbf{X}$, unless $\mathbf{A}$ becomes known to any party. However, it is infeasible to use the same $\mathbf{A}$ for different inference requests, as it would reveal the relative changes in $\mathbf{X}$ across requests.
- The trained parameter matrix $\mathbf{Y}$ remains constant for a GNN model, and the same $\mathbf{Y}$ is used for all inference requests. Therefore, we can use the same $\mathbf{B}$ in the Beaver triple for all inference requests. As long as $\mathbf{B}$ remains unknown to the computing parties, revealing $\mathbf{V}$ will not disclose $\mathbf{Y}$.

Based on these observations, we plan to use the same $\mathbf{B}$ in all Beaver triples. However, $\mathbf{A}$ needs to be changed in different requests, and consequently, $\mathbf{C}$ needs to be adjusted to hold the property of Beaver triples. To derive a new set of matrices, denoted as $(\mathbf{A}', \mathbf{B}, \mathbf{C}')$, from the initial Beaver triple $(\mathbf{A}, \mathbf{B}, \mathbf{C})$, we can construct a row in $\mathbf{A}'$ through a linear combination of the rows of $\mathbf{A}$ and use a similar linear combination of rows from $\mathbf{C}$ to compute the corresponding row in $\mathbf{C}'$. Specifically, if we express $\mathbf{A}'[j] = \sum_{i=1}^{N} k_{ji} * \mathbf{A}[i]$, then $\mathbf{C}'$ can be computed as $\mathbf{C}'[j] = \sum_{i=1}^{N} k_{ji} * \mathbf{C}[i]$. Following this approach, we propose a secure protocol $\mathcal{F}_{MatMul}$ to perform matrix multiplication in a GNN's linear layer using the following steps:

1. During the preprocessing stage (Algorithm 3 Offline Phase), $\mathcal{F}_{InitBeaver}$ generates an initial Beaver triple $(\llbracket \mathbf{A} \rrbracket, \llbracket \mathbf{B} \rrbracket, \llbracket \mathbf{C} \rrbracket)$ using HE [202] or OT [203] for each client. Here, $\mathbf{A} \in \mathbb{R}^{N \times K}$, $\mathbf{B} \in \mathbb{R}^{K \times K'}$ and $\mathbf{C} \in \mathbb{R}^{N \times K'}$. K and K' are fixed and depend on the number of input features and output features of the linear layer respectively. N is the maximum number of rows we may need to support and depends on the maximum number of nodes in a graph.
2. During inference, each party uses the same pseudo-random function $\mathcal{F}_{RandComb}$ to pick a random linear combination of the rows to modify $\llbracket \mathbf{A} \rrbracket$ to $\llbracket \mathbf{A}' \rrbracket$ and $\llbracket \mathbf{C} \rrbracket$ to $\llbracket \mathbf{C}' \rrbracket$ (Algorithm 3 Online Phase Lines 1-2). All computing parties execute the same operations locally to generate new $\llbracket \mathbf{A}' \rrbracket$ and $\llbracket \mathbf{C}' \rrbracket$ based on the number of nodes in the input graph.
3. Use the new Beaver triple $(\llbracket \mathbf{A}' \rrbracket, \llbracket \mathbf{B} \rrbracket, \llbracket \mathbf{C}' \rrbracket)$ to compute $\llbracket \mathbf{X} \rrbracket \otimes \llbracket \mathbf{Y} \rrbracket$ (Algorithm 3 Online Phase Line 3).

Algorithm 3 Secure Matrix-Multiplication, $\mathcal{F}_{MatMul}$

Input: $\llbracket \mathbf{X} \rrbracket$, $\llbracket \mathbf{Y} \rrbracket$ **Output:** Compute $\llbracket \mathbf{Z} \rrbracket = \llbracket \mathbf{X} \rrbracket \otimes \llbracket \mathbf{Y} \rrbracket$

Offline Phase, $\mathcal{F}_{InitBeaver}$

1: Generate Beaver triples $(\llbracket \mathbf{A} \rrbracket, \llbracket \mathbf{B} \rrbracket, \llbracket \mathbf{C} \rrbracket)$ for each client

Online Phase

1: $\llbracket \mathbf{A} \rrbracket, \llbracket \mathbf{B} \rrbracket, \llbracket \mathbf{C} \rrbracket \leftarrow \mathcal{F}_{InitBeaver}(CID)$ %CID is the current client's ID

2: $\llbracket \mathbf{A}' \rrbracket, \llbracket \mathbf{C}' \rrbracket \leftarrow \mathcal{F}_{RandComb}(\llbracket \mathbf{A} \rrbracket, \llbracket \mathbf{C} \rrbracket, N)$ %N is the number of rows in $\mathbf{X}$

3: $\llbracket \mathbf{Z} \rrbracket \leftarrow \mathcal{F}_{BeaverMul}(\llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{Y} \rrbracket, \llbracket \mathbf{A}' \rrbracket, \llbracket \mathbf{B} \rrbracket, \llbracket \mathbf{C}' \rrbracket)$

CryptGNN uses $\mathcal{F}_{MatMul}$ to execute a linear layer to compute the output feature states $\mathbf{Z}$ from the intermediate feature state $\mathbf{X}$ and the trained parameter matrix $\mathbf{Y}$ of the linear layer. The pre-processing step is executed once for each client to generate the initial Beaver triples. To further reduce communication costs, we can compute and reveal $\mathbf{V}$ once and use the same $\mathbf{V}$ (since $\mathbf{Y}$ and $\mathbf{B}$ are fixed) for subsequent inference requests from the same client.

4.4.2 Secure element-wise multiplication

Several types of FTLs require support for secure element-wise multiplication $[\![Z]\!] = [\![X]\!] \times [\![Y]\!]$, where neither X nor Y are assumed to be constant. In A-SS, we can compute the result of multiplication using a Beaver triple as discussed in Section 4.1. The maximum number of element-wise multiplications required for a single GNN inference request is predetermined by the specific model architecture. One approach is to pre-compute this specific number of Beaver triples and employ them during inference. However, as outlined in the preceding subsection, we should not use the same triple for multiple inference requests due to the potential risk of information leakage. Therefore, we introduce $\mathcal{F}_{ElemMul}$ to perform element-wise multiplications within the GNN layers by generating a fresh set of Beaver triples for each inference request. The steps followed in $\mathcal{F}_{ElemMul}$ are described as below:

1. $\mathcal{F}_{MsAsPair}$: At the pre-processing stage, the parties generate a set of numbers both in A-SS and M-SS format. (Algorithm 4 Offline Phase) which will be used in Step 3.
2. $\mathcal{F}_{BeaverM}$: The parties generate the triples $(\langle\!\langle A \rangle\!\rangle, \langle\!\langle B \rangle\!\rangle, \langle\!\langle C \rangle\!\rangle)$ in the multiplicative format (Algorithm 4 Online Phase Line 1).
3. $\mathcal{F}_{BeaverMtoA}$: The parties communicate with each other and use the data generated in $\mathcal{F}_{MsAsPair}$ to convert the Beaver triple from multiplicative to additive format as $([\![A]\!], [\![B]\!], [\![C]\!])$ (Algorithm 4 Online Phase Line 2).
4. The parties use the Beaver triples $([\![A]\!], [\![B]\!], [\![C]\!])$ to compute $[\![X]\!] \times [\![Y]\!]$ following the steps in Section 4.1 (Algorithm 4 Online Phase Line 3).

Algorithm 4 Element-wise multiplication: $\mathcal{F}_{ElemMul}$

Input: $[\![X]\!], [\![Y]\!]$

Output: Compute $[\![Z]\!] = [\![X]\!] \times [\![Y]\!]$

Offline Phase, $\mathcal{F}_{MsAsPair}$

1: Generates a list **AM**, where i-th ($i \in |\mathbf{AM}|$) element of **AM** is a pair ($[\![R_i]\!]$, $\langle\!\langle R_i \rangle\!\rangle$) representing additive and multiplicative share of value a random value R_i.

Online Phase

1: Generate Beaver triples ($\langle\!\langle A \rangle\!\rangle, \langle\!\langle B \rangle\!\rangle, \langle\!\langle C \rangle\!\rangle$) using $\mathcal{F}_{BeaverM}$.

2: $([\![A]\!], [\![B]\!], [\![C]\!]) \leftarrow \mathcal{F}_{BeaverMtoA}(\langle\!\langle A \rangle\!\rangle, \langle\!\langle B \rangle\!\rangle, \langle\!\langle C \rangle\!\rangle)$ using three pairs from **AM**.

3: $[\![Z]\!] \leftarrow \mathcal{F}_{BeaverMul}([\![X]\!], [\![Y]\!], [\![A]\!], [\![B]\!], [\![C]\!])$.

Next, we describe the $\mathcal{F}_{MsAsPair}$, $\mathcal{F}_{BeaverM}$ and $\mathcal{F}_{BeaverMtoA}$ in more detail.

Generating additive-multiplicative pair: $\mathcal{F}_{MsAsPair}$. At the pre-processing stage, $\mathcal{F}_{ElemMul}$ generates a list of pairs **AM** for each client, where i-th ($i \in |\mathbf{AM}|$) element of **AM** is a pair ($[\![R_i]\!], \langle\!\langle R_i \rangle\!\rangle$), i.e., a random value R_i in A-SS and M-SS. The size of the list, $k = |\mathbf{AM}|$ depends on the maximum number of element-wise multiplications required for a GNN inference. If the total number of multiplications is m, then $k = 3 \times m$, since each multiplication operation uses three pairs from **AM** in $\mathcal{F}_{BeaverMtoA}$. To protect the relative changes in values between two different inference requests from the same client, it is necessary to use a different set of pairs for multiplication when processing an element at the same index of a GNN layer. This can be done by shifting the elements of **AM** by three for each inference. In this way, $\mathcal{F}_{ElemMul}$ can support m inference requests from the same client.

To generate a number in A-SS and M-SS in $\mathcal{P}$ parties, $\mathcal{F}_{MsAsPair}$ extends the algorithm proposed by Xiong et al. [204], which works only for two parties, to make it work for any number of parties. In this approach, two parties CP_1 and CP_2 generates random numbers X_1 and X_2, and convert $(X_1 + X_2)$ to the multiplicative secret-shared M-SS format $\langle\!\langle X_1 + X_2 \rangle\!\rangle$, without revealing X_1 and X_2 to each other.

Following this approach, $\mathcal{F}_{MsAsPair}$ can compute $[\![R_i]\!]$ and $\langle\!\langle R_i \rangle\!\rangle$ for $i \in [1, .., P-1]$, where $R_i = X_i + X_{i+1}$. Here, the value X_i is known to i-th party only. Since, $\langle\!\langle R_i \rangle\!\rangle$ is in multiplicative format, each party can compute $\langle\!\langle R \rangle\!\rangle = \prod_{i=1}^{P-1} \langle\!\langle R_i \rangle\!\rangle$ locally. The parties can compute the additive share of $[\![R]\!] = \prod_{i=1}^{P-1} [\![R_i]\!]$ using the Beaver triples generated using a state-of-the-art technique [203]. Thus, $\mathcal{F}_{MsAsPair}$ can generate a pair ($[\![R]\!]$, $\langle\!\langle R \rangle\!\rangle$), the A-SS and M-SS version of the same value R.

Algorithm 5 presents the following steps required to generate a pair ($[\![R]\!], \langle\!\langle R \rangle\!\rangle$)) for a random value R.

1. Follows two-party protocol [204] for i and $(i+1)$-th parties $i \in [1, \cdots, P-1]$ to generate a pair of containing additive and multiplicative shares of a random value. Set 0 for additive share and 1 for multiplicative share for the parties that are not involved in the computation. (Lines 1 - 5). In this way, we get $P-1$ pairs.
2. Compute the product of values in additive shares in all pairs to generate a random value R in A-SS (Line 6).
3. Compute the product of multiplicative shares in all pairs to generate the same random value R in M-SS (Line 7).

Following this approach, we can generate k pairs to prepare the list **AM** within the same communication round.

Generating Beaver triples in multiplicative format: $\mathcal{F}_{BeaverM}$. Each party CP_i generates two random variables A_i and B_i, and computes $C_i = A_i \times B_i$. In this way, A_i, B_i and C_i constitute the A, B and C in multiplicative share format. Since $C = \prod_{i=1}^{P} C_i = \prod_{i=1}^{P} A_i \times B_i = \prod_{i=1}^{P} A_i \times \prod_{i=1}^{P} B_i = A \times B$, the parties can generate Beaver triples in the multiplicative format without any communication.

Converting Beaver triples in additive format: $\mathcal{F}_{BeaverMtoA}$. $\mathcal{F}_{ElemMul}$ converts the Beaver triples from M-SS to A-SS using $\mathcal{F}_{BeaverMtoA}$, which internally calls $\mathcal{F}_{MtoA}$ to convert each value in the triples. To convert a value U from M-SS format $\langle\!\langle W \rangle\!\rangle$ to A-SS format $[\![W]\!]$, $\mathcal{F}_{MtoA}$ selects a pair ($[\![R]\!]$, $\langle\!\langle R \rangle\!\rangle$) from pre-computed list **AM**, where R is in A-SS and M-SS format as $[\![R]\!]$ and $\langle\!\langle R \rangle\!\rangle$, respectively. Then, each party

Algorithm 5 Generate additive-multiplicative pair, $\mathcal{F}_{MsAsPair}$

Output: Generate additive share $[\![R]\!]$ and multiplicative share $\langle\!\langle R \rangle\!\rangle$ of a random value R

1: **for** $i \leftarrow 1$ to $P-1$ **do**

2: CP_i and CP_{i+1} generates two random values X_i and X_{i+1}. Thus, a $R_i = X_i + X_{i+1}$ is generated in A-SS format $[\![R_i]\!]$.

3: Two parties communicates to generate multiplicative shares $\langle\!\langle R_i \rangle\!\rangle \leftarrow \langle\!\langle X_i + X_{i+1} \rangle\!\rangle$ using [204]

4: $[\![R_i]\!]_j \leftarrow 0$ and $\langle\!\langle R_i \rangle\!\rangle_j \leftarrow 1$, for $j \neq i$, $j \neq (i+1)$

5: **end for**

6: Compute $[\![R]\!] \leftarrow \prod_{i=1}^{P-1} [\![R_i]\!]$ using Beaver triples generated using [203], thereby each party gets a additive share of R.

7: Each party p computes locally $\langle\!\langle R \rangle\!\rangle_p \leftarrow \prod_{i=1}^{P-1} \langle\!\langle R_i \rangle\!\rangle_p$, thereby computing the multiplicative shares of R.

computes locally and communicates with each other to reveal the ratio (α) of W and R. Next, each party uses α and $[\![R]\!]$ to compute the additive share of W as $[\![W]\!]$.

To convert Beaver triples from M-SS to A-SS format $\mathcal{F}_{BeaverMtoA}$ internally uses $\mathcal{F}_{MtoA}$, which invokes the following steps to convert a value W from M-SS to A-SS format.

1. Select a pair ($[\![R]\!]$, $\langle\!\langle R \rangle\!\rangle$) from **AM** (Algorithm 6 Line 1).
2. Apply Extended Euclidean Algorithm [205] to compute the inverse of R as $\langle\!\langle R^{-1} \rangle\!\rangle$ in M-SS format [206] (Algorithm 6 Line 2).
3. Each party computes locally the product of $\langle\!\langle W \rangle\!\rangle$ and $\langle\!\langle R^{-1} \rangle\!\rangle$ (Algorithm 6 Line 3) and reveals the ratio α (Algorithm 6 Line 4).
4. Each party computes $[\![W]\!]_i \leftarrow \alpha \times [\![R]\!]_i$ to get W in A-SS format (Algorithm 6 Line 5).

$\mathcal{F}_{BeaverMtoA}$ converts each value of Beaver triple from M-SS format using $\mathcal{F}_{MtoA}$ as shown in Algorithm 7.

Algorithm 6 Multiplicative to Additive Shares, $\mathcal{F}_{MtoA}$

Input: $\langle\langle W \rangle\rangle$

Output: Generate additive shares $[\![W]\!]$

1: Pick a pair $([\![R]\!], \langle\langle R \rangle\rangle)$ from **AM** computed in $\mathcal{F}_{MsAsPair}$

2: Each party p computes locally $\langle\langle R^{-1} \rangle\rangle_p$ using Extended Euclidean Algorithm, thereby computing the multiplicative shares of R^{-1}.

3: CP_i computes $\langle\langle \alpha \rangle\rangle_i \leftarrow \langle\langle W \rangle\rangle_i \times \langle\langle R^{-1} \rangle\rangle_i$.

4: All parties collaboratively recover α.

5: CP_i computes $[\![W]\!]_i \leftarrow \alpha \times [\![R]\!]_i$

Algorithm 7 Generate Beaver Triples, $\mathcal{F}_{BeaverMtoA}$

Input:$(\langle\langle A \rangle\rangle, \langle\langle B \rangle\rangle, \langle\langle C \rangle\rangle)$

Output: Beaver Triples in A-SS format $([\![A]\!], [\![B]\!], [\![C]\!])$

1: $[\![A]\!] \leftarrow \mathcal{F}_{MtoA}(\langle\langle A \rangle\rangle)$

2: $[\![B]\!] \leftarrow \mathcal{F}_{MtoA}(\langle\langle B \rangle\rangle)$

3: $[\![C]\!] \leftarrow \mathcal{F}_{MtoA}(\langle\langle C \rangle\rangle)$

4.5 Security and System Analysis

This section presents the security analysis, including theorem proofs, as well as the correctness and overhead analysis of CryptGNN.

4.5.1 Correctness analysis

In CryptGNN, during the execution of secure protocols in inference, parties mask their shares with random noise to protect the data from adversaries. The protocols also guarantee the correctness of their results by eliminating the effect of noise. In this section, we provide the correctness analysis of the protocols in CryptGNN.

Correctness of message-passing layer using CryptMPL. First, we analyze the correctness of CryptMPL without data masking in a P-party SMPC setting. As discussed in Section 4.3, to access the feature vector at the source index of

the feature matrix, each share of the matrix is rotated $\sum_{i=1}^{P} r_i$ times, and $\mathcal{F}_{SR}$ accesses the feature vector at index $\sum_{i=1}^{P}([\![S]\!]_i + r_i) = S + \sum_{i=1}^{P} r_i$ (equivalent to accessing the value at index S in the original feature matrix). To update the feature at the destination index, $\mathcal{F}_{SW}$ writes the intermediate vector $[\![\mathbf{Y}]\!]$ at index 0 of the matrix $[\![\mathbf{G}]\!]$, and it is rotated overall by $\sum_{i=1}^{P} D_i = D$. Thus, $[\![\mathbf{Y}]\!]$ reaches the destination index D of $[\![\mathbf{G}]\!]$, while the vectors in the other indices are 0. Then, in $\mathcal{F}_{SA}$ each party updates its share of the output feature matrix $\mathbf{A}^*$ with $\mathbf{G}$, which is equivalent to updating the feature vector at index D. CryptMPL executes the same operation for all edges, completing the message passing correctly.

Next, to protect the data exchanged with other parties, each party masks the feature matrix and intermediate results with random noise, such that the final feature matrix equals $\mathbf{A}^* + \boldsymbol{\xi}'$, where $\boldsymbol{\xi}'$ is the error due to the added noise. As the same operations are performed at the client side on the input noise matrices, the client can calculate the effect of the overall noise $\boldsymbol{\xi}^* = \boldsymbol{\xi}'$ and share it with all the SMPC parties. Therefore, the noise can be removed to retrieve the correct feature matrix $\mathbf{A}^*$.

Using batching, CryptMPL reads the feature vector at the first index of a batch, and the feature vectors at the relative indices for the batch. Similarly, CryptMPL updates the intermediate matrix $\mathbf{G}$ at the first index of a batch and the indices relative to the first index. Since the first index of a batch can be processed correctly, reads and writes at relative indices give the correct result.

Correctness of feature transformation layers using CryptMUL. To compute FTLs, CryptGNN uses standard techniques as described in Section 4.1. In CryptMUL, we propose new techniques to generate Beaver triples for secure matrix multiplication and element-wise multiplications in additive secret-shared format (A-SS), which are used to implement the secure versions of the FTLs. Other operations in FTLs are straightforward and can be used without requiring any specialized protocol. Here, we demonstrate the correctness of the Beaver triples

generated in $\mathcal{F}_{MatMul}$ and $\mathcal{F}_{ElemMul}$ for secure matrix multiplication and element-wise multiplications respectively.

In $\mathcal{F}_{MatMul}$, $\mathcal{F}_{InitBeaver}$ follows [203] to generate the initial Beaver triples in A-SS format as ($[\![\mathbf{A}]\!]$, $[\![\mathbf{B}]\!]$, $[\![\mathbf{C}]\!]$), where $\mathbf{A} \in \mathbb{R}^{N\times K}, \mathbf{B} \in \mathbb{R}^{K\times K'}, \mathbf{C} \in \mathbb{R}^{N\times K'}$ and $\mathbf{A} \otimes \mathbf{B} = \mathbf{C}$. Next, the parties compute the linear combinations of the rows in the matrix using $\mathcal{F}_{RandComb}$ to generate two new matrices $\mathbf{A}'$ and $\mathbf{C}'$, where $[\![\mathbf{A}'[j]]\!] = \sum_{i=1}^{N} k_{ji} \times [\![\mathbf{A}[i]]\!]$ and $[\![\mathbf{C}'[j]]\!] = \sum_{i=1}^{N} k_{ji} \times [\![\mathbf{C}[i]]\!]$ for $i \in [1, N], j \in [1, N]$. The random real values k_{ji} are generated by a pseudo-random function (PRF). Since, $\mathbf{B}$ is fixed, modifying $\mathbf{A}$ to $\mathbf{A}'$ and $\mathbf{C}$ to $\mathbf{C}'$ using the same linear combination maintains the correctness for $(\mathbf{A}', \mathbf{B}, \mathbf{C}')$.

In $\mathcal{F}_{ElemMul}$, $\mathcal{F}_{MsAsPair}$ follows [204] to compute the A-SS and M-SS of a random value R_i for two parties. For other parties, we consider the share of $[\![R_i]\!] = 0$ and $\langle\!\langle R_i \rangle\!\rangle = 1$, thus the R_i is correct in A-SS and M-SS format for P parties. Following this approach, we generate $P-1$ random values $R_i, i \in [1, P-1]$ in A-SS and M-SS format. Finally, $\mathcal{F}_{MsAsPair}$ takes the products of $[\![R_i]\!] = \prod_{i=1}^{P-1} [\![R_i]\!]$ using the pre-computed Beaver triples (generated using [203]) to compute R in A-SS format. It also calculates the products of $\langle\!\langle R_i \rangle\!\rangle = \prod_{i=1}^{P-1} \langle\!\langle R_i \rangle\!\rangle$ to get the multiplicative share of $\langle\!\langle R \rangle\!\rangle$. Thus, $\mathcal{F}_{MsAsPair}$ gets a random value R which is correct in both A-SS and M-SS format. For each element-wise multiplication operation, $\mathcal{F}_{MsAsPair}$ generates three random values R_A, R_B, R_C in A-SS and M-SS format following this approach.

As part of $\mathcal{F}_{ElemMul}$, $\mathcal{F}_{BeaverM}$ generates a Beaver triple (A, B, C) in M-SS format. Each party CP_p generates random values $\langle\!\langle A \rangle\!\rangle_p$ and $\langle\!\langle B \rangle\!\rangle_p$, and computes $\langle\!\langle C \rangle\!\rangle_p = \langle\!\langle A \rangle\!\rangle_p \times \langle\!\langle B \rangle\!\rangle_p$. In this way, we get the correct Beaver triples in the M-SS format as ($\langle\!\langle A \rangle\!\rangle$, $\langle\!\langle B \rangle\!\rangle$, $\langle\!\langle C \rangle\!\rangle$), since $\langle\!\langle C \rangle\!\rangle = \prod_{i=1}^{P} \langle\!\langle C \rangle\!\rangle_i = \prod_{i=1}^{P} \langle\!\langle A \rangle\!\rangle_i \times \langle\!\langle B \rangle\!\rangle_i = \prod_{i=1}^{P} \langle\!\langle A \rangle\!\rangle_i \times \prod_{i=1}^{P} \langle\!\langle B \rangle\!\rangle_i = \langle\!\langle A \rangle\!\rangle \times \langle\!\langle B \rangle\!\rangle$.

Finally, $\mathcal{F}_{BeaverMtoA}$ converts each element of ($\langle\!\langle A \rangle\!\rangle$, $\langle\!\langle B \rangle\!\rangle$, $\langle\!\langle C \rangle\!\rangle$) to A-SS format ($[\![A]\!]$, $[\![B]\!]$, $[\![C]\!]$) using the R_A, R_B, R_C in A-SS and M-SS format following the

approach in STR [207], which is proved to be correct. Thus, $\mathcal{F}_{ElemMul}$ gets a correct Beaver triple as the required format ($[\![A]\!]$, $[\![B]\!]$, $[\![C]\!]$), which can be used for the secure element-wise multiplication operation.

4.5.2 Security analysis

This section proves the protocol security throughout the execution of CryptGNN, which preserve the privacy of the client's (DO's) input graph and the model owner's (MO's) model parameters against the threat model TM. CryptGNN follows the standard A-SS approach to store model parameters and uploads graph data to the computing parties. Thus, data at rest (model parameters, feature matrix, source and destination index of each edge) are information-theoretically secure [195] against adversaries following Axiom 1.

Axiom 1. A value x is information-theoretically secure in A-SS format even if $P-1$ out of P parties collude.

To provide the security analysis in a structured way, we consider three different cases within the threat model TM, (a) TM_P: At most $P-1$ parties may collude to learn DO's input graph data or MO's GNN model parameters (but they do not collude with DOs or MO), (b) TM_M: The colluding parties in TM_P may collude with MO to gain access to DO's private graph data, (c) TM_D: The colluding parties in TM_P may collude with a data owner DO_{fake} to learn MO's GNN model parameters or the input graph the other DOs. To prove the security of our protocols against TM_P, TM_M and TM_D, we adopt the following security definitions [208]:

Definition 1. Let parties $CP_1, \cdots, CP_P$ engage in a protocol π that computes function $\mathcal{F}(in_1, \cdots, in_P) = (out_1, \cdots, out_P)$, where in_i and out_i denote the input and output of party CP_i, respectively. Let, $VIEW_\pi(CP_i)$ denote the view of participant CP_i during the execution of protocol π. More precisely, CP_i's view is formed by its input, internal random coin tosses r_i, pseudo-random values pr_i, as

well as messages $m_1, \cdots, m_k$ passed between the parties during protocol execution: $VIEW_\pi(CP_i) = (in_i, r_i, pr_i, m_1, \cdots, m_k)$. Let I denote a subset of at most $P-1$ parties that collude in our threat model TM_P. $VIEW_\pi(I)$ denote the combined view of participants in I during the execution of protocol π (i.e., the union of the views of the parties in I), and $\mathcal{F}_I(in_1, \cdots, in_P)$ denote the projection of $\mathcal{F}(in_1, \cdots, in_P)$ on the coordinates in I (i.e., $\mathcal{F}_I(in_1, \cdots, in_P)$ consists of the output of function $\mathcal{F}$ of the colluding parties). We say that the protocol π is secure against TM_P if for each coalition of size at most $P-1$ there exist a probabilistic polynomial time (PPT) simulator S_I such that $\{S_I(in_I, \mathcal{F}_I(in_1, \cdots, in_P)), \mathcal{F}(in_1, \cdots, in_P)\} \equiv \{VIEW_\pi(I), (out_1, \cdots, out_P)\}$, where $in_I = \bigcup_{CP_i \in I} in_i$ and $\equiv$ denotes computational or statistical indistinguishability..

Definition 2. In the case of TM_M, we consider the model parameters Θ are also known to the colluding parties I, and the protocol π is secure against TM_M if there exists a probabilistic polynomial time (PPT) simulator S_I such that $\{S_I(in_I, \Theta, \mathcal{F}_I(in_1, \cdots, in_P)), \mathcal{F}(in_1, \cdots, in_P)\} \equiv \{VIEW_\pi(I), (out_1, \cdots, out_P)\}$, where $in_I = \bigcup_{CP_i \in I} in_i$.

Definition 3. In the case of TM_D, a data owner DO_{fake}'s private input $(\mathbf{X}_f, \mathbf{S}_f, \mathbf{D}_f, \xi_f)$ is known to the colluding parties, and the protocol π is secure against TM_D, if there exists a probabilistic polynomial time (PPT) simulator S_I such that $\{S_I(in_I, \mathbf{X}_f, \mathbf{S}_f, \mathbf{D}_f, \xi_f, \mathcal{F}_I(in_1, \cdots, in_P)), \mathcal{F}(in_1, \cdots, in_P)\} \equiv \{VIEW_\pi(I), (out_1, \cdots, out_P)\}$, where $in_I = \bigcup_{CP_i \in I} in_i$.

The following theorems ensure protocol security throughout the execution of CryptGNN.

Theorem 1. The node features are protected against TM in CryptMPL.

Proof. To execute the message-passing, CryptMPL executes secure read, write and aggregation protocols for each edge of the input graph data stored in A-SS format

as $([\![\mathbf{X}]\!], [\![\mathbf{S}]\!], [\![\mathbf{D}]\!])$. To prove this theorem we consider that the simulator $S_{CryptMPL}$ calls the simulators S_{SR}, S_{SW} and S_{SA} of $\mathcal{F}_{SR}$, $\mathcal{F}_{SW}$ and $\mathcal{F}_{SA}$ respectively.

In the secure read protocol with data masking $\mathcal{F}_{SR}$, each party CP_i exchanges the share of feature matrix $[\![\mathbf{A}]\!]_i$ with other parties. We consider the case where the parties do not rotate the feature matrix to prove this theorem. In this case, a party has all the shares of the feature matrix and can reconstruct the original feature matrix by taking the sum of the shares as, $\mathbf{A} = \sum_{p=1}^{P} [\![\mathbf{A}]\!]_p$. To protect the feature matrix, each party CP_i masks data with noise matrix $\boldsymbol{\xi}_i$ and shares $[\![\mathbf{A}_\xi]\!]_i = [\![\mathbf{A}]\!]_i + \boldsymbol{\xi}_i$ with other parties.

As in Definition 1, I denotes the set of at most $P-1$ parties that collude in our threat models. We build a simulator S_{SR}, which simulates the view of parties in I. In the simulated view, S_{SR} can compute $\mathbf{A}_\xi = \sum_{p=1}^{P} [\![\mathbf{A}_\xi]\!]_p = \sum_{p=1}^{P} [\![\mathbf{A}]\!]_p + \sum_{p=1}^{P} \boldsymbol{\xi}_p$, as it has all the masked shares $[\![\mathbf{A}_\xi]\!]_i$ for $i \in [1, \cdots, P]$. However, $\mathbf{A}_\xi$ is uniformly random in I's View, since there is at least one mask matrix $\boldsymbol{\xi}_i$ (from the non-colluding party) which is unknown to I. Therefore the distribution over the real $\mathbf{A}_\xi$ received by the colluding parties and over the simulated $\mathbf{A}_\xi$ generated by the simulator is identically distributed.

Similarly, during the secure write operation, the share of the temporary matrix $[\![\mathbf{G}]\!]_i$ from each party i is masked with a mask matrix. In the simulated view of $\mathcal{F}_{SW}$, I can compute $\mathbf{G}_\xi = \sum_{p=1}^{P} [\![\mathbf{G}_\xi]\!]_p = \sum_{p=1}^{P} [\![\mathbf{G}]\!]_p + \sum_{p=1}^{P} \boldsymbol{\xi}_p$, which is uniformly random in I's View, since there is at least one mask matrix $\boldsymbol{\xi}_i$ (from non-colluding party) which is unknown to I. Therefore the distribution over the real $\mathbf{G}_\xi$ received by the colluding parties and over the simulated $\mathbf{G}_\xi$ generated by the simulator is identically distributed.

The secure aggregation operation $\mathcal{F}_{SA}$ does not require exchanging data with other parties, as the addition operations on additive secret-shared data can be executed locally. Since the views produced by the simulator $S_{CryptMPL}$ in the read

and write protocols are indistinguishable from the parties' views in the real protocol execution, the views remain indistinguishable after the simulation of S_{SA}, which proves that the input and output node features of CryptMPL are secure against TM_P, while processing an edge. CryptMPL follows the same procedure to process all the edges. Since the collective view of the protocol execution while processing each edge is computationally indistinguishable from a simulated view, the node features are protected in $\mathcal{F}_{CryptMPL}$. Finally, each party locally subtracts the noise from their share of the result. Since no data is exchanged in this step, the node features remain protected. Processing multiple edges in a batch has no additional impact on the node features, ensuring their security during batching.

The node features are also protected against TM_M, since the model parameters Θ are not involved in any step of $\mathcal{F}_{CryptMPL}$. In the case of TM_D, the input of the data owner DO_{fake} is known to the colluding parties I. However, since each client generates its own mask matrix, knowing the DO_{fake}'s data does not reveal other client's private input. Thus, for each client, the views of I produced by the simulator $S_{CryptMPL}$ remain indistinguishable from the parties' views in the real protocol execution, which proves that the input and output node features of CryptMPL are secure against TM_D.

Since, the node features are secured against TM_P, TM_M, and TM_D, thereby protected against the threat model TM.

Theorem 2. The graph structure is secured against TM in CryptMPL.

Proof. To prove this theorem, first, we analyze the protocols followed to process an edge in CryptMPL. During the read operation, each party CP_i shifts the share of the source node index $[\![S]\!]_i$ by a random amount r_i. Thus, the simulator S_{SR} of $\mathcal{F}_{SR}$ gets an aggregated value as $S' = \sum_{p=1}^{P} [\![S]\!]_p + \sum_{p=1}^{P} r_p$. Since at least one r_i from the i-th party is unknown to the view of S_{SR}, the source index is uniformly random in the I's view. The destination index for an edge is also protected in the view of the simulator S_{SW} of $\mathcal{F}_{SW}$, since the parties use the share of the destination index to rotate the

intermediate matrix locally and do not share the destination index during the write operation.

Since both source and destination indices are protected, if the colluding parties I try to estimate the source-destination pair, the probability of correct estimation is $\frac{1}{N\times(N-1)}$, where N is the number of nodes in the graph. Processing multiple edges in CryptMPL does not reveal additional information. Since CryptMPL rotates the matrices by different amounts and uses different noise matrices for each edge, an adversary can not learn anything from the access pattern. Similar to processing an edge in each round, the probability of correct estimation of source-destination pairs is $\frac{1}{N\times(N-1)}$ for a batch in case of batch processing. Thus, to process all edges in R batches in CryptMPL, the probability of correct reconstruction of graph structure is N^{-2R}. Therefore, increasing the number of batches ensures stronger security against TM_P.

Similar to the logic described in Theorem 1, the graph structure is also protected against TM_M and TM_D, thereby it is protected against TM.

Lemma 1. Let A and B be two secrets encrypted using A-SS in a P-party SMPC setting, represented as $[\![A]\!]$ and $[\![B]\!]$, respectively. Let the linear combination of $[\![A]\!]$ and $[\![B]\!]$ be $[\![C]\!] = a \cdot [\![A]\!] + b \cdot [\![B]\!]$, where a and b are public coefficients. The shares of C preserve the information-theoretic security of the original secrets against TM.

Proof. We analyze the combined view of the participants in I as defined in Defintion 1. We build a simulator S_{LC}, which simulates the view of parties in I. In the simulated view, S_{LC} can compute $C = a \cdot \sum_{i=1}^{P} [\![A]\!] + b \cdot \sum_{i=1}^{P} [\![B]\!]$. However, since at least one share of A and B is unknown to I, the distribution over the real C received by the colluding parties and the simulated C generated by the simulator is identically distributed.

Theorem 3. DO's input graph and MO's model parameters are secured in CryptGNN against TM.

Proof. Considering the scenario where each client may upload data for numerous inference requests, to prove the security of DO's input graph and MO's model parameters in CryptGNN, we demonstrate that the protocols employed to generate a fresh set of Beaver triples for matrix multiplication ($\mathcal{F}_{MatMul}$) and element-wise multiplication ($\mathcal{F}_{ElemMul}$) operations in FTLs are secure.

$\mathcal{F}_{MatMul}$ is secure against TM. To prove $\mathcal{F}_{MatMul}$ is secure, we consider a simulator S_{MatMul} that uses the simulators $S_{InitBeaver}$ and $S_{RandComb}$ of $\mathcal{F}_{InitBeaver}$ and $\mathcal{F}_{RandComb}$ respectively.

In $\mathcal{S}_{MatMul}$, $S_{InitBeaver}$ generates the initial Beaver triples. We refer to [203] for the security proof that shows $\mathcal{F}_{InitBeaver}$ is secure against TM_P. Each element of the generated triple, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ is secure in additive secret-shared format according to Axiom 1.

To compute $[\![\mathbf{X}]\!] \otimes [\![\mathbf{Y}]\!]$, $S_{RandComb}$ modifies $\mathbf{A}$ and $\mathbf{C}$ using the same linear combination of the rows (Step 2 in Subsection 4.4.1). Although, the combination of the rows used to generate $\mathbf{A}'$ and $\mathbf{C}'$ is known to the view of $S_{RandComb}$, since the elements in each row involved in the computation are in the secret-shared domain, $\mathbf{A}'$ and $\mathbf{C}'$ remain secure against the threat models (as shown in Lemma 1).

Following Step 3 of $\mathcal{F}_{MatMul}$, the matrices $\mathbf{U} = \mathbf{X} - \mathbf{A}'$ and $\mathbf{V} = \mathbf{Y} - \mathbf{B}$ are revealed to S_{MatMul}. Since, $\mathbf{A}'$ and $\mathbf{B}$ are unknown to the simulator S_{MatMul}, the distribution over elements in the private inputs $\mathbf{X}$ and $\mathbf{Y}$ remain identically distributed. $\mathcal{F}_{MatMul}$ uses a newly generated matrix $\mathbf{A}'$ for each inference request. Therefore, revealing $\mathbf{U}$ will not reveal the relative changes in the private input $\mathbf{A}$ in two different requests.

Finally, $\mathbf{U}$ and $\mathbf{V}$ are used to compute $[\![\mathbf{Z}]\!] = [\![\mathbf{X}]\!] \otimes [\![\mathbf{Y}]\!]$, which does not involve any data sharing between the computing parties. Therefore, the simulated view is

identical to the real view of I. This proves that $\mathcal{F}_{MatMul}$ is secure against the threat model TM_P.

In the case of TM_M, the model parameters Θ are known, which means the input $\mathbf{Y}$ of the matrix multiplication in the linear layer is known to I. Using $\mathbf{V}$ and $\mathbf{Y}$, the I can learn $\mathbf{B}$. However, since $\mathbf{A}'$ and $\mathbf{C}'$ are protected, the distribution over the private input $\mathbf{X}$ and the output $\mathbf{Z}$ received by I and over the simulated matrices generated by the simulator are identically distributed.

In the threat model TM_D, using DO_{fake}'s private input, the colluding parties I can learn $\mathbf{X}$ and consequently $\mathbf{A}'$ from $\mathbf{U}$ in the inference requests from that data owner. However, $\mathbf{B}$ and $\mathbf{C}'$ are still protected, and the distribution over a linear layer's private input (model parameter) $\mathbf{Y}$ and the output $\mathbf{Z}$ received by I and over the simulated matrices generated by the simulator are identically distributed. Since, $\mathcal{F}_{InitBeaver}$ generates a fresh set of Beaver triples for each client, learning $\mathbf{A}'$ using DO_{fake}'s input by I does not help to learn other DO's private input. Thus, $\mathcal{F}_{MatMul}$ is secure against TM_D.

$\mathcal{F}_{ElemMul}$ is secure against TM. At the pre-processing stage, $\mathcal{F}_{ElemMul}$ generates additive and multiplicative shares of random values for two parties using oblivious transfer. We refer to [204] for the proof of this step. In $\mathcal{F}_{MsAsPair}$, the parties CP_i and CP_{i+1} for $i \in [1, \cdots, P-1]$ generate $P-1$ numbers of R_i values. Then the additive share $[\![R_i]\!]$ are multiplied using Beaver triples generated using secure protocol used in Mascot [203]. The multiplicative shares $\langle\!\langle R_i \rangle\!\rangle$ are used locally to compute the multiplicative shares of R. Since at least one R_i value is unknown to I, the distribution over the real pair ($[\![R]\!]$, $\langle\!\langle R \rangle\!\rangle$) received by I and over the simulated ($[\![R]\!]$, $\langle\!\langle R \rangle\!\rangle$) generated by the simulator is identically distributed.

The simulator $S_{BeaverM}$ of $\mathcal{F}_{BeaverM}$ generates the Beaver triples in multiplicative format, which does not require any communication among the parties. Since the parties in I do not receive any data from other parties, the simulated view is

identical to the real view. To covert multiplicative shares to additive shares, the simulator $S_{BeaverMtoA}$ of $\mathcal{F}_{BeaverMtoA}$ uses the additive-multiplicative pairs generated in $\mathcal{F}_{MsAsPair}$. We refer to the proof of protocol $SecMulResh$ described in STR [207] to prove that $\mathcal{F}_{BeaverMtoA}$ is secure against TM_P. Thus, $\mathcal{F}_{ElemMul}$ is secure, since its sub-protocols are proven to be secure.

To compute the element-wise multiplication in different layers of GNN, we pre-compute the required amount of additive-multiplicative pairs in $\mathcal{F}_{MsAsPair}$. For each inference request, $\mathcal{F}_{BeaverM}$ generates a fresh Beaver triple in multiplicative format and $\mathcal{F}_{BeaverMtoA}$ uses a different pair from $\mathcal{F}_{MsAsPair}$ to compute an element-wise multiplication. Thus, the ratio α recovered in $\mathcal{F}_{BeaverMtoA}$ does not reveal any relative value for two different inference requests.

In the case of TM_M, the simulator may learn B in the Beaver triple from V, if the model parameter is used as Y in the multiplication step and it is known to I. However, the elements A and C are still protected, which are related to the DO's private input and the result of the multiplication. Therefore, the DO's private input and the private output are secure against TM_M.

In the threat model TM_D, using DO_{fake}'s private input, the colluding parties I can learn A of the Beaver triple, but the model parameters and the final result are protected since B and C are unknown. Furthermore, the other DO's data is also protected, since we generate a new set of (additive-multiplicative) shares for each client, which are used to generate the Beaver triples.

4.5.3 Overhead analysis

Overhead analysis of CryptMPL. In this part, we present the overhead analysis of CryptMPL in terms of: (i) the number of nodes in the graph, N, (ii) the number of edges M, (iii) the number of features of each node K, (iv) the number of computing parties P, and (v) the number of batches R to process M edges.

Table 4.1 Performance Comparison of Secure Message-passing

	CryptMPL	**AdjacencyMatrix**
Computation Cost (Client)	$O(N \times K \times P^2 \times R)$	$O(N^2 \times K)$
Computation Cost (Each party)	$O(N \times K \times P \times R)$	$O(N^2 \times K)$
Communication Cost (Client to each CP)	$(N \times K + M \times 2 + P) \times L$	$(N^2 + 2 \times N \times K) \times L$
Communication Cost (Each CP to others)	$(N \times K \times R + M) \times P \times L$	$(N^2 + 2 \times N \times K) \times P \times L$

If we consider both the graph data and the model in plain text, the message passing layer of GNN can be computed by processing the edges as shown in Equation (4.1). If the in-degree of a i-th node is d_i, then there are $\sum_{i=1}^{n} d_i = M$ additions of two vectors of size $(1, K)$ in total. However, this approach is not secure, as both model and graph data are revealed to the computing party.

CryptMPL preserves data privacy, with overhead in terms of computation and communication among servers. It also introduces overhead on the client side, as the client computes a noise matrix at the pre-processing stage and uploads the noise-matrix along with the graph data. The computation cost at the client side is proportional to the size of feature matrix $(N \times K)$, the number of batches (R), and the number of the computing parties (P^2), since it requires to compute the effect of noise added by each party on its own data and on the data of the other parties. To protect the feature matrix and graph structure, CryptMPL adds noise and rotates matrices by a random amount. Overall, there are $2 \times P \times R$ numbers of rotations and $(P + 1) \times R + 1$ numbers of addition of matrices of size (N, K) by each server. Similar to plain text, each party needs to add two $(1, K)$ sized vectors M times. So, the computation overhead with respect to the plain text version is $O(N \times K \times P \times R)$. These computations can be done in a multi-threaded way (or transferred to GPU) to make them faster. Additionally, each server sends $N \times K \times P \times 2 \times R + M$ number of $L = 64$-bit values to other servers in total. To process the batches, CryptMPL transfers all the bits in one round, reducing the propagation and queuing delay [209].

CryptMPL has significantly less overhead than MPL using an adjacency matrix, which requires multiplying two matrices of size (N, N) and (N, K). In the adjacency matrix-based approach, either the client or the trusted server needs to distribute three matrices $\mathbf{A} \in \mathbb{R}^{N \times K}, \mathbf{B} \in \mathbb{R}^{N \times N}, \mathbf{C} \in \mathbb{R}^{N \times K}$ as the Beaver triples to P parties to support secure multiplication in secret-shared domain following [128].

We summarize the computation and communication cost at the client and server side for different approaches in Table 4.1. Here, L is the number of bits required to represent each value.

Overhead analysis of CryptMUL. In this part, we present the overhead analysis of the sub-protocols of CryptMUL for one multiplication operation during the inference. We compare the overhead with a trusted server-based approach from CrypTen [11].

The protocol stack of CryptMUL's secure matrix multiplication $\mathcal{F}_{MatMul}$ is shown in Figure 4.8. In $\mathcal{F}_{MatMul}$, to compute $[\![\mathbf{Z}]\!] = [\![\mathbf{X}]\!] \otimes [\![\mathbf{Y}]\!]$, $\mathbf{X} \in \mathbb{R}^{N \times K}, \mathbf{Y} \in \mathbb{R}^{K \times K}, \mathbf{Z} \in \mathbb{R}^{N \times K'}$ during inference, the parties compute locally to generate a new Beaver Triple from the pre-computed Beaver triple. The computation cost in this process is $O(N^2 \times (K + K'))$. This process does not require any communication among the computing parties, while [11] requires one round of communication between the computing parties and the trusted server to get a new Beaver triple. The cost associated with computing matrix-multiplication using the Beaver triple is the same (one round of communication and local computation) in both [11] and $\mathcal{F}_{MatMul}$. Thus, the overall cost of matrix-multiplication using CryptMUL is lower compared to [11], since the communication overhead is reduced through the local computation in $\mathcal{F}_{MatMul}$.

Figure 4.9 shows the protocol stack for CryptMUL's secure element-wise multiplication protocol $\mathcal{F}_{ElemMul}$. To compute $[\![Z]\!] = [\![X]\!] \times [\![Y]\!]$, $\mathcal{F}_{ElemMul}$ requires each party to generate a random Beaver triple $([\![A]\!], [\![B]\!], [\![C]\!])$. The computation

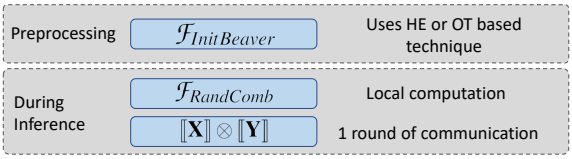


Figure 4.8 Secure matrix multiplication protocol.

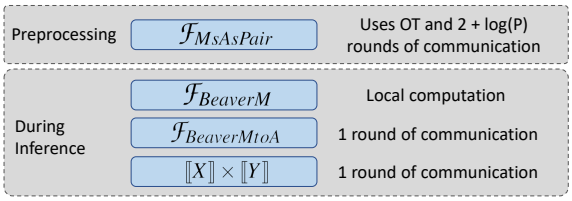


Figure 4.9 Secure element-wise multiplication protocol.

cost of this step is constant $O(1)$. Then, one round of communication among the computing parties is required to convert Beaver triple from multiplicative format to additive format. CrypTen [11] also requires a round of communication between the parties and the trusted server to get the Beaver triple. The cost associated with computing element-wise multiplication using the Beaver triple is the same (one round of communication and local computation) in both CrypTen [11] and $\mathcal{F}_{ElemMul}$. Therefore, the overhead of both CryptMUL and CrypTen is of the same order.

4.6 Evaluation

We implement a CryptGNN prototype in Python and conduct experiments to compare its performance with baselines. To create arithmetic shares of the private data and to implement FTLs, we use CrypTen [11]. In CryptGNN, the model owner does not require any preprocessing, except for encrypting the model parameters in A-SS format, which is a one-time process. We implement CryptMPL with client-side data preprocessing and server-side batching. Additionally, we develop CryptMUL protocols for secure matrix multiplication and element-wise multiplication. To avoid using a trusted party, we replace the multiplication operations in the FTLs of CrypTen using our CryptMUL. We use $L = 64$ bits to represent the values in A-SS format. We perform the experiments on a 3.4GHz Intel Core i7, with the parties running in

Table 4.2 Graph Dataset Statistics

Dataset	#Graphs	#Nodes	#Edges	#Features	#Classes
ENZYMES	600	32.6	124.3	3	6
PROTEINS	1,113	39.1	145.6	3	2
Cora	-	2,708	10,556	1,433	7
CiteSeer	-	3,327	9,104	3,703	6
PPI	20	2,245.3	61,318.4	50	121
FAUST	100	6,890	41,328	3	10

separate processes. We also use AWS instances to evaluate CryptGNN in a realistic distributed cloud setting. We conduct each experiment 30 times and report the average execution time.

Graph datasets. Table 4.2 summarizes the statistics of the graph datasets we use in our experiments. To evaluate the performance of the CryptGNN system for the graph classification task, we use three benchmark datasets: TUDataset (ENZYMES), TUDataset (PROTEINS) [210], and FAUST [211]. Among these three datasets, the FAUST dataset has the largest graphs with 6,890 nodes and 41,328 edges. To assess the performance of the secure message-passing layer CryptMPL, we employed three additional datasets: Cora, CiteSeer [212], and PPI [213]. These benchmark datasets are typically well-suited for node classification tasks and have a substantial number of nodes, edges, and features. We use these datasets to assess the performance of our secure message-passing layer on large graphs. We also generate synthetic data to systematically observe the effects of different parameters.

Network architecture. Figure 4.10 shows the architecture of the GIN [40] model we use to evaluate the performance of secure graph classification tasks. The network comprises three message-passing layers stacked one after another. The outputs of the three layers are concatenated and passed through a linear layer with the ReLU activation function. Finally, another linear layer with the Softmax activation

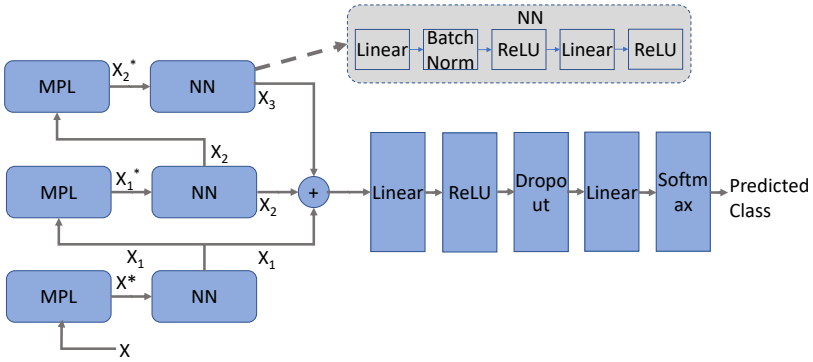


Figure 4.10 GIN architecture.

function predicts the probability of each class for the sample input graph. A dropout layer is used in training, which does not have any effect during inference. The model is trained in private infrastructure and the parameters for linear and batch normalization layers are stored in the cloud in secret-shared format.

4.6.1 Overall CryptGNN performance

We use FAUST, PROTEINS, and ENZYMES datasets. For each dataset, 70% of the graphs are used for training the GIN model; the remaining graphs are considered as the client's private input graphs. In all experiments, we set the batch size to a value that allows us to process all edges in 20 batches, and process all the batches in a single round as described in Subsection 4.3.4.

CryptGNN vs. plain-text performance. We train three GIN models on the three benchmark datasets, and then compare the results of the plain-text versions of the models with the CryptGNN versions. The plain-text version utilizes PyTorch APIs to compute the GNN layers. In CryptGNN, we leverage CrypTen's API, which uses numerical approximations to compute non-linear functions. The necessary multiplication operations are performed using CryptMUL. The fixed-point encoding to represent floating-point values and the approximation techinques may introduce some precision errors in intermediate results. For instance, in a graph with 2000 nodes and 10 features, we observed that the mean difference between the values in plaintext

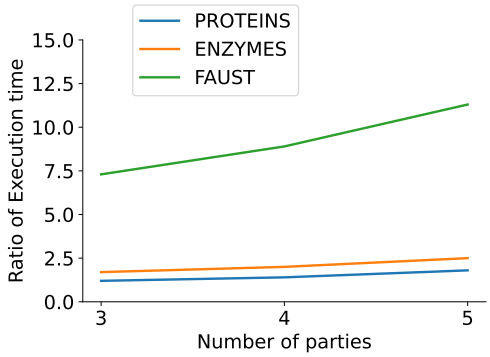


Figure 4.11 Execution time ratio of CrypTen over CryptGNN for GIN, while varying the number of parties.

and the secret-shared domain is 5.1×10^{-5}. This error is acceptable for deep learning tasks. Since our focus is on predicting the classification IDs rather than obtaining the exact float values, the final result remains unaffected. We achieve the same inference accuracy results for each pair of models (i.e., plain-text vs. CryptGNN). This demonstrates that CryptGNN works correctly from a machine learning point of view.

Efficiency. We compare the performance of CryptGNN with an implementation of the GIN model using CrypTen [11]. For CryptGNN, we measure execution time at the server after the data is uploaded by the client. The execution time does not include the server side preprocessing required in CryptMUL, since it is a one time process for each client that generates the initial Beaver Triples. To execute the operations required for GNN inference for the Baseline, we use CrypTen's functions for FTLs and implement an adjacency-matrix based implementation for MPL. Unlike CryptGNN, CrypTen uses a trusted server.

Figure 4.11 shows that CryptGNN is significantly faster than CrypTen, particularly for large-scale graphs in FAUST. This is due to CrypTen incurring higher computation overhead in MPLs and FTLs, and this overhead increases with the number of nodes and features in the graph. As the figure shows, the execution time ratio between CrypTen and CryptGNN also increases with the number of

parties, since CrypTen requires the parties to communicate with the trusted server for many operations. In terms of absolute inference time, the models using CryptGNN work well in practice. For instance, on the FAUST dataset with 6890 nodes and 41328 edges, the CryptGNN model achieved an average inference time of 22.3s in a three-party setting. Furthermore, in order to evaluate scalability, we employed a synthetic graph dataset with an average of 20,000 nodes and 200,000 edges. CryptGNN's average inference time for the graphs in this dataset is approximately 75s. These results show that CryptGNN can work efficiently in practical situations where security matters more than inference latency, such as in drug discovery and automated code analysis. Further reduction in inference latency can be achieved on powerful servers, instead of a laptop as in our experiments.

To evaluate CryptGNN considering network delay and bandwidth restrictions, we performed an experiment, where we used three AWS instances (t2.micro, us-east-1 region) as the computing parties. For the graphs in TUDataset, CryptGNN takes around 2.3 seconds to obtain the inference results for each graph. This indicates that network latency has minimal impact on the overall inference time.

In CryptGNN, the offline pre-processing time on the client side is low compared to the overall execution time. For a benchmark dataset (TUDataset), it takes around 0.1s, with the ratio of offline/online overhead being approximately 1:25.

To evaluate the communication overhead during the online phase, we use two datasets: TUDataset, which consists of smaller graphs with an average of 36 nodes, and a synthetic dataset with larger graphs averaging 20,000 nodes. For both datasets, we measure the communication overhead for each inference request using a trained GIN model with parameters encrypted in a three-party SMPC setting. We report the number of communication rounds, the size of the data (in MB) uploaded by the client, and the total amount of data (in MB) communicated per party using the

Table 4.3 Communication Performance

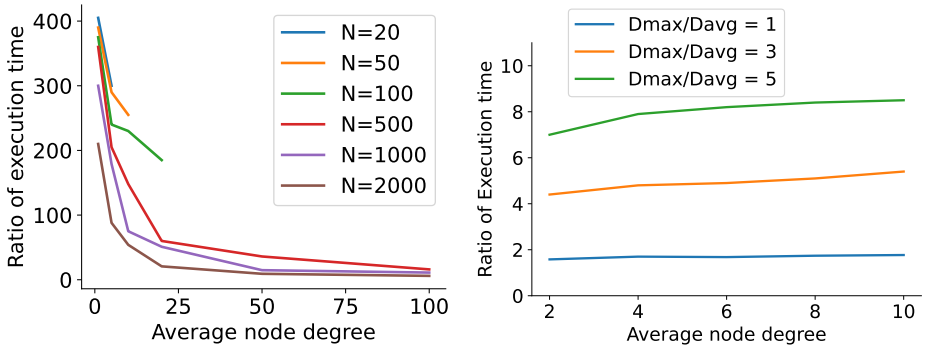
	TUDataset		**Synthetic Dataset**	
	CryptGNN	CrypTen	CryptGNN	CrypTen
Client Comm. (MB)	0.1	0.4	12	3050
Trusted Server Comm. (MB)	-	0.4	-	3050
Each Party Comm. (MB)	1.24	1.31	81.9	6103
Number of Comm. Rounds	120	128	122	130

CryptGNN protocols. Additionally, we compare the communication overhead with CrypTen, which also requires a trusted party during the online phase.

As shown in Table 4.3, CryptGNN achieves significantly lower communication overhead compared to CrypTen on larger datasets. It requires fewer communication rounds and transfers 74 times less data. The high overhead in CrypTen arises from its adjacency matrix-based implementation, which necessitates large matrix multiplications. Additionally, the client uploads substantially less data since there is no need to upload a large adjacency matrix. On smaller datasets, CryptGNN still outperforms CrypTen, reducing communication overhead by 5%, while being more secure. For both datasets, the majority of data is transferred in the message-passing layer, which CryptGNN optimizes using CryptMPL protocols. Furthermore, the majority of communication rounds occur in the non-linear layers due to the element-wise multiplications needed for the numerical approximation of non-linear functions, which can be further optimized through parallelized computations.

4.6.2 CryptMPL results

Overhead. We generate graphs with the numbers of nodes N ranging from 20 to 2000. The average degree D_{avg} is varied from one to $max(100, (N-1)/2)$, having a total number of edges $M = N \times D_{avg}$. We choose a batch size $ceil(M/20)$ to process the edges in 20 batches. Figure 4.12(a) shows the ratio of the execution time of

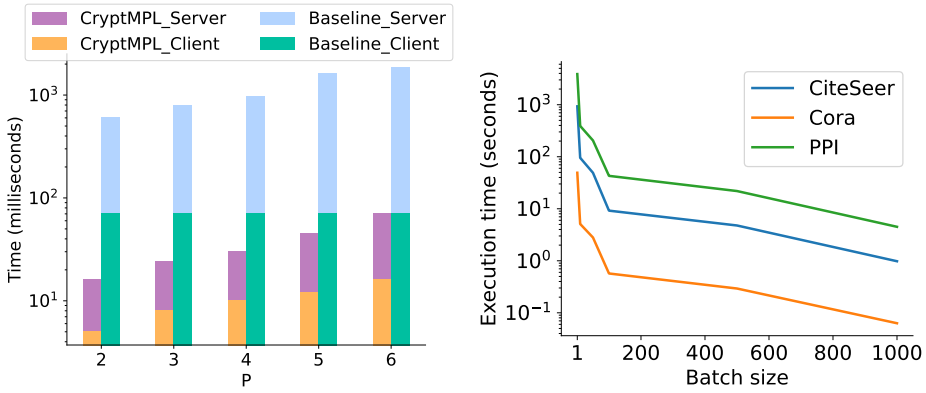


(a) (b)

Figure 4.12 Comparison of CryptMPL with existing techniques: (a) plain text (b) SecGNN.

CryptMPL and the non-secure MPL for different graphs. The results demonstrate that CryptMPL performs efficiently for medium and large-scale graphs, which are the types of graphs encountered in real-life scenarios. As the plain text computation has a linear relation with the number of edges M, the ratio of the execution times decreases as M increases. Despite the high overhead for small graphs, the execution time remains low (e.g., 190ms) and does not have a major impact on the inference latency.

Comparison with SecGNN. This experiment compares the efficiency of CryptMPL with that of SecGNN (see Section 2.3). We measure the ratio of execution time of the MPL between SecGNN and CryptMPL for graphs with $N = 2000$ nodes and $K = 10$ features, while varying the node degrees. Figure 4.12(b) shows that CryptMPL is about 1.5× faster than SecGNN. Moreover, the execution time ratio increases linearly as D_{max}/D_{avg} increases, where D_{max} and D_{avg} are the maximum and average node degree in the graph, respectively. This demonstrates that CryptMPL works better for real-life graphs, as D_{max} is usually much higher compared to D_{avg}. In addition, CryptMPL provides better security than SecGNN, as it works with more than two parties and does not require a trusted party.



(a) (b)

Figure 4.13 Effect of (a) number of parties and (b) batch size on CryptMPL (Y-axis is log scaled).

Comparison with adjacency matrix-based MPL. We compare the execution time of CryptMPL with a hypothetical solution based on representing the graph as an adjacency matrix. The experiment uses a PPI dataset and varies the number of parties. Figure 4.13(a) shows CryptMPL is 25 times faster compared to the adjacency matrix solution when using six parties. This demonstrates that CryptMPL's choice of graph representation and its novel SMPC techniques to compute MPL lead to large performance improvements.

Effect of batching. We use three datasets (Cora, CiteSeer, PPI) in a three-party SMPC setting. Figure 4.13(b) shows that the execution time of CryptMPL decreases as the batch size increases, since CryptMPL requires a low number of rounds $R = \lceil M/B \rceil$ to process the edges, where M and B are the number of edges and the size of each batch, respectively. However, as discussed in Section 4.5, the security guarantees improve exponentially with R and CryptMPL can use a relatively large batch size while still guaranteeing a good level of security.

Scalability test on synthetic data. To evaluate CryptMPL with respect to graph parameters, we use synthetic data and compare the performance with adjacency-matrix based implementation using CrypTen. In each experiment, we vary

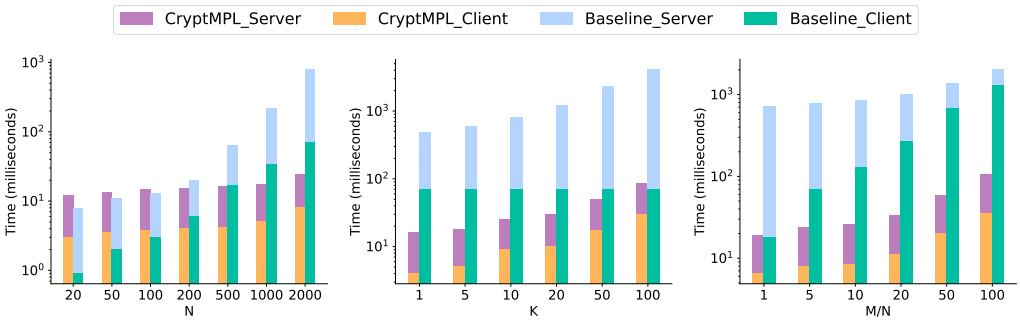
one parameter. Unless otherwise stated, we use three parties in these experiments and the synthetic graph has $N = 2000$ nodes, $K = 10$ features, $D_{avg} = 5$ edges per node.

Effect of the number of nodes. We compare the execution time of CryptMPL with the Baseline using synthetic graphs with different numbers of nodes. Figure 4.14(a) shows the execution times at the client and server are not affected much in CryptMPL. The results show that CryptMPL has superior performance compared to the Baseline for medium and large-scale graphs with many nodes. The reason is that the Baseline approach needs to manage a large adjacency matrix of size (N, N) and performs large matrix multiplications.

Effect of the number of features. We use synthetic graphs with different numbers of features and compare the execution time of CryptMPL with the Baseline. Figure 4.14(b) shows that CryptMPL performs significantly better than the Baseline. Although the size of the adjacency matrix in the Baseline does not change, increasing the number of features increases the number of multiplication operations, which in turn increases the execution time at a higher rate compared to CryptMPL as shown in Figure 4.14(b). As N, P and M are constant, the number of addition and rotation operations remain constant for CryptMPL. However, its execution time increases slightly with K, as addition is now executed on larger matrices.

Effect of the number of edges. Figure 4.14(c) shows the execution time of CryptMPL and the Baseline, when we vary the number of edges. The results show the execution time increases slightly in the case of CryptMPL, while the rate of change is low compared to the Baseline. For the Baseline, the computation and the communication costs remain the same at the server side, as the size of matrices is the same. However, the computation cost at the client side increases to create the adjacency matrix from the list of edges. Increasing the number of edges in a graph increases the computation cost on the client side for CryptMPL, as it needs to process more edges on the noise

matrix to compute the overall effect of noise. Similarly, the server needs to do more computation to execute the MPL on the masked feature matrix.



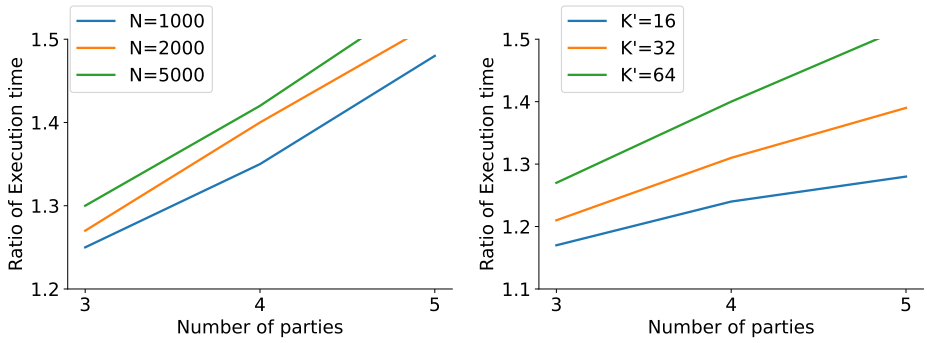
(a) Variable N (b) Variable K (c) Variable D_{avg}

Figure 4.14 Effect of different parameters of graph data (Y-axis is log-scaled).

4.6.3 CryptMUL results

We use CryptMUL to design linear and non-linear layers that work for any number of parties. In this section, we use synthetic data to evaluate the performance of some of these layers and compare the performance with similar layers implemented using CrypTen [11].

Linear layers using CryptMUL. In this experiment, we evaluate the performance of a linear layer that computes $[\![\mathbf{Z}]\!] = [\![\mathbf{X}]\!] \otimes [\![\mathbf{Y}]\!] + [\![\mathbf{B}]\!]$, where $\mathbf{X} \in \mathbb{R}^{N \times K}, \mathbf{Y} \in \mathbb{R}^{K \times K'}, \mathbf{B} \in \mathbb{R}^{N \times K'}$. The linear layer transforms the number of features from K to K' for the same number of nodes N. To see the effect of different parameters of the data, we generate synthetic matrices and vary $N \in [1000, 2000, 5000]$ and $K' \in [16, 32, 64]$. We set $K = 3$ in this experiment. We measure the ratio of the execution time between linear layers using CrypTen and CryptMUL by varying the number of parties $\mathcal{P} \in [3, 4, 5]$. Our results demonstrate that the linear layer implemented with CryptMUL outperforms its counterpart using CrypTen. As illustrated in Figure 4.15, the ratio of execution time between CrypTen and CryptMUL exhibits an increasing trend with $\mathcal{P}$, since the communication overhead in CrypTen increases with the number of parties involved. Moreover, this execution ratio also rises in relation to N and K', since higher



(a) (b)

Figure 4.15 Ratio of execution time of a linear layer between CrypTen and CryptMUL: (a) by varying N and P (b) by varying K' and P.

values of these parameters increase the computation overhead at the trusted server and the communication between the parties with the trusted server. The increase in N or K' raises the computational cost in CryptMUL as well. However, it is worth noting that the addition operations involved in CryptMUL can be parallelized and are generally faster than the matrix-multiplication operations.

Non-linear layers using CryptMUL. In this experiment, we compare the execution time of the Sigmoid function, which requires secure multiplications, between the implementations using CrypTen and CryptMUL's protocol $\mathcal{F}_{ElemMul}$. We generate a list of 1000 random values $\mathbf{X}$ and measure the overall execution time to compute $[\![\mathbf{Z}]\!] = Sigmoid([\![\mathbf{X}]\!])$ for a different number of parties. We observe that the execution time of CrypTen and CryptMUL is almost the same, since both approaches have similar computation and communication costs. However, the protocol using CryptMUL is more secure since it does not require a trusted server as CrypTen.

4.7 Chapter Summary

This chapter presents the system design, analysis, and evaluation of CryptGNN, a provably secure and effective inference system for GNN in MLaaS scenarios. CryptGNN has two main protocols, CryptMPL and CryptMUL, to support secure

MPLs and FTLs in GNN. These novel SMPC protocols preserve the privacy of the model parameters and input graph data while providing the same results as the non-secure inference version. CryptGNN works with an arbitrary number of SMPC parties, and it protects the input data, the intermediate results, and the output, even if $\mathcal{P}-1$ out of $\mathcal{P}$ parties collude. The experimental results demonstrate CryptGNN's correctness and low overhead compared to state-of-the-art approaches.

CHAPTER 5

GOPLACES: AN APP FOR PERSONALIZED INDOOR PLACE PREDICTION

This chapter presents the design and evaluation of an indoor place prediction smart phone app GoPlaces. Compared with existing indoor localization systems, GoPlaces does not require any infrastructure, except for one cheap off-the-shelf WiFi access point that supports ranging with RTT. In addition, it enables personalized place naming and prediction through its on-the-phone data collection and protects users' location privacy because user's data never leaves the phone. In this chapter, Section 5.1 presents the overview of GoPlaces and the problem it attempts to solve. Section 5.2 describes the system architecture of GoPlaces, and Section 5.3 discusses several optimizations in the training procedure. Section 5.4 discusses the experimental results obtained from the prototype implementation. The chapter is summarized in Section 5.5.

5.1 Problem Definition

Our primary objective is to design a mobile app that predicts the place where a user may go in an indoor space, using only one WiFi-RTT AP as infrastructure. Next, we define the concepts required to describe the workflow of GoPlaces and provide a definition for the place prediction task.

5.1.1 Data block

GoPlaces collects user's movement data from phone sensors and WiFi-RTT protocol. Data is collected periodically and stored as a list of *data blocks*, where a block is represented as $db = (ts, w_d, w_{rtt})$. Here ts is the timestamp when db is collected, w_d corresponds to the walking direction of the user (in degrees), which is calculated by

fusing data from the accelerometer and magnetometer, and w_{rtt} is the WiFi-RTT distance in *millimeters* from the AP to the user's position.

5.1.2 Trajectory and segment

When the user walks between two places, GoPlaces collects raw sensor data along the walking trajectory. A trajectory is an ordered sequence of data blocks, $trData = \{db_i\}_{i=1}^{s}$, where s is the number of samples and depends on the travel duration along the path and the sampling rate. GoPlaces divides each trajectory into shorter segments for the following reasons: (i) it is easier to classify accurately short segments than whole trajectories, which ultimately improves prediction accuracy; (ii) the inference latency is reduced, as segments can be classified as soon as their data is collected; (iii) it enables inference based on sub-trajectories, as users may start walking from any position of a trajectory used in training.

To create the segments, GoPlaces analyzes the direction values in the data blocks of the trajectory, and determines the change-of-direction events during walking. A segment contains the data between two change-of-direction events, and it is represented as $sg = (trID, dbS, dbE)$, where $trID$ is the ID of the trajectory, and dbS and dbE are start and end indexes of the data blocks for that segment. Thus, each trajectory is represented as a sequence of segments, $tr = \{sg_i\}_{i=1}^{p}$, where p is the number of segments that form the trajectory.

5.1.3 Semantic place

Users can define indoor places and label them with semantic names, as each user may have different places/trajectories in a shared indoor space. These semantic places (SP) can be in the same room or in different rooms of an indoor space, as long as they are separated by a minimum distance derived from the measurement accuracy of WiFi-RTT [175]. The size of the place is also determined by this accuracy (i.e., 1.5m×1.5m in our experiments). GoPlaces maintains a list of places and assigns a unique ID to

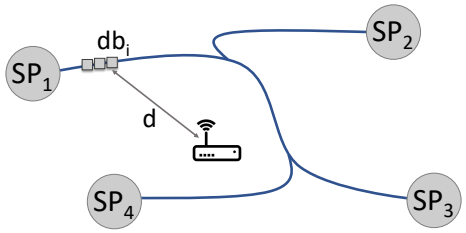


Figure 5.1 Trajectories and data blocks among four semantic places (SP).

each place. During the training phase, it records the trajectory data $trData$ from one place to another. This data is represented as $spTM = (sSpID, eSpID, trID)$, where $sSpID$ is the ID of the start place and $eSpID$ is the ID of the end place of the trajectory.

After data collection, GoPlaces has a list of IDs of semantic places $spList = \{sp_i\}_{i=1}^{n}$ and trajectories for different pairs of SPs $spTMList = \{spTM_i\}_{i=1}^{m}$. Here, m is the total number of trajectories for n places. Figure 5.1 shows trajectories between 4 different places and sample data blocks collected along the trajectory, where d is the distance from the AP to the position where the data block is recorded on the phone.

5.1.4 Place prediction

During the inference phase, GoPlaces analyzes the sequence of data blocks along the user trajectory, divides the trajectory into segments, and uses two Attention-BiLSTM classifiers (Subsection 5.2.4) to infer the IDs for the segments visited so far during the current walk. Thus, GoPlaces gets a list of segment IDs ($sgID_l$) by processing a batch of segments $sgDB_l$ using the segment classifiers $attBiLSTM_l$ as shown in Equation (5.1).

$$\mathbf{sgID_l} = attBiLSTM_l(\mathbf{sgDB_l}), l \in [1, 2] \tag{5.1}$$

Then, GoPlaces traverses the prediction tree (Subsection 5.2.5) that stores historical segment ID sequences from one semantic place to another, and calculates the

probability of each place being the destination (Subsection 5.2.6). Finally, GoPlaces predicts the ID of the place with the highest probability.

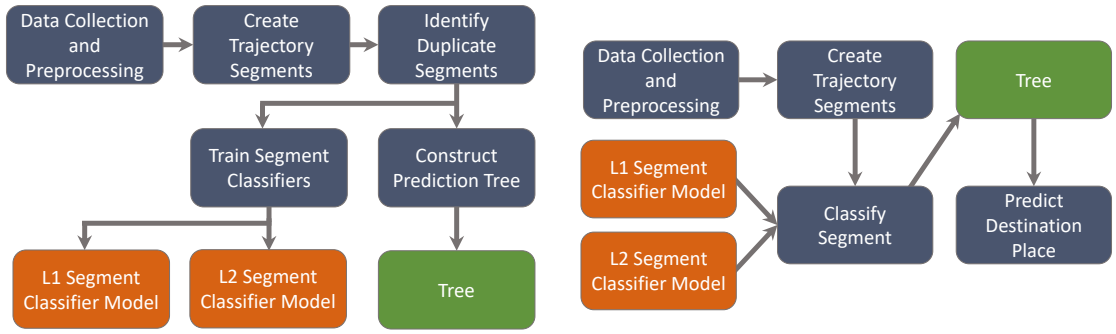
5.2 System Architecture

Since GoPlaces uses a single access point, it is not possible to accurately locate users (i.e., at coordinate level) in indoor environments. Therefore, we design novel algorithms and deep learning models for trajectory data collection, trajectory segment detection and classification, and place prediction. The system architecture shown in Figure 5.2 illustrates the training and the inference phases of GoPlaces. Data collection and preprocessing are similar for both phases. The data is stored on the phone as a sequence of data blocks, and then it is preprocessed to remove noise, especially from the WiFi-RTT distance sequence. Next, the trajectories are divided into segments using a change point detection (CPD) algorithm that analyzes changes in the walking direction. Since the same segment can be identified in different overlapping trajectories, the next module identifies duplicate segments and assigns the same segment ID to all of them. At this stage, GoPlaces has a list of segments, identified by unique IDs. These segments are used as input by the Attention-BiLSTM segment classifier and by the prediction tree.

During inference, user data is collected while walking and data blocks for the last t seconds are analyzed by the CPD algorithm to divide trajectories into segments. Then, the classifier will get the ID of each segment, and the prediction tree will match trajectories consisting of a sequence of segments and predict places. The details of each module are described in the rest of the section.

5.2.1 Data collection and preprocessing

GoPlaces collects accelerometer and magnetometer data at a fixed sampling rate, and calculates the cross product of the gravity vector from the accelerometer and the magnetic field vector from the magnetometer [214]. The rotation of the resulting



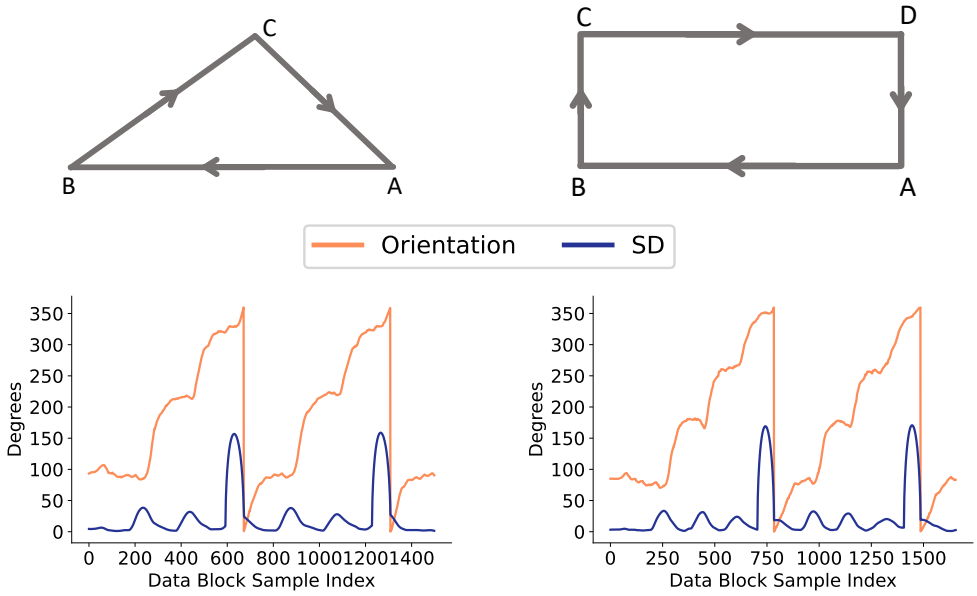
(a) Training phase (b) Inference phase

Figure 5.2 Architecture of GoPlaces app.

vector is then measured and stored as the walking direction in angle degrees (w_d). To detect if the user is walking, GoPlaces uses Android's Activity Recognition API which periodically reads short bursts of data from multiple sensors in the device and reports walking events. At the same time, GoPlaces also submits requests to the AP to get the WiFi-RTT distance between the phone and the AP. In this way, we have a sequence of data blocks, as described in Section 5.1. In our experiments, we assume that the user holds the phone in their hand. However, our approach can be easily extended to everyday scenarios where data can be collected from various placements [183], such as in a pocket, handbag, or on a trolley or stroller.

The WiFi-RTT measurements are noisy, and the errors in measurement are not Gaussian, not always unimodal, have outliers, and are position-dependent [215]. GoPlaces applies a moving average to smooth out the short-term fluctuations and outliers. We experimentally determined that a window size of 10 data blocks works well, and it does not introduce a significant delay for segment classification and place prediction.

GoPlaces collects data until each trajectory has a minimum number of samples for successful training, which we determined experimentally to be seven. By selecting a minimum number of samples, we avoid wasting resources on the phones to collect



(a) Trajectory ABCABCA (b) Trajectory ABCDABCDA

Figure 5.3 Direction and SD patterns for two trajectories.

too much training data. To reduce the manual effort required for data collection, we proposed several optimization techniques in Section 5.3.

5.2.2 Creating trajectory segments

GoPlaces analyzes the time series of walking direction data (w_d) to find the point where it changes by a significant amount. For this purpose, we use a change point detection (CPD) algorithm [216] that divides a time series into pieces, where each piece has its own statistical characteristics. In our case, we know the range of values, angles in $[0, 360)$, and also know that humans do not change their walking direction with high frequency. Therefore, the CPD algorithm can apply an approach based on a sliding window through the data points. Given a window of size sz_w, the CPD algorithm uses a cost function to obtain a cost value, and if the cost exceeds a predefined threshold value, the midpoint of the window is marked as a change point. The cost function and the threshold value are determined experimentally based on data and the requirements of an application.

GoPlaces uses standard deviation (SD) of w_d as the cost function. The SD values are low if there is no change in direction, and they rise if there is a significant transition in the direction pattern. We create a new segment when the change in direction is at least 45 degrees, which we choose as a threshold for a significant turn. Experimentally, we found that most of the segments can be detected using 20 as the threshold value for SD. During the preliminary experiments, we noticed that one fixed-size sliding window might miss some change points because a smaller sliding window fails to capture transitions which take a long time, while a larger sliding window might miss short transitions. Therefore, GoPlaces uses several window sizes (sz_w in [60,180]) and executes the CPD algorithm for each window to capture both short and long transitions. This design introduces an acceptable overhead while ensuring accurate segment detection. Figure 5.3 shows the variations in the SD patterns for two different trajectories, and it demonstrates that GoPlaces can detect the change points accurately.

After dividing the trajectories into segments based on the direction patterns, GoPlaces analyzes the segments based on duration (i.e., number of data blocks). If the size of a segment is less than 50 data blocks (equivalent to one second), it merges the segment with the next one. If the size of a segment is very large, the segment is divided into equal-sized segments of less than 500 data blocks each. This allows for faster segment classification in real-time. Finally, each segment is assigned a unique ID, and the sequence of segments for each trajectory is stored in a database.

5.2.3 Identifying duplicate segments

Some of the trajectories in an indoor space will likely overlap and share segments. As shown in Figure 5.4, segments sg_2 and sg_5 are identical segments shared by two trajectories, AB and CD. GoPlaces identifies the segments that are duplicated in different trajectories and assigns them the same ID. For training, GoPlaces has

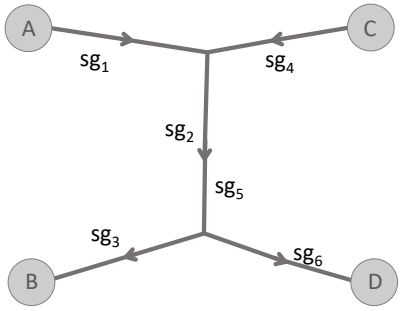
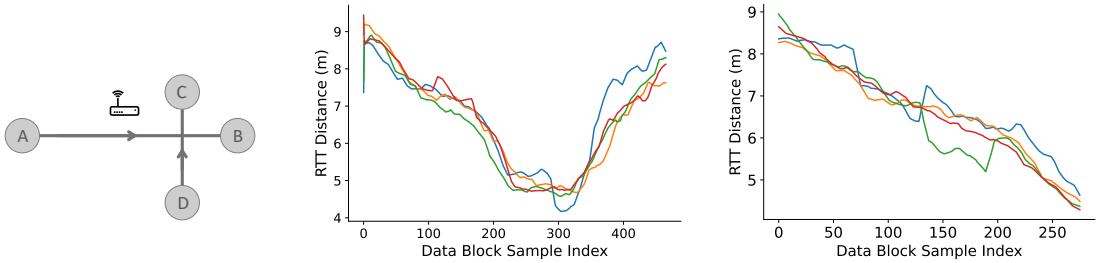


Figure 5.4 Segment with different IDs in two trajectories.



(a) (b) For Segment AB (c) For Segment CD

Figure 5.5 RTT distance trends for different samples of two segments (b, c), where each line shows a trial. Segments and router position shown in (a).

multiple samples for each segment of a trajectory. Although the WiFi-RTT distance measurements are noisy, we observe similar trends for measurements of the same segment. For example, Figure 5.5 shows the WiFi-RTT data trends for two segments: AB and CD. There are four samples for each segment, and their patterns are the same for each segment.

To check if two segments from different trajectories are identical, we follow three steps: First, we check all possible pairs of samples by taking one sample from each segment. The complexity of this part is $\mathcal{O}(m^2n^2)$, where m is the number of segments and n is the number of samples for each segment. The number of segments, m, depends on the size of the indoor space and the number of trajectories covered. The number of samples, n, is a constant, as GoPlaces uses a fixed number of samples for each trajectory. While this step is expensive, it is executed offline only once before training.

Second, we consider two samples to be matched if: (a) the difference between the mean w_d of two samples is less than 10 degrees and direction is constant or follows a similar (increasing or decreasing) trend. This step is required because the segments are not necessarily straight; (b) the statistical measures describing the shape of their distributions are similar. GoPlaces calculates the skewness and kurtosis of two samples, and considers them similar if the absolute differences are less than 20% of actual values; and (c) the similarity score between the WiFi-RTT distance sequences of two samples is less than a predefined threshold (sim_{th}). We apply the Dynamic Time Warping (DTW) algorithm [217] to measure the similarity score on the normalized WiFi-RTT distance sequence. DTW compares sequences with different lengths by calculating the Euclidean distance between data blocks. This is done by building one-to-many and many-to-one matches to create a warping path, such that the total distance can be minimized between the two sequences. The average distance of the warping path is reported as the similarity score, and we consider two samples to be identical if the similarity score is less than sim_{th}, which is defined as the normalized distance value for average WiFi-RTT error divided by the maximum WiFi-RTT distance in a given indoor space. We take this value as the threshold, since we define a place as a square with the sides equal to the WiFi-RTT error.

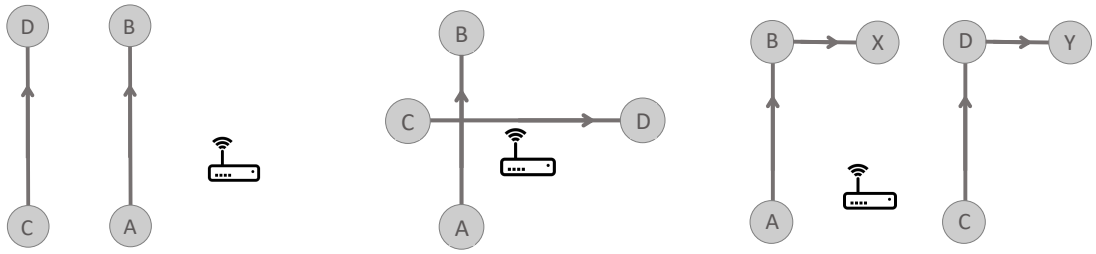
Third, if a certain percentage (d_{th}) of samples for two segments are matched, we consider these segments identical and assign them the same ID. The effect of setting different d_{th} values is discussed in Subsection 5.4.10.

5.2.4 Segment classifier model

Due to the capacity of deep learning techniques to extract information from time series in a quicker and more thorough manner than traditional methods, we choose to apply them for segment classification. The segment classifier model in GoPlaces takes the segment data blocks as input and infers the ID of the segment. Segments that have

different WiFi-RTT distance trends and different walking direction trends are easy to identify. However, there are three cases where either the WiFi-RTT distance trends or the walking direction trends are similar between different segments, and these cases make classification difficult. Figure 5.6 illustrates these cases. In Figure 5.6(a), we see two segments with the same direction, but different WiFi-RTT distances. As long as the difference between the two WiFi-RTT distances is higher than the typical error of WiFi-RTT ranging (1.5m in our experiments), the classification is expected to work due to the difference in distance from the AP. In Figure 5.6(b), we see two segments with the same WiFi-RTT distance, but different walking directions. In this case, the classification is expected to work due to the difference in the walking direction. In Figure 5.6(c), we see two segments, AB and CD, that have the same WiFi-RTT distance trends and the same walking direction trends. A classifier can differentiate between these segments if it analyzes the segments with which they are connected. In our example, if the user moves from AB to BX, and from CD to DY, then the WiFi-RTT distance patterns of ABX and CDY will be different, even though the walking direction patterns are the same. Due to this case, we decided to build classifiers for both individual segments (L1 segments), called L1 classification, and segments consisting of two connected segments (L2 segments), called L2 classification. We do not need to consider three or more level segments, as our experiments showed they do not improve place prediction accuracy significantly, while requiring more training, increasing the number of branches in the prediction tree (Subsection 5.2.5), and increasing the inference time.

The next question we face is whether to train a single classifier for both L1 and L2 classifications or one classifier for L1 and one for L2. The experiments presented in Subsection 5.4.10 show that having two different classifiers works better because a common classifier can misclassify an L2 segment as one of the two segments that form it.



(a) (b) (c)

Figure 5.6 Segments with similar patterns: (a) AB and CD have the same walking direction, (b) AB and CD have the same WiFi-RTT distance (c) AB and CD have the same walking direction and WiFi-RTT distance.

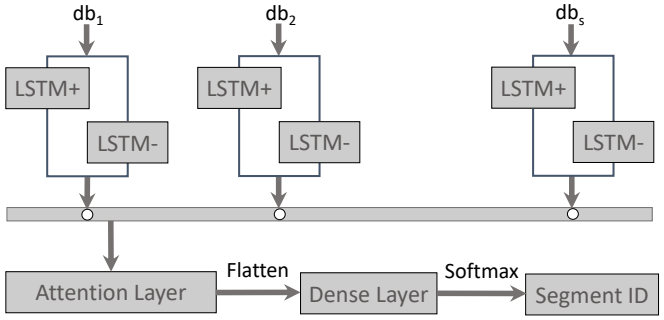


Figure 5.7 Framework for attention-BiLSTM model.

Deep learning-based systems [183] avoid the classic accumulation errors of rule-based methods (e.g., dead reckoning [218]) by focusing on features extracted from data, rather than directly integrating sensor signals. To classify segments, we designed a BiLSTM model with an attention layer (attention-BiLSTM), as shown in Figure 5.7, where the output of the BiLSTM layer is used as the input of a self-attention layer with a sigmoid activation function. The input of the model consists of the sequences of data blocks associated with the segments. Multivariate time series classification, such as the classification of the sequences of data blocks in our case, has been broadly examined in diverse domains over the past decade. Recurrent neural networks (RNN) have been used to solve such problems, and we experimented with several types of RNNs, such as gated recurrent units (GRU), long short term memory (LSTM), and bidirectional LSTM (BiLSTM) for segment classification. Compared to the other

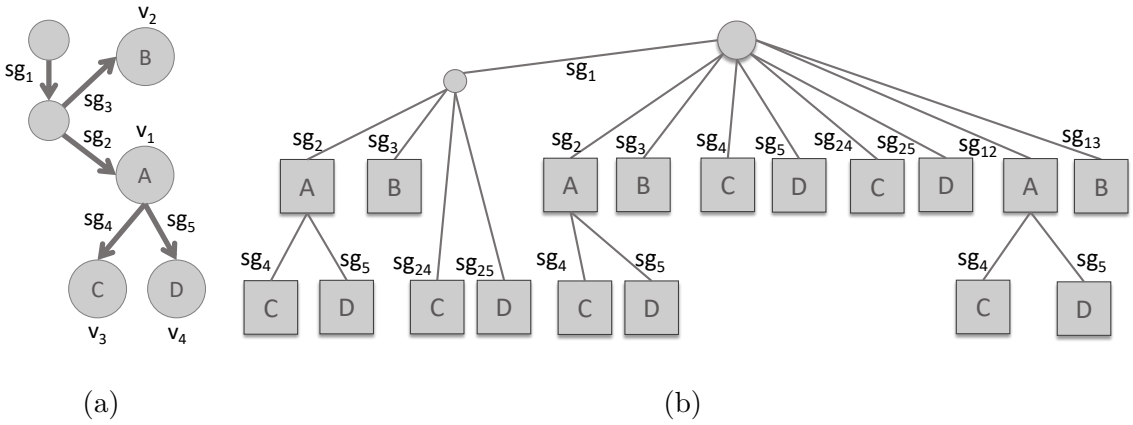


Figure 5.8 Trajectories for four places (a), and their associated prediction tree showing the segments, the place nodes, and the abstract nodes (b).

models, BiLSTM captures more contextual information, which helps to perform better and learn faster. A specific benefit of BiLSTM is that it processes data in both the forward and the backward directions and, thus, it learns the sequence of data blocks in both directions, even if the training data contains data for only one direction. We augmented the BiLSTM layer with an attention layer because previous studies [219] have shown that this combination helps to boost performance in the case of sequential data since the attention layer is able to focus on important information such as the rate of pattern changes. In order to avoid potential over-fitting problems, a dropout layer is used between these two layers. The output of the attention layer is fed into a dense layer with softmax as an activation function. The final output of the model is the probability for each segment ID. To predict the destination place, GoPlaces considers the segment with the highest probability score. GoPlaces uses the same framework to train both L1 and L2 classifiers.

5.2.5 Prediction tree

GoPlaces stores trajectories in a tree data structure, where each segment of a trajectory is a branch and each place is a node. A place node stores the place ID and visit frequency for a trajectory, indicating the number of times the user has visited it.

In addition to place nodes, the tree also contains the root and internal nodes, which are abstract entities that do not store any information but serve as points to connect segments in the data structure. During the tree construction, GoPlaces uses both L1 and L2 segments to create the possible paths from the root to the destination places.

A sample prediction tree is shown in Figure 5.8 for four trajectories $tr_1 = \prec sg_1, sg_2, A \succ$, $tr_2 = \prec sg_1, sg_3, B \succ$, $tr_3 = \prec sg_1, sg_2, sg_4, C \succ$ and $tr_4 = \prec sg_1, sg_2, sg_5, D \succ$. The prediction tree stores all the possible paths for these trajectories as a combination of L1 and L2 segments. In Figure 5.8(b), sg_{ab} represents a L2 segment that joins sg_a and sg_b. If there are k L1 segments in a trajectory to a place, we have at most $f(k)$ paths from the root to the place nodes, including the IDs for both L1 and L2 segments in the prediction tree, as shown in Equation (5.2).

$$f(k) = 2 + f(k-1) + f(k-2), where f(1) = 1, f(0) = 0 \tag{5.2}$$

The place nodes of the prediction tree store the visit frequencies v_i following a trajectory tr_i. For example, the nodes for place A and C store the visit frequencies v_1 and v_3, respectively.

The depth of the prediction tree $tree_d$ for a given indoor space depends on the maximum number of segments in a trajectory. To avoid extensive computation during inference, if the maximum number of segments is very high, GoPlaces can limit $tree_d$ to a certain value k_{max}, and analyze just the last k_{max} segments, as these segments are most likely to determine the destination.

There are several benefits of storing trajectory data in this format: (i) As we have paths from any segment of a trajectory to the destination place, GoPlaces can predict well even if the user does not start from the original place of the training trajectory; (ii) some incorrectly classified segments can be handled by the prediction tree because incorrect segment IDs from the classifiers typically lead to an invalid path in the tree. However, there are situations when an incorrect segment ID may lead to

a valid, but incorrect path, which will result in an incorrect prediction. Nevertheless, the classifiers can detect most segment IDs correctly, and the prediction will work well. Thus, storing all possible paths in the tree helps to check multiple options and improves the probability of traversals through correct paths that increases the accuracy of place prediction; (iii) if there are several paths to different destinations from the same segment, these paths can be represented in the prediction tree without creating multiple branches (e.g., sg_1 in Figure 5.8 is a common branch for four destinations); (iv) keeping track of visit frequencies allows predicting a place with the highest probability when multiple places follow the same trajectory.

5.2.6 Inference

During inference, GoPlaces collects and analyzes sequences of data blocks while the user is walking. As discussed in Subsection 5.2.2, the maximum length of the data block sequence of a segment is 500. Therefore, GoPlaces analyzes the last $t = 500 \times tree_d$ data blocks. Algorithm 8 presents the pseudo-code for the inference phase. First, the trajectory data for the last t data blocks is divided into segments using our CPD algorithm (Algorithm 8 Line 2). Then, the last $k = tree_d$ segments are analyzed to create both L1 and L2 segments, and the classifiers are used to predict the ID for each segment (Algorithm 8 Lines 3-4). Using k segments, GoPlaces can create sequences as described in Equation (5.2), and it traverses the sequences following the paths in the prediction tree (Algorithm 8 Line 5). The sequences created by the correctly predicted segment IDs follow the correct path to the place nodes. On the contrary, incorrect segment IDs will lead to invalid sequences, which can be discarded as they fail to follow the branches of the prediction tree. Incorrect classification may rarely result in a real, but incorrect path, too.

To predict the destination place, GoPlaces calculates the probabilities (Algorithm 8 Line 6) for the places by (a) counting the possible paths that lead to that place,

following a predicted sequence of segments; and (b) assigning higher weights to longer paths and paths with L2 segments. If there are p possible places in a testbed, the weight for each place sp is assigned by Equation (5.3), where r is the number of paths that lead to sp, and $n1_i$ and $n2_i$ are the numbers of L1 and L2 segments in each path. GoPlaces calculates the visit probability for a place sp_i using Equation (5.3), where r is the number of paths that lead to sp_i, $P(sp_i|path_j)$ is the visit probability of a place sp_i following a path $path_j$, and $n1_j$ and $n2_j$ are the numbers of L1 and L2 segments in $path_j$. GoPlaces measures $P(sp_i|path_j)$ using the visit frequencies associated to a place node in the prediction tree following a path $path_j$, and assigns higher weights to longer paths and paths with L2 segments.

$$P[sp_i] = \frac{\sum_{j=1}^{r}((n1_j + 2 \times n2_j) \times P(sp_i|path_j))}{\sum_{j=1}^{r}((n1_j + 2 \times n2_j)} \tag{5.3}$$

Finally, GoPlaces outputs the place with the highest probability as destination places. (Algorithm 8 Line 7)

Algorithm 8 Inference Pseudo-code

Require: data_blocks

Ensure: Predicted Place ID

Analyze($data\_blocks$)

1: $segment\_list \leftarrow CPD(data\_blocks)$

2: $L1\_IDs \leftarrow L1\_Classifier(segment\_list)$

3: $L2\_IDs \leftarrow L2\_Classifier(segment\_list)$

4: $valid\_paths \leftarrow prediction\_tree(L1\_IDs, L2\_IDs)$

5: $place\_probabilities \leftarrow predict\_place(valid\_paths)$

6: **return** Place ID with $max(place\_probabilities)$

5.3 Training Optimizations

In this section, we propose several optimization techniques to reduce the manual effort required for training data collection. We introduce a data collection technique wherein a few samples are manually collected and labeled, which can be used for labeling the data collected automatically in the background. Additionally, we present a data augmentation technique to increase the training sample size, which in turn can be utilized to train the segment classifiers. Furthermore, we propose a design where a global user shares labeled data with individual users, eliminating the need for them to collect training data explicitly.

5.3.1 Automatic training data collection

To reduce the manual effort required for data collection, we propose an automatic training data collection technique. Initially, the user collects a few samples for a trajectory by selecting the origin and the destination places explicitly. The manually collected samples are labeled by the user and added to the trajectory list, denoted as trL_M. Once a trajectory has a few such samples, GoPlaces starts collecting and labeling additional training samples for this trajectory automatically (in the background), while the user is walking. Finally, GoPlaces concatenates the automatically labeled trajectory list (denoted as trL_A) with trL_M to prepare the trajectory list trL as the training dataset.

To label the automatically collected trajectories, GoPlaces matches them with the trajectories collected manually by the user. Two trajectories are matched if their similarity score is less than a predefined threshold value. We apply the DTW algorithm (Subsection 5.2.3) to measure the similarity score on the normalized WiFi-RTT distance sequence and the direction sequence. We consider two trajectories to be identical if the similarity score for the direction pattern is less than dir_{th} and the similarity score for the WiFi-RTT distance pattern is less than sim_{th}. We

experimentally determined the value for dir_{th} to be 0.001, which is fixed for all indoor environments. For sim_{th}, we use the same logic as in Subsection 5.2.3. This algorithm discards incomplete or invalid trajectories, since they get similarity scores exceeding the threshold values. We use low threshold values to reduce false positives, at the expense of discarding some data. GoPlaces also discards the trajectories matched with more than one trajectory to ensure a uniquely labeled trajectory list.

5.3.2 Segment data augmentation

GoPlaces further minimizes the manual effort for training data collection by augmenting the training dataset collected manually and automatically with synthetic training data, which helps the classifiers (Subsection 5.2.4) to learn faster and improves generalization performance [220]. To generate synthetic data, we add random noise to the original sequences of walking direction and WiFi-RTT distance data blocks. Since both types of sensor data are noisy, including new samples drawn from the vicinity domain of known samples can smooth the structure of the input space. We also expand or shrink the sequences to simulate data blocks generated at different walking speeds. Also, we extract partial trajectories from the original ones and add them to the dataset, which can improve the segment classification for partial segments (i.e., the user travels part of a segment).

5.3.3 Global data collection

To reduce the effort of explicit data collection by each user, we propose an approach where the initial global data $\mathcal{G}_D$ is collected by the owner $\mathcal{G}$ of the space (e.g., a public space) and shared with the users. During the global data collection process, $\mathcal{G}$ labels the semantic places and collects at least two samples for each potential trajectory in the indoor space. Thus, the trajectory data can be used as a global prediction tree, as described in Subsection 5.2.5. However, the prediction tree does not include visit

frequencies. The global prediction tree, together with the trajectory data, is then shared with users within the indoor space.

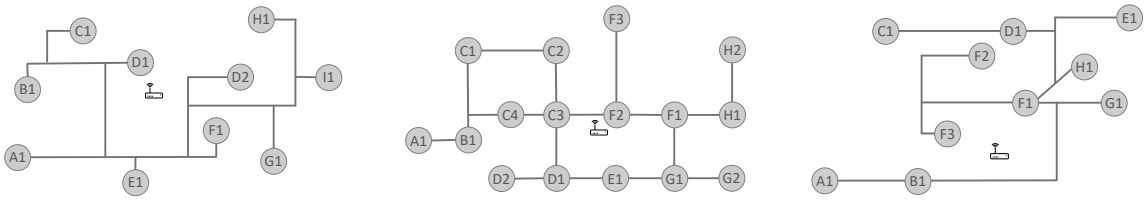
At the user level, trajectory data is automatically collected and compared with trajectories in $\mathcal{G}_D$, as detailed in Subsection 5.3.1. At this stage, visit frequencies for each trajectory are counted, as GoPlaces identifies matches with trajectories in $\mathcal{G}_D$. Thus, the user-level prediction tree holds personalized information that can be used for personalized prediction.

It might be possible that the user may visit new trajectories that are not in $\mathcal{G}_D$. These data are stored separately as $\mathcal{G}_U$. GoPlaces computes the similarity of the current trajectory with trajectories in $\mathcal{G}_U$ as in Subsection 5.3.1. If a trajectory has been recorded a sufficient number of times, users can be prompted to label it by providing the start and destination places.

5.4 Evaluation

Since GoPlaces aims to provide a practical solution for place prediction on smart phones, with minimal infrastructure support, our problem settings are different from those of other place prediction systems. We do not attempt to compare against them quantitatively, as those systems cannot work with the data model of GoPlaces. However, we provide a qualitative comparison in Section 2.4.

The evaluation has several goals: (i) quantify the overall performance of place prediction in indoor spaces of different sizes, (ii) evaluate the automated data collection technique, (iii) analyze the performance of segment classifiers, (iv) quantify the benefits of model personalization, (v) evaluate the prediction accuracy for new users who did not collect data or trained the model, (vi) perform ablation studies to understand the effects of different parameters, and (vii) test the app latency and resource consumption on phones.



(a) Testbed#1 (b) Testbed#2 (c) Testbed#3

Figure 5.9 Indoor testbeds, showing the places, the AP positions, and the trajectories between the places.

5.4.1 Implementation and experimental settings

We implemented GoPlaces in Android using DL4J [15], which supports a wide range of neural network architectures for both training and inference on the phones.[1] We used Google Nest Wifi, as an AP which supports WiFi-RTT. During training, GoPlaces stores the collected WiFi-RTT and sensor data on the device under the App's container. Since WiFi-RTT (i.e., 802.11-2016 FTM standard) is supported on phones running Android 9 (API level 28) or newer, we chose to use this Android version for implementation. The Android prototype of GoPlaces has been tested using Android Google Pixel 3 & 4 phones.

We also implemented training and testing in Keras and used it to optimize the algorithms and evaluate their performance offline, using data collected on the phones. The best algorithm parameters in Keras were used in the Android implementation.

Since GoPlaces needs to distinguish between short trajectories and nearby places, we test it in relatively smaller spaces with many places, rather than in larger spaces with few places. We used three indoor environments for testing with areas of 170 m^2, 167 m^2, and 300 m^2, respectively. Figure 5.9 shows the setups for these indoor spaces: Testbed#1, Testbed#2 and Testbed#3, with 10, 16 and 10 semantic places, respectively. Each place is labeled with one character and one digit, where the character represents a room and the digit represents a place ID in the room.

[1]TensorFlow Lite [16] has recently become available for on-device training, and therefore it could also be used to implement GoPlaces.

We considered walls and large objects in our experimental setups and tested places that are both in LoS and NLoS. We found that the RTT distance patterns remain consistent as long as there is no change in the environment, such as alteration in indoor layouts or wall materials. We tested GoPlaces in the presence of multiple users.

5.4.2 Data collection

For training, GoPlaces needs to collect trajectory data that cover all the segments and all the places. However, GoPlaces does not need to collect training data for every pair of places. We experimentally determined that collecting seven samples for each segment (as part of the same or different trajectories) works well. For manual training data collection[2], the user has to select the origin and destination places, and then start walking. In all experiments, the user holds the phone in hand, including for automatic data collection or inference. The segments are identified and labeled automatically by our CPD algorithm. We use the sampling rate of 50 samples/second for inertial sensors to ensure change of direction is detected well, and a sampling rate of 10 samples/second for WiFi-RTT measurements. We store the data locally as a sequence of data blocks at a rate of 50 data blocks per second. Since WiFi-RTT sampling rate is lower than this rate, the WiFi-RTT distance collected at a certain timestamp is copied to consecutive data blocks until GoPlaces gets the next RTT measurement. Our results in Subsection 5.4.11 demonstrate that lower WiFi-RTT sampling rates can be used effectively to save battery power, with a minimal decrease in the prediction accuracy.

For training, two samples for each trajectory are collected explicitly by the user, and other samples are collected and labeled automatically following the technique discussed in Subsection 5.2.1. All automatically collected trajectories that pass the identification threshold are used in training, even if some of them are misidentified, in

[2]Data was collected by the members of our team.

Table 5.1 Statistics of the Training and Test Datasets

	Testbed#1	Testbed#2	Testbed#3
Places	10	16	10
Trajectories	30	24	18
Samples (Training, Testing) for trajectories	(236, 40)	(178, 40)	(126, 53)
L1 segments	48	48	57
L2 segments	62	28	41
Samples (Training, Testing) for L1 segments	(856, 148)	(388, 83)	(368, 151)
Samples (Training, Testing) for L2 segments	(620, 108)	(211, 43)	(242, 98)

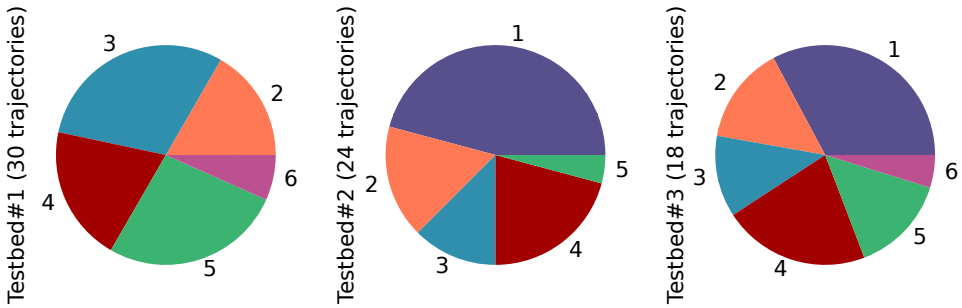


Figure 5.10 Distribution of number of segments per trajectory.

order to provide realistic results. The statistics of the trajectories and segments in the experiments are presented in Table 5.1. These statistics do not include the synthetic data used for training data augmentation. Figure 5.10 shows the percentages of trajectories in the experiments that have n segments, where $n \in [2..6]$ for Testbed#1, $n \in [1..5]$ for Testbed#2 and $n \in [1..6]$ for Testbed#3.

5.4.3 Deep learning models

For the segment classifiers, we experiments with different sequence-based neural network models, each designed for multi-class classification task. Each model begins with a masking layer, which ignores padding values (set to 0.0) in the input sequence, allowing the model to focus on the relevant parts of the data. A recurrent neural network layer (GRU, LSTM, or BiLSTM) with 20 or 40 units follows, capturing temporal relationships within the sequences while applying dropout for regularization. In some configurations, the models also include a self-attention layer with a sigmoid

activation, which highlights important sequence elements and enhances the model's ability to learn meaningful patterns over time. After flattening the output, a final dense layer with softmax activation produces a probability distribution across the classes (i.e., segment IDs). The models use categorical_crossentropy as the loss function and the ADAM optimizer with a learning rate of 0.001, and it tracks accuracy as a performance metric. This architecture is well-suited for tasks that require context-based sequence classification. The results for different models used as the segment classifiers are presented in Subsection 5.4.10.

5.4.4 Metrics

To evaluate the segment classifiers, we use accuracy, precision, recall, and F1 score metrics. We use the notation L1-C and L2-C to denote the accuracy of L1 and L2 classifiers. We also report the place prediction accuracy at a certain percentage $p\%$ of the current traveled trajectory. Specifically, $p\%$ is computed based on the total number of data blocks of a trajectory. For system performance on the phone, we report training and inference latency, memory, and battery consumption.

5.4.5 Overall classification and prediction results

The experiment measures the effectiveness of our segment classification and place prediction algorithms. For this experiment, each segment collected by the user was augmented with 30 synthetic samples, as we discussed in Subsection 5.2.1. The model contains a BiLSTM layer with 40 neurons and 25% dropout rate, followed by the attention layer. We use the same architecture to train both L1 and L2 classifiers. We train the networks for 80 epochs, with early stopping.

Table 5.2 shows the results of segment classification. Both the L1 and L2 segments in all testbeds are classified with more than 87% accuracy. Although segments from all classes are not represented equally in the training dataset, which represents a realistic scenario where users walk some trajectories more often than

Table 5.2 Performance of L1 and L2 Classifiers

	Testbed#1		**Testbed#2**		**Testbed#3**	
	L1-C	L2-C	L1-C	L2-C	L1-C	L2-C
Classes	48	62	48	28	57	41
Accuracy(%)	89.2	90.8	91.1	92.3	87.6	90.0
Precision(%)	91.0	91.0	91.0	91.0	87.3	90.0
Recall(%)	89.0	91.0	91.0	92.0	87.9	90.0
F1 Score(%)	88.0	90.0	90.0	90.0	87.6	89.0

others, we achieve a high F1 score for both classifiers. This means the classifiers work well for realistic imbalanced multi-class datasets. The L2 classifier performs slightly better than the L1 classifier, as it is easier to distinguish between different classes for L2 segments.

Using both segment classifiers, Table 5.3 presents the place prediction accuracy, with respect to the percentage of progression towards destination (p). GoPlaces achieves competitive results when p is at least 70%-75%. The accuracy for place prediction is over 86% when $p > 90\%$. The accuracy for Testbed#3 is higher compared to the other testbeds, as the places are sparsely distributed in a larger area and trajectories are longer compared to the trajectories in the other testbeds.

The accuracy at low values of p is relatively low because, at that stage, there are multiple possible trajectories, leading GoPlaces to identify more than one destination with equal probability. This accuracy can improve over time as GoPlaces collects more data automatically and uses the visit frequencies to predict the most likely destination with higher probability. Additionally, the application developer can set a threshold probability to meet specific application requirements.

5.4.6 Performance of automatic data collection

This experiment evaluates the performance of the automatic training data collection technique in the three testbeds. As shown in Figure 5.11, more than 76% of trajectories are correctly identified in all testbeds, while the false positive rate is

Table 5.3 Place Prediction Accuracy for Different Percentages of Progression Toward Destination (p)

p	Testbed#1	Testbed#2	Testbed#3
65	51.8	69.0	71.0
70	56.9	71.8	72.9
75	66.8	73.7	75.1
80	74.6	76.5	78.5
85	81.5	78.7	83.2
90	84.0	82.0	86.2
95	87.0	87.2	88.1
100	90.7	91.0	90.9

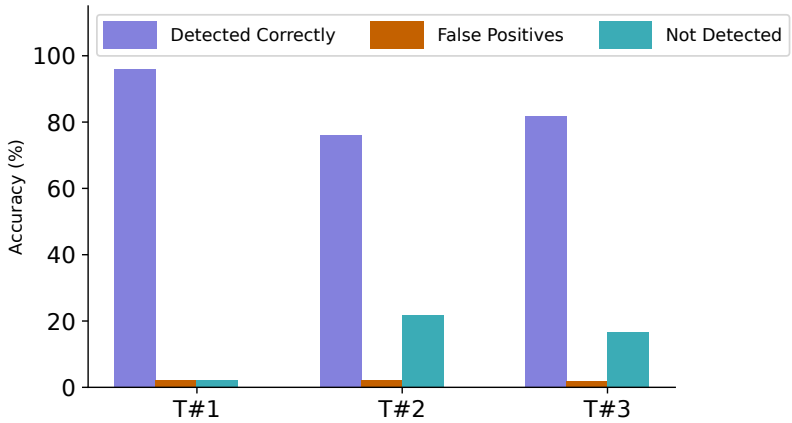


Figure 5.11 Accuracy of automatically collected datasets.

less than 2%. More than 95% trajectories in Testbed#1 are labeled correctly, as all of the trajectories in this testbed have two or more segments, which means that changes in the orientation patterns of the trajectories help to match them uniquely with manually labeled trajectories. We discard the trajectories which are (a) not matched with any trajectory or (b) matched with more than one trajectories. All the other trajectories are included to the training dataset, including the false positives (i.e., the system would not know they are false positives).

Table 5.4 Performance of Global Data Collection, Where Data for 24 Unique Trajectories Are Collected Globally and Shared with Two Users

User ID	Number of Unique Trajectories	Total Number of Samples	Percentage of Samples Detected Correctly
U_1	15	80	75%
U_2	18	94	82%

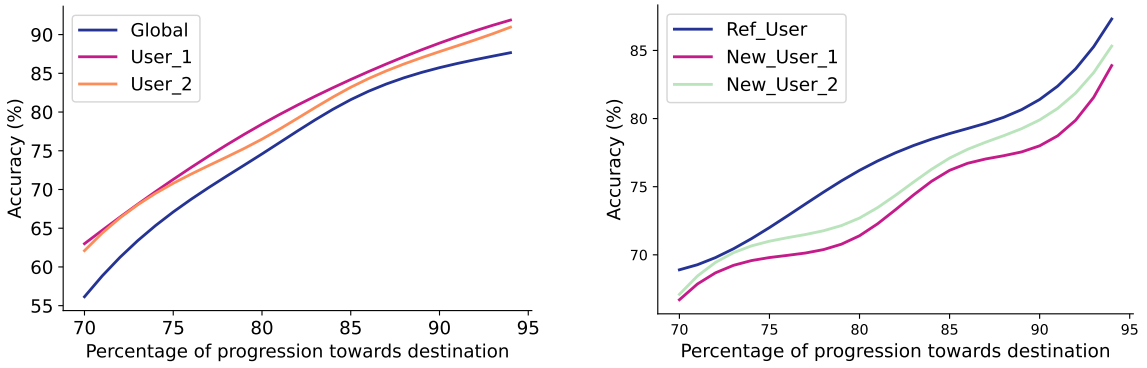
5.4.7 Performance of global data collection

In Subsection 5.4.6, we evaluated the performance of automatic data collection, wherein both manual and automatic data are collected by the same user. In this experiment, we assess the performance of global data collection across M different trajectories, where k samples per trajectory are collected by a global user $\mathcal{G}$. These samples ($\mathcal{G}_D$) are then shared with individual users to enable them to continue data collection automatically for a subset of trajectories, without explicitly selecting the start and destination places.

In Testbed#2, $\mathcal{G}$ collects $k = 2$ samples for $M = 24$ trajectories and shares the data with two users U_1 and U_2, who subsequently collect samples for a subset of trajectories, $M_1 = 15$ and $M_2 = 18$ respectively. Our results in Table 5.4 show that 75% and 82% of trajectory samples are detected correctly for U_1 and U_2 respectively. This demonstrates a significant reduction in the effort required for individual data collection. Each user can then train their own segment classifiers and prediction trees to obtain personalized predictions from the automatically collected dataset.

5.4.8 Performance of personalization

GoPlaces allows users to collect data and train personalized prediction models that work for their place labels and trajectories. To quantify the benefits of personalization, this experiment compares the place prediction accuracy of the personalized model against a global model that includes data from all users in the indoor space. This experiment ignores the potential privacy problems of a global model that uses data from multiple users. Specifically, we conducted the experiment in Testbed#1, where



(a) (b)

Figure 5.12 Place prediction accuracy (a) for personalized and global models (b) for new users.

two users (User_1 and User_2) collect data and train their personalized models individually. Both users collect data for 20 unique trajectories covering 10 places, train the personalized segment classifiers, and build the prediction trees. Then, the data from both users are combined to get 30 unique trajectories (some trajectories are common for both users) and the global classifiers and the global prediction tree are constructed. For each user, we use both the personalized model and the global model to predict the destination places. The average prediction accuracy for both models is reported in Figure 5.12(a), which shows the place prediction accuracy with respect to the percentage of progression toward destination. The results demonstrate that the personalized models for User_1 and User_2 perform better compared to the global model in all cases. Also, the personalized models can predict earlier than the global model with good accuracy because the individual users' historical data helps the models and the prediction tree to reduce the search space, whereas the global model does not work well when there are multiple possible destination places on the current trajectory.

5.4.9 Performance for new users

This experiment evaluates if GoPlaces trained by one user (reference user) works without retraining for other users in the same space. If it does, it means new users can easily use GoPlaces in new indoor spaces by re-using already trained models. At a later time, they can add new personalized semantic places and retrain the model. In this experiment, we assume that the new users' movement habits are the same as the reference user. However, if new users exhibit different movement patterns, such as varying frequencies in visiting specific trajectories compared to the reference user, then the prediction tree will need to be updated. This update can be performed automatically through our automatic data collection module.

The experiment trains the segment classifiers and constructs the prediction tree using a dataset collected by a reference user (Ref_User) in Testbed#2. Then, we report the place prediction accuracy with three different test datasets collected by (a) Ref_User, (b) New_User_1, and (c) New_User_2. Here, similar trajectories are followed by all users in the same environment. Figure 5.12(b) shows that GoPlaces trained by Ref_User works well for the new users, with only a slight decrease in performance. For example, both new users achieve over 70% accuracy when $p \geq 80\%$. These results show that, although the new users' walking speeds may differ, the segment classifiers perform well in detecting trajectories and GoPlaces can use the reference user's prediction tree to predict the destination places for the new users.

5.4.10 Ablation study

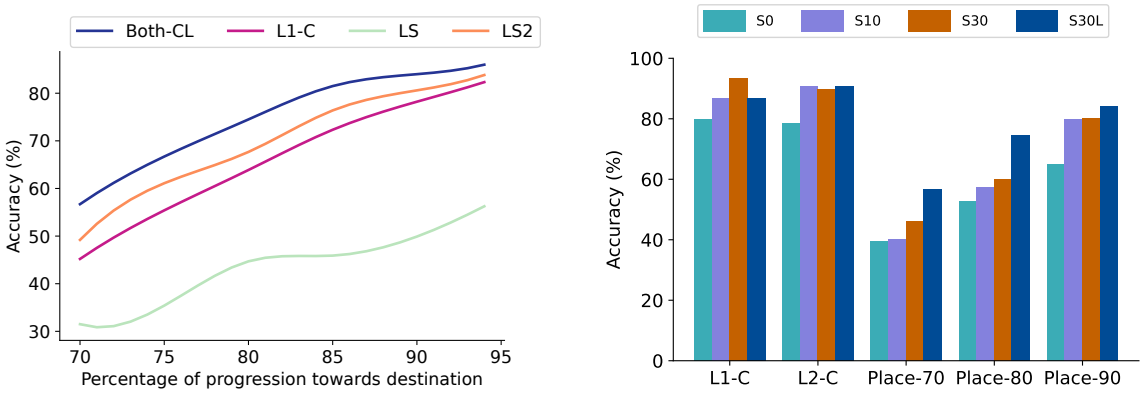
This set of experiments uses the Keras implementation to analyze the effects of several parameters on classification and prediction. We report the results only for Testbed#1 because other testbeds show similar behavior.

Effect of classifier type. GoPlaces uses L1 and L2 segments to predict destination places, which requires to construct the prediction tree and train the

classifiers using both types of segments. This experiment compares the place prediction performance when GoPlaces uses only the L1 classifier vs. L1 and L2 classifiers together. The results in Figure 5.13(a) show that GoPlaces with both classifiers achieves better accuracy, as the L2 classifier helps to detect connected segments and mitigate misclassifications by the L1 classifier.

Effect of short history. Figure 5.13(a) also shows the place prediction accuracy results when the prediction is based only on the last segment (LS) or the last two segments (LS2), instead of all the segments of the trajectory up to the point of prediction. The aim of this experiment is to see if GoPlaces can speed up its inference time and reduce its overhead, while still achieving competitive accuracy. The results show that LS performs poorly in most of the cases, as it fails to utilize the user walking patterns learned from the historical data of trajectories. LS2, however, achieves competitive performance, as it uses both classifiers to detect individual and connected segments; in addition, it can use the prediction tree to predict places based on slightly more historical data than LS. Therefore, if latency or resource constraints on the phone are important factors for certain applications, GoPlaces can balance accuracy with these two factors and can still achieve competitive performance.

Effect of training data augmentation. We evaluate the effectiveness of our data augmentation technique by comparing different amounts of synthetic data in the training dataset. Figure 5.13(b) shows the accuracy of both classifiers and the overall place prediction when the classifiers are trained on: the dataset without any synthetic data (S0), the augmented dataset that includes S0 and 10 or 30 synthetic data samples (i.e., S10 and S30) for each original data sample in S0. The results demonstrate the substantial benefits of data augmentation, especially on place prediction. We also observe that at shorter percentages of progression toward the destination (i.e., 70%), larger amounts of data augmentation lead to significantly better performance. Finally, we observe that the best performance in place prediction is obtained for the training



(a) (b)

Figure 5.13 (a) Place prediction accuracy when using one classifier (L1-C), two classifiers (Both-CL), only the last segment in the trajectory (LS), and only the last two segments in the trajectory (LS2); (b) accuracy of classifiers and place prediction for different percentages of progression toward destination (70%-90%), when varying the amount of synthetic data in training.

dataset with 30 synthetic data samples (S30L) for each original sample, which includes parts of each sample with different lengths (i.e., x% of data blocks from the beginning of the original sequence, where $x \in [50, 100]$). By adding more synthetic data, the overall performance does not improve, while training time increases.

Effect of parameters and algorithms on segment classification. We studied the performance of segment classifiers using different ML algorithms and numbers of neurons to find the most suitable classifier for our problem. Table 5.5 shows results for GRU, LSTM, and BiLSTM with or without the attention layer (Atn). The results show that the accuracy for classifiers with BiLSTM is better compared to those using LSTM and GRU. The accuracy does not improve much by increasing the number of neurons from 20 to 40. The performance of the models is further enhanced by the addition of the attention layer. This benefit comes at the cost of a substantially larger number of parameters, which affects the training latency, but not the inference latency. In general, the performance of a L2 classifier is slightly better than that of the L1 classifier, as it can distinguish between different segments by utilizing the information about changes in orientation and WiFi-RTT distance

Table 5.5 Performance of Different L1 and L2 Classifiers

Network Type	**L1 Classifier**		**L2 Classifier**	
	#Parameters	Acc (%)	#Parameters	Acc (%)
BiLSTM(40)+Atn	786,993	89.19	1,011,007	90.82
BiLSTM(20)+Atn	390,353	88.34	502,367	90.40
LSTM(40)+Atn	393,553	87.40	505,567	87.76
LSTM(20)+Atn	195,233	84.87	251,247	86.78
GRU(40)+Atn	391,953	77.45	503,967	80.44
BiLSTM(40)	17,648	87.76	18,782	88.78
BiLSTM(20)	5,648	86.84	6,222	87.76
LSTM(40)	8,848	83.87	9,422	75.51
LSTM(20)	2,848	82.47	3,142	69.39
GRU(40)	7,248	82.90	7,822	84.26

patterns of a longer sequence of data blocks. Finally, we decided to use BiLSTM with 40 neurons followed by an attention layer, as this attention-BiLSTM model provides the best performance.

Effect of CPD algorithm inaccuracy. This experiment assesses the effect of parameters in the CPD algorithm, which divides the trajectories into segments. Due to noisy sensor data, the number of segments detected for the same trajectory can be different across several samples of the trajectory. In total, there are 2148 segments in the testbeds, and more than 92% of these segments are identified correctly by our CPD algorithm.

To understand what happens when CPD does not identify segments correctly during inference, we evaluate the performance of place prediction using the training dataset from Testbed#1. In this experiment, we use a different test dataset with 29 trajectories, where the number of segments for each trajectory differs by $m \in [1, 2]$ from the number of segments in the training dataset. The result shows that accuracy of GoPlaces does not degrade significantly. For example, GoPlaces achieves

Table 5.6 Segment Classification and Place Prediction Accuracy for Different Threshold Values in DTW Algorithm

Threshold	L1 Classifier		L2 Classifier		Place Prediction
	#Segments	Acc(%)	#Segments	Acc(%)	p=80%
50	40	92.1	55	94.3	71.3
60	48	89.4	62	90.8	74.9
70	64	86.2	69	88.8	71.8
80	78	81.9	75	82.2	70.4
90	89	77.4	78	78.5	69.6

an accuracy of more than 77% at 85% of the traveled trajectory. The effects of lower or higher number of segments in the test trajectories are handled by the classifiers and the prediction tree. For example, a segment from A to B can be divided into two segments, AC and CB, which are not in the training dataset. However, as we use L2 segments, the concatenated version of AC and CB can be detected by L2 classifier. If some segments are misclassified, the prediction tree can still predict well because it considers all the possible combinations of segments to create paths from the root to the leaves.

Effect of threshold value for DTW algorithm. GoPlaces applies the DTW algorithm to detect identical segments. We experimented with different values of the threshold d_{th}, which result in different sets of similar segments. Table 5.6 shows the number of unique segments in the training dataset of Testbed#1 and the accuracy for different values of d_{th}. We observe that classification accuracy is higher for lower values of the threshold. This is because a higher threshold value sometimes fails to detect identical segments, which increases the number of segments unnecessarily. The place prediction accuracy, however, may suffer if the threshold value is too low because some segments can be wrongly considered duplicates and removed. This increases classifier accuracy, but drops the overall place prediction accuracy. Therefore, we choose $d_{th} = 60$ for the other experiments.

5.4.11 Performance on smart phones

We used GoPlaces to conduct data collection, training, and inference on two Android phone models (Google Pixel 3 and 4), and we measured latency, memory, and battery consumption. We also report the effect of ranging request frequency on segment identification and the effect of sampling data rates on classification and prediction accuracy. We report the results only for Testbed#1 because the differences between the testbeds are not significant.

Training performance. Before training, GoPlaces preprocesses the data, divides the trajectories into segments, and removes duplicated segments. Then, it assigns unique IDs to each segment, maps the trajectories onto a list of segments, and constructs the prediction tree. GoPlaces stores all these data locally in JSON format, which takes less than 600 KB. These steps take about three seconds. Next, GoPlaces trains the segment classifiers for L1 and L2 segments. To measure the training performance of the classifiers, we record the training time, memory and battery usage by training over 26536 samples for 10 epochs. We take 10 measurements and report the mean and standard deviation in Table 5.7. The training latency for one epoch is less than 10 minutes. The maximum RAM usage of the app during training is less than 300 MB. It takes 15% of battery to train both classifiers on Google Pixel 4 (with 2800 mAh Li-ion battery). The size of the models is less than 200 KB. These results show that training is feasible in terms of resource consumption. It is also worth noting that training is a one-time process, and the user needs to retrain the model only if they want to add new places or collect more data to improve accuracy.

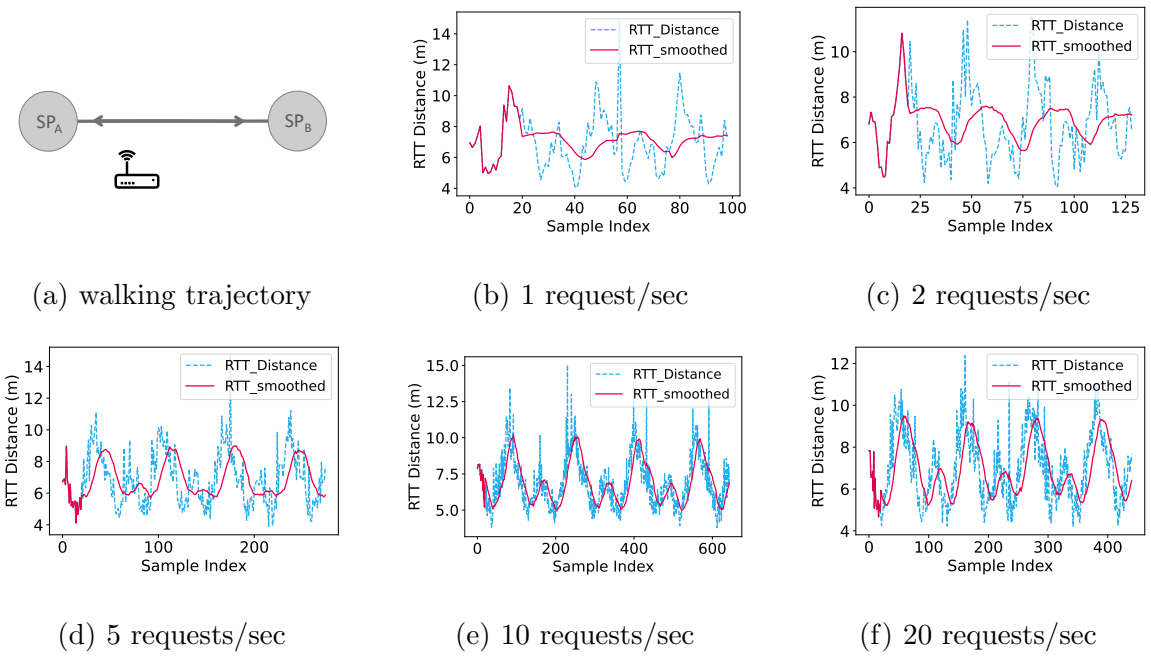
Inference performance. In this experiment, we measure the inference time and resource usage to predict IDs for 5000 segments and report the mean value of 10 measurements, as shown in Table 5.8. The overall place prediction task takes around 142 ms and uses less than 125 MB RAM. These results are usable for most practical app scenarios. We also observe that the most expensive operation during

Table 5.7 Training Resource Consumption and Latency

Phone Model	Training Time (seconds)		RAM (MB)	Battery (mAh)
	L1 Classifier (per epoch)	L2 Classifier (per epoch)		
Google Pixel 4	260 ± 25	175 ± 20	< 280	< 450
Google Pixel 3	590 ± 32	410 ± 21	< 300	< 550

Table 5.8 Inference Resource Consumption and Latency

Phone Model	Inference Time (milliseconds)			RAM (MB)	Battery (mAh)
	L1 Classifier	L2 Classifier	Overall Place Predictor		
Google Pixel 4	140 ± 3	136 ± 4	142 ± 4	< 125	< 30
Google Pixel 3	146 ± 2	141 ± 5	150 ± 3	< 125	< 35



(a) walking trajectory (b) 1 request/sec (c) 2 requests/sec

(d) 5 requests/sec (e) 10 requests/sec (f) 20 requests/sec

Figure 5.14 WiFi-RTT distance patterns for different ranging request frequencies.

inference is segment classification, with L1 and L2 classifiers taking 140 ms and 136 ms, respectively. Both phones can execute around 0.5 million predictions with a full battery.

Ranging request frequency for data collection. This experiment aims to set the sampling rate for WiFi-RTT to ensure accurate segment and trajectory

identification, while not consuming too much battery power. Figure 5.14 shows the raw and the smoothed WiFi-RTT measurement patterns for a trajectory ABABABABA between two places SP_A and SP_B for ranging frequency from one request/sec to 20 requests/sec. We observe that a frequency of 10 requests/sec is optimal, and we use it in all the other experiments. A lower value adds noise, and a higher value leads to more power consumption. At 10 requests/sec, GoPlaces uses $< 0.5\%$ of the battery per hour for Google Pixel 4 to collect data.

Data sampling rate for training and inference. Once the data is collected at the sampling rate necessary for accurate segment and trajectory identification, GoPlaces could downsample a sequence of data blocks for training or inference in order to save battery power during these processes. The aim of this experiment is to find a balance point for the data sampling rate where GoPlaces achieves good prediction accuracy, while saving battery power. Figure 5.15 shows the results, when varying the sampling rate from one sample/sec to 10 samples/sec. The results demonstrate that a sampling rate of four samples/sec achieves the best balance: the inference is substantially faster, which also means less battery consumption, while the accuracy is just slightly lower than the one achieved for the standard rate of 10 samples/sec. By lowering the sample rate, we are using fewer data blocks for a segment, which reduces the training and inference time. For example, GoPlaces takes 430 seconds per epoch for 10 samples/sec, whereas it requires 175 seconds per epoch for four samples/sec to train the model.

5.5 Chapter Summary

This chapter presents the design, implementation, and evaluation of GoPlaces, an app for personalized indoor place prediction. GoPlaces fuses inertial sensor data with WiFi-RTT estimated distances to predict the indoor places visited by a user. GoPlaces provides personalization by design to improve user experience and

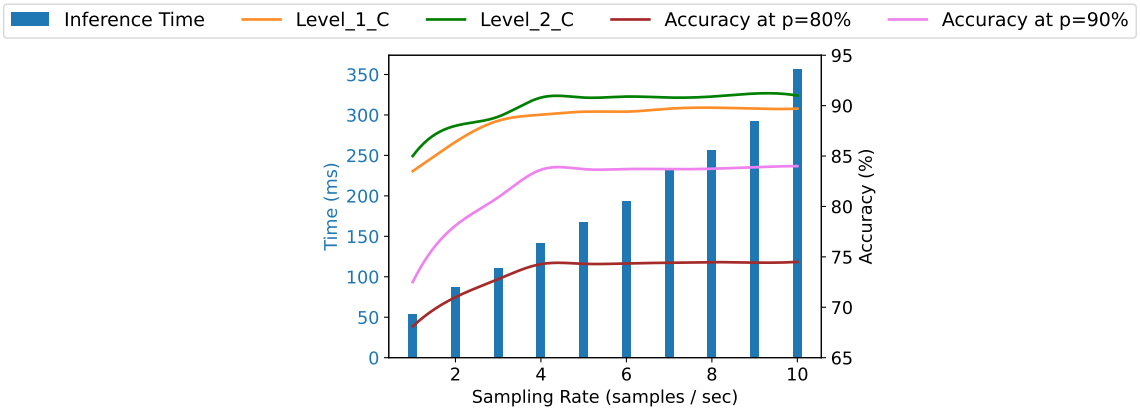


Figure 5.15 Inference time (bars) and accuracy (lines) of classifiers and place prediction for different sample rates.

prediction accuracy. Due to its use of a single AP and processing of all data on the user's phone, GoPlaces does not pose any privacy risks. GoPlaces is implemented as an Android app, and extensive experiments have demonstrated its feasibility in real life due to its low latency and resource consumption.

CHAPTER 6

CONCLUSIONS

Privacy-preserving machine learning holds great promise for enabling the development of machine learning models that can be applied to sensitive data. However, there are several challenges that must be addressed for the approach to be effective. These challenges include balancing privacy and accuracy, balancing privacy and usability, and the need for data collaboration. Addressing these challenges is crucial for ensuring that privacy-preserving machine learning can be used effectively and responsibly in a range of applications. In this dissertation, we discussed privacy risks in existing approaches across three scenarios and propose systems that enhance privacy, utility, and efficiency.

Firstly, we introduced FedMTL, a novel FL aggregation technique that enables clients to obtain improved personalized MTL models for their sets of tasks by collaborating with other clients. FedMTL computes aggregation weights for each client by analyzing the parameters of task-specific layers in MTL models and applies a layer-wise aggregation policy on the models from participating clients. The FedMTL aggregation can be integrated with established privacy-preserving techniques for secure aggregation, thus guaranteeing the privacy of clients' private data. The experimental results demonstrated that FedMTL outperforms state-of-the-art FL aggregation approaches and can work in cases where clients are involved in different sets of tasks. We also implemented a secure version of FedMTL using secret-sharing SMPC, which achieves the same accuracy performance as plain text while preserving the privacy of client data.

Secondly, we described CryptGNN, a provably secure and effective solution for GNN models in the cloud. CryptGNN uses novel SMPC protocols to preserve the

privacy of the model weights and of the graph input data of the clients, including the graph structure, while providing the same results with a non-secure inference version. CryptGNN works with an arbitrary number of SMPC parties, and the input data, the intermediate results, and the output are protected in secret-shared format even if $P-1$ out of P parties collude with each other. We evaluated the performance of CryptGNN with different types of graphs and configurations. The experimental results demonstrate its correctness and low execution time compared to other approaches. In the future, CryptGNN can be extended to support the message passing of GNN with different aggregation and update mechanisms.

Finally, we presented GoPlaces, a privacy-preserving app that predicts the user's next location in indoor spaces by fusing phone sensors and WiFi-RTT data. GoPlaces does not require complex infrastructure for accurate localization and, therefore, can work in many places and can easily be deployed on smart phones. GoPlaces is also designed to provide personalization and mitigate privacy risks, which further enhances its practicality. The experimental results demonstrate good accuracy, low-latency, and low resource consumption on the phones. The prediction accuracy of GoPlaces can be further improved by incorporating the user's behavioral and temporal data like wait time at some place, day of the week, or time of day in the prediction tree, as it can capture the user's trend information. By employing techniques such as personalized federated learning, model accuracy can be improved in a way that uses data from all users, while still performing personalization and protecting location privacy. In the future, we will investigate the possibility of using GoPlaces in spaces that are larger than the transmission range of one AP, which is a limitation of the current solution. This can be done in buildings with multiple APs by assigning one space for each AP and designing a transition algorithm among adjacent places.

REFERENCES

[1] A. Nawrocka, A. Kot, and M. Nawrocki, "Application of machine learning in recommendation systems," in *2018 19th International Carpathian Control Conference (ICCC)*, 2018.

[2] I. Kononenko, I. Bratko, and M. Kukar, "Application of machine learning to medical diagnosis," *Machine Learning and Data Mining: Methods and Applications*, vol. 389, p. 408, 1997.

[3] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, vol. 207, p. 117921, 2022.

[4] S. Mohseni, M. Pitale, V. Singh, and Z. Wang, "Practical solutions for machine learning safety in autonomous vehicles," *arXiv preprint arXiv:1912.09630*, 2019.

[5] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, and A. V. Vasilakos, "Privacy and security issues in deep learning: A survey," *IEEE Access*, vol. 9, pp. 4566–4593, 2021.

[6] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[7] Z. Ji, Z. C. Lipton, and C. Elkan, "Differential privacy and machine learning: a survey and review," *Computing Research Repository (CoRR)*, vol. abs/1412.7584, 2014. [Online]. Available: http://arxiv.org/abs/1412.7584

[8] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, 2016.

[9] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[10] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic encryption for machine learning in medicine and bioinformatics," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, August 2020.

[11] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 4961–4973.

[12] S. Sayyad, "Privacy preserving deep learning using secure multiparty computation," in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2020.

[13] M. S. Alvim, M. E. Andrés, K. Chatzikokolakis, P. Degano, and C. Palamidessi, "Differential privacy: On the trade-off between utility and information leakage." *Formal Aspects in Security and Trust*, vol. 7140, pp. 39–54, 2011.

[14] D. Das, "Secure cloud computing algorithm using homomorphic encryption and multi-party computation," in *2018 International Conference on Information Networking (ICOIN)*, 2018.

[15] Eclipse, "Deeplearning4j Suite Overview," https://deeplearning4j.konduit.ai/, Accessed on 02/10/2025.

[16] TensorFlow Blog, "On-device training in TensorFlow Lite," https://blog.tensorflow.org/2021/11/on-device-training-in-tensorflow-lite.html, Accessed on 02/10/2025.

[17] H. Zheng and H. Hu, "Missile: A system of mobile inertial sensor-based sensitive indoor location eavesdropping," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3137–3151, 2020.

[18] Y.-A. Montjoye, C. Hidalgo, M. Verleysen, and V. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Scientific Reports*, vol. 3, p. 1376, March 2013.

[19] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *Computing Research Repository (CoRR)*, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[20] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[21] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, July 1997.

[22] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017.

[23] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[24] R. Li, F. Ma, W. Jiang, and J. Gao, "Online federated multitask learning," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019.

[25] J. Geng, Y. Mou, F. Li, Q. Li, O. Beyan, S. Decker, and C. Rong, "Towards general deep leakage in federated learning," *arXiv preprint arXiv:2110.09074*, 2021.

[26] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, 2017.

[27] Y. Dong, X. Chen, L. Shen, and D. Wang, "Eastfly: Efficient and secure ternary federated learning," *Computers & Security*, vol. 94, p. 101824, 2020.

[28] A.-T. Tran, T.-D. Luong, J. Karnjana, and V.-N. Huynh, "An efficient approach for privacy preserving decentralized deep learning models based on secure multi-party computation," *Neurocomputing*, vol. 422, pp. 245–262, 2021.

[29] C. Zhao, S. Zhao, M. Zhao, Z. Chen, C.-Z. Gao, H. Li, and Y. an Tan, "Secure multi-party computation: Theory, practice and applications," *Information Sciences*, vol. 476, pp. 357–372, 2019.

[30] D. F. Nettleton, "Survey: Data mining of social networks represented as graphs," *Computer Science Review*, vol. 7, p. 1–34, February 2013. [Online]. Available: https://doi.org/10.1016/j.cosrev.2012.12.001

[31] B. Lu, X. Gan, H. Jin, L. Fu, and H. Zhang, "Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, ser. CIKM '20, 2020.

[32] X. Wang, Y. Ma, Y. Wang, W. Jin, X. Wang, J. Tang, C. Jia, and J. Yu, "Traffic flow prediction via spatial temporal graph neural network," in *Proceedings of The Web Conference 2020*, ser. WWW '20, 2020.

[33] Y. Pei, F. Lyu, W. van Ipenburg, and M. Pechenizkiy, "Subgraph anomaly detection in financial transaction networks," in *Proceedings of the First ACM International Conference on AI in Finance*, ser. ICAIF '20, 2021.

[34] C. B. Bruss, A. Khazane, J. Rider, R. Serpe, A. Gogoglou, and K. E. Hines, "Deeptrax: Embedding graphs of financial transactions," *Computing Research Repository (CoRR)*, vol. abs/1907.07225, 2019. [Online]. Available: http://arxiv.org/abs/1907.07225

[35] L. Alrahis, J. Knechtel, and O. Sinanoglu, "Graph neural networks: A powerful and versatile tool for advancing design, reliability, and security of ics," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023.

[36] K. Liu, X. Sun, L. Jia, J. Ma, H. Xing, J. Wu, H. Gao, Y. Sun, F. Boulnois, and J. Fan, "Chemi-net: A molecular graph convolutional network for accurate drug property prediction," *International Journal of Molecular Sciences*, vol. 20, no. 14, 2019.

[37] I. Fischer and T. Meinl, "Graph based molecular data mining - an overview," in *2004 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, 2004.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[39] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017.

[40] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019.

[41] J. Li, D. Cai, and X. He, "Learning graph-level representation for drug discovery," *arXiv preprint arXiv:1709.03741*, 2017.

[42] Z. Wang, M. Liu, Y. Luo, Z. Xu, Y. Xie, L. Wang, L. Cai, Q. Qi, Z. Yuan, T. Yang, and S. Ji, "Advanced graph and sequence neural networks for molecular property prediction and drug discovery," *Bioinformatics*, vol. 38, no. 9, pp. 2579–2586, February 2022.

[43] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *2019 IEEE International Conference on Data Mining (ICDM)*, 2019.

[44] D. Cheng, X. Wang, Y. Zhang, and L. Zhang, "Graph neural network for fraud detection via spatial-temporal attention," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3800–3813, 2022.

[45] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, ser. WWW '19, 2019.

[46] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: A survey," *ACM Computing Surveys*, March 2022. [Online]. Available: https://doi.org/10.1145/3535101

[47] K. Guo, Y. Hu, Z. Qian, H. Liu, K. Zhang, Y. Sun, J. Gao, and B. Yin, "Optimized graph convolution recurrent neural network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 1138–1149, 2021.

[48] C. Chen, K. Li, S. G. Teo, X. Zou, K. Wang, J. Wang, and Z. Zeng, "Gated residual recurrent graph neural networks for traffic prediction," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[49] M. Ribeiro, K. Grolinger, and M. A. Capretz, "MLaaS: Machine learning as a service," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015.

[50] Amazon, "Amazon Rekognition," https://aws.amazon.com/rekognition/, Accessed on 02/10/2025.

[51] Amazon AWS, "Amazon Forecast," https://aws.amazon.com/forecast/, Accessed on 02/10/2025.

[52] Microsoft, "Azure Machine Learning," https://azure.microsoft.com/en-us/services/machine-learning, Accessed on 02/10/2025.

[53] Google, "Discover insights from text with AutoML natural language, now generally available," https://cloud.google.com/blog/products/ai-machine-learning/machine-learning-feature-automl-natural-language-generally-available, Accessed on 02/10/2025.

[54] Microsoft, "AI anomaly detector," https://azure.microsoft.com/en-us/products/ai-services/ai-anomaly-detector/#overview, Accessed on 02/10/2025.

[55] L. Liu, H. Nguyen, G. Karypis, and S. H. Sengamedu, "Universal representation for code," in *PAKDD*, 2021.

[56] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, August 2018.

[57] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18, 2018.

[58] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, 2017.

[59] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, 2016.

[60] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[61] J. Alvarez-Valle, P. Bhatu, N. Chandran, D. Gupta, A. V. Nori, A. Rastogi, M. Rathee, R. Sharma, and S. Ugare, "Secure medical image analysis with cryptflow," *ArXiv*, vol. abs/2012.05064, 2020.

[62] A. Soin, P. Bhatu, R. Takhar, N. Chandran, D. Gupta, J. Alvarez-Valle, R. Sharma, V. Mahajan, and M. P. Lungren, "Multi-institution encrypted medical imaging ai validation without data sharing," arXiv, August 2021. [Online]. Available: https://arxiv.org/abs/2107.10230

[63] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience (extended abstract)," in *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '94, 1994.

[64] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, vol. 64, no. 1, pp. 86–96, 2020.

[65] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[66] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, ser. PPMLP'20, 2020.

[67] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: Efficient and private neural network training," Cryptology ePrint Archive, Paper 2018/442, 2018, https://eprint.iacr.org/2018/442.

[68] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," *Computing Research Repository (CoRR)*, vol. abs/2104.10949, 2021. [Online]. Available: https://arxiv.org/abs/2104.10949

[69] W.-q. Huang, C. Ding, S.-y. Wang, and S. Hu, "An efficient clustering mining algorithm for indoor moving target trajectory based on the improved agnes," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015.

[70] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.

[71] M. Alarfaj, Z. su, R. Liu, A. Alhumam, and H. Liu, "Image-tag-based indoor localization using end-to-end learning," *International Journal of Distributed Sensor Networks*, vol. 17, p. 155014772110523, November 2021.

[72] A. Poulose and D. Han, "UWB indoor localization using deep learning LSTM networks," *Applied Sciences*, vol. 10, p. 6290, September 2020.

[73] N. D. R. Rose, L. T. Jung, and M. Ahmad, "3D trilateration localization using RSSI in indoor environment," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 2, 2020.

[74] M. Youssef and A. Agrawala, "The horus WLAN location determination system," in *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '05, 2005.

[75] S. Yoon, K. Lee, and I. Rhee, "FM-based indoor localization via automatic fingerprint db construction and matching," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13, 2013.

[76] L. Ni, Y. Liu, Y. C. Lau, and A. Patil, "LANDMARC: Indoor location sensing using active RFID," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)*, 2003.

[77] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Indoor localization without infrastructure using the acoustic background spectrum," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11, 2011.

[78] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara, "Accurate GSM indoor localization," in *UbiComp 2005: Ubiquitous Computing*, M. Beigl, S. Intille, J. Rekimoto, and H. Tokuda, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[79] M. Azizyan, I. Constandache, and R. Roy Choudhury, "SurroundSense: Mobile phone localization via ambience fingerprinting," in *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '09, 2009.

[80] J. Chung, M. Donahoe, C. Schmandt, I.-J. Kim, P. Razavai, and M. Wiseman, "Indoor location sensing using geo-magnetism," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11, 2011.

[81] H. Li, L. Sun, H. Zhu, X. Lu, and X. Cheng, "Achieving privacy preservation in WiFi fingerprint-based localization," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014.

[82] N. N. Neto, S. Madnick, A. M. G. D. Paula, and N. M. Borges, "Developing a global data breach database and the challenges encountered," *Journal of Data and Information Quality (JDIQ)*, vol. 13, no. 1, January 2021. [Online]. Available: https://doi.org/10.1145/3439873

[83] Electric, "High-profile company data breaches 2023," https://www.electric.ai/blog/recent-big-company-data-breaches, Accessed on 02/10/2025.

[84] TFEncrypted, "Encrypted deep learning in tensorflow," https://tf-encrypted.io/, Accessed on 02/10/2025.

[85] Microsoft, "Microsoft SEAL," https://www.microsoft.com/en-us/research/project/microsoft-seal/, Accessed on 02/10/2025.

[86] OpenMined, "PySyft," https://blog.openmined.org/tag/pysyft/, Accessed on 02/10/2025.

[87] M. AI, "CrypTen: A new research tool for secure machine learning with PyTorch," https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch/, Accessed on 02/10/2025.

[88] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *Computing Research Repository (CoRR)*, vol. abs/1803.05961, 2018. [Online]. Available: http://arxiv.org/abs/1803.05961

[89] V. Chen, V. Pastro, and M. Raykova, "Secure computation for machine learning with SPDZ," *Computing Research Repository (CoRR)*, vol. abs/1901.00329, 2019. [Online]. Available: http://arxiv.org/abs/1901.00329

[90] S. Dhar, J. Guo, J. J. Liu, S. Tripathi, U. Kurup, and M. Shah, "A survey of on-device machine learning: An algorithms and learning theory perspective," *ACM Transactions on Internet of Things*, vol. 2, no. 3, July 2021. [Online]. Available: https://doi.org/10.1145/3450494

[91] Apple, "Core ML," https://developer.apple.com/machine-learning/core-ml/, Accessed on 02/10/2025.

[92] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[93] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[94] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[95] H. Hu, Z. Salcic, G. Dobbie, and X. Zhang, "Membership inference attacks on machine learning: A survey," *Computing Research Repository (CoRR)*, vol. abs/2103.07853, 2021. [Online]. Available: https://arxiv.org/abs/2103.07853

[96] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, 2015.

[97] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, 2016.

[98] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: Improved deep leakage from gradients," *Computing Research Repository (CoRR)*, vol. abs/2001.02610, 2020. [Online]. Available: http://arxiv.org/abs/2001.02610

[99] R. Xu, N. Baracaldo, and J. Joshi, "Privacy-preserving machine learning: Methods, challenges and directions," *Computing Research Repository (CoRR)*, vol. abs/2108.04417, 2021. [Online]. Available: https://arxiv.org/abs/2108.04417

[100] B. McCann, N. S. Keskar, C. Xiong, and R. Socher, "The natural language decathlon: Multitask learning as question answering," *arXiv preprint arXiv:1806.08730*, 2018.

[101] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International Conference on Machine Learning*. PMLR, 2018.

[102] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[103] A. C. Stickland and I. Murray, "BERT and pals: Projected attention layers for efficient adaptation in multi-task learning," *Computing Research Repository (CoRR)*, vol. abs/1902.02671, 2019. [Online]. Available: http://arxiv.org/abs/1902.02671

[104] L. Pascal, P. Michiardi, X. Bost, B. Huet, and M. A. Zuluaga, "Maximum roaming multi-task learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021.

[105] J. Liu, J. Wu, J. Chen, M. Hu, Y. Zhou, and D. Wu, "FedDWA: Personalized federated learning with dynamic weight adjustment," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, August 2023.

[106] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 586–19 597, 2020.

[107] M. Zhang, K. Sapra, S. Fidler, S. Yeung, and J. M. Alvarez, "Personalized federated learning with first order model optimization," *arXiv preprint arXiv:2012.08565*, 2020.

[108] J. Chen and A. Zhang, "FedMSplit: Correlation-adaptive federated multi-task learning across multimodal split networks," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22, 2022.

[109] C. T. Dinh, T. T. Vu, N. H. Tran, M. N. Dao, and H. Zhang, "A new look and convergence rate of federated multitask learning with laplacian regularization," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022.

[110] O. Marfoq, G. Neglia, A. Bellet, L. Kameni, and R. Vidal, "Federated multi-task learning under a mixture of distributions," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021.

[111] R. Cai, X. Chen, S. Liu, J. Srinivasa, M. Lee, R. Kompella, and Z. Wang, "Many-task federated learning: A new problem setting and a simple baseline," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2023.

[112] W. Zhuang, Y. Wen, L. Lyu, and S. Zhang, "MAS: Towards resource-efficient federated multiple-task learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[113] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[114] Z. Ying, Y. Zhang, and X. Liu, "Privacy-preserving in defending against membership inference attacks," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, 2020.

[115] J. So, B. Guler, and A. Avestimehr, "Turbo-Aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE Journal on Selected Areas in Information Theory*, vol. PP, pp. 1–1, January 2021.

[116] Z. Zhang, J. Li, S. Yu, and C. Makaya, "SAFELearning: Secure aggregation in federated learning with backdoor detectability," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3289–3304, 2023.

[117] J. Shu, T. Yang, X. Liao, F. Chen, Y. Xiao, K. Yang, and X. Jia, "Clustered federated multitask learning on non-iid data with enhanced privacy," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3453–3467, 2023.

[118] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure multiparty computation and secret sharing*. Cambridge University Press, 2015.

[119] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999.

[120] Google Cloud, "AI platform training & prediction API (training)," https://cloud.google.com/ai-platform/training/docs/reference/rest, Accessed on 01/15/2025.

[121] Microsoft, "Azure machine learning," https://azure.microsoft.com/en-us/products/machine-learning/, Accessed on 01/15/2025.

[122] IBM, "Machine learning service instances," https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-service-instance.html?context=dph, Accessed on 01/15/2025.

[123] Amazon, "Machine Learning on AWS," https://aws.amazon.com/machine-learning/, Accessed on 01/15/2025.

[124] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18, 2018.

[125] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[126] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," *ArXiv*, vol. abs/1805.08090, 2018.

[127] S. Wang, Y. Zheng, and X. Jia, "SecGNN: Privacy-preserving graph neural network training and inference as a cloud service," *IEEE Transactions on Services Computing*, vol. 16, no. 04, pp. 2923–2938, July 2023.

[128] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, ser. Lecture Notes in Computer Science, vol. 576. Springer, 1991.

[129] R. Ran, W. Wang, Q. Gang, J. Yin, N. Xu, and W. Wen, "CryptoGCN: Fast and scalable homomorphically encrypted graph convolutional network inference," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 37 676–37 689.

[130] H. Chen and K. Han, "Homomorphic lower digits removal and improved fhe bootstrapping," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018.

[131] L. Chen, Z. Zhang, and X. Wang, "Batched multi-hop multi-key fhe from ring-lwe with compact ciphertext extension," in *Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2017.

[132] T. Hoang, C. D. Ozkaptan, A. A. Yavuz, J. Guajardo, and T. Nguyen, "S3ORAM: A computation-efficient and constant client bandwidth blowup oram with shamir secret sharing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, 2017.

[133] E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," *Journal of the ACM (JACM)*, vol. 65, no. 4, April 2018. [Online]. Available: https://doi.org/10.1145/3177872

[134] H. Chen, I. Chillotti, and L. Ren, "Onion ring ORAM: Efficient constant bandwidth oblivious RAM from (leveled) TFHE," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, 2019.

[135] H. Chen, M. Kim, I. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh, "Maliciously secure matrix multiplication with applications to private deep learning," Cryptology ePrint Archive, Paper 2020/451, 2020, https://eprint.iacr.org/2020/451.

[136] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, August 2019.

[137] J. Huang, D. Dahlmeier, Z. Lin, B.-K. Ang, M.-L. Seeto, and H. Shi, "Wi-Fi based indoor next location prediction using mixed state-weighted markov-chain model," *International Journal of Machine Learning and Computing*, vol. 4, pp. 505–509, January 2014.

[138] P. Wang, J. Yang, J. Zhang, and C. So-In, "A spatial-contextual indoor trajectory prediction approach via hidden markov models," *Wireless Communications and Mobile Computing*, vol. 2022, January 2022. [Online]. Available: https://doi.org/10.1155/2022/6719514

[139] P. Wang, H. Wang, H. Zhang, F. Lu, and S. Wu, "A hybrid markov and LSTM model for indoor location prediction," *IEEE Access*, vol. 7, pp. 185 928–185 940, 2019.

[140] P. Wang, J. Yang, and J. Zhang, "Indoor trajectory prediction for shopping mall via sequential similarity," *Information*, vol. 13, no. 3, 2022.

[141] O. O. Barzaiq, S. W. Loke, and H. Lu, "On trajectory prediction in indoor retail environments for mobile advertising using selected self-histories," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, 2015.

[142] K. Nguyen and Z. Luo, "Reliable indoor location prediction using conformal prediction," *Annals of Mathematics and Artificial Intelligence*, vol. 74, June 2013.

[143] M. Dash, K. K. Koo, J. B. Gomes, S. P. Krishnaswamy, D. Rugeles, and A. Shi-Nash, "Next place prediction by understanding mobility patterns," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2015.

[144] X. Jiang, S. Zhao, G. Jacobson, R. Jana, W.-L. Hsu, M. Talasila, S. A. Aftab, Y. Chen, and C. Borcea, "Federated meta-location learning for fine-grained location prediction," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021.

[145] S. Gambs, M.-O. Killijian, and M. N. n. del Prado Cortez, "Next place prediction using mobility markov chains," in *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, ser. MPM '12, 2012.

[146] F. Khoroshevsky and B. Lerner, "Human mobility-pattern discovery and next-place prediction from GPS data," in *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*. Springer, 2017.

[147] C. Schreckenberger, S. Beckmann, and C. Bartelt, "Next place prediction: A systematic literature review," in *Proceedings of the 2nd ACM SIGSPATIAL Workshop on Prediction of Human Mobility*, ser. PredictGIS, 2018.

[148] F. Khoroshevsky and B. Lerner, "Human mobility-pattern discovery and next-place prediction from gps data," in *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*, F. Schwenker and S. Scherer, Eds. Cham: Springer International Publishing, 2017, pp. 24–35.

[149] E. Soltanaghaei, A. Kalyanaraman, and K. Whitehouse, "Multipath triangulation: Decimeter-level WiFi localization and orientation with a single unaided receiver," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18, 2018.

[150] M. Xue, W. Sun, H. Yu, H. Tang, A. Lin, X. Zhang, and R. Zimmermann, "Locate the mobile device by enhancing the WiFi-based indoor localization model," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8792–8803, 2019.

[151] W. Gong and J. Liu, "SiFi: Pushing the limit of time-based WiFi localization using a single commodity access point," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, March 2018. [Online]. Available: https://doi.org/10.1145/3191742

[152] J.-Y. Wang, C.-P. Chen, T.-S. Lin, C.-L. Chuang, T.-Y. Lai, and J.-A. Jiang, "High-precision RSSI-based indoor localization using a transmission power adjustment strategy for wireless sensor networks," in *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, 2012.

[153] S. Sadowski and P. Spachos, "RSSI-based indoor localization with the internet of things," *IEEE Access*, vol. 6, pp. 30 149–30 161, 2018.

[154] J. Wisanmongkol, L. Klinkusoom, T. Sanpechuda, L.-o. Kovavisaruch, and K. Kaemarungsi, "Multipath mitigation for RSSI-based bluetooth low energy localization," in *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*, 2019.

[155] N. Singh, S. Choe, and R. Punmiya, "Machine learning based indoor localization using Wi-Fi RSSI fingerprints: An overview," *IEEE Access*, vol. 9, pp. 127 150–127 174, 2021.

[156] A. T. Mariakakis, S. Sen, J. Lee, and K.-H. Kim, "SAIL: Single access point-based indoor localization," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14, 2014.

[157] W. Wang, Z. Liu, J. Gao, N. Saoda, and B. Campbell, "UbiTrack: Enabling scalable & low-cost device localization with onboard WiFi," in *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, ser. BuildSys '21, 2021.

[158] M. Rea, T. E. Abrudan, D. Giustiniano, H. Claussen, and V.-M. Kolmonen, "Smartphone positioning with radio measurements from a single WiFi access point," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19, 2019.

[159] D. Vasisht, S. Kumar, and D. Katabi, "Decimeter-Level Localization with a Single WiFi Access Point," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'16. USA: USENIX Association, 2016, p. 165–178.

[160] C. Zhang, F. Li, J. Luo, and Y. He, "ilocscan: Harnessing multipath for simultaneous indoor source localization and space scanning," November 2014.

[161] Z. Chen, G. Zhu, S. Wang, Y. Xu, J. Xiong, J. Zhao, J. Luo, and X. Wang, "m^3m3: Multipath assisted wi-fi localization with a single access point," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 588–602, 2021.

[162] M. T. Hoang, B. Yuen, K. Ren, X. Dong, T. Lu, R. Westendorp, and K. Reddy, "A cnn-lstm quantifier for single access point csi indoor localization," *ArXiv*, vol. abs/2005.06394, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218614108

[163] Z. Deng, J. Wu, S. Wang, M. Zhang, and X. Liu, "Indoor localization with a single access point based on tdoa and aoa," *Wirel. Commun. Mob. Comput.*, vol. 2022, Jan. 2022. [Online]. Available: https://doi.org/10.1155/2022/9526532

[164] A. Xiao, R. Chen, D. Li, Y. Chen, and D. Wu, "An indoor positioning system based on static objects in large indoor scenes by using smartphone cameras," *Sensors*, vol. 18, p. 2229, July 2018.

[165] A. Mulloni, D. Wagner, I. Barakonyi, and D. Schmalstieg, "Indoor positioning and navigation with camera phones," *IEEE Pervasive Computing*, vol. 8, no. 2, pp. 22–31, 2009.

[166] G. Yang and J. Saniie, "Indoor navigation for visually impaired using AR markers," in *2017 IEEE International Conference on Electro Information Technology (EIT)*, 2017.

[167] Y. Zhou, G. Li, L. Wang, S. Li, and W. Zong, "Smartphone-based pedestrian localization algorithm using phone camera and location coded targets," in *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, 2018.

[168] K. Yu, K. Wen, Y. Li, S. Zhang, and K. Zhang, "A novel NLOS mitigation algorithm for UWB localization in harsh indoor environments," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 686–699, 2019.

[169] Y.-T. Wang, R. Zheng, and D. Zhao, "Towards zero-configuration indoor localization using asynchronous acoustic beacons," in *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. IEEE, 2016.

[170] J. Bordoy, C. Schindelhauer, F. Höflinger, and L. M. Reindl, "Exploiting acoustic echoes for smartphone localization and microphone self-calibration," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 4, pp. 1484–1492, 2020.

[171] F. Höflinger, R. Zhang, J. Hoppe, A. Bannoura, L. M. Reindl, J. Wendeberg, M. Bührer, and C. Schindelhauer, "Acoustic self-calibrating system for indoor smartphone tracking (ASSIST)," in *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2012.

[172] A. Kulkarni and A. Lim, "Preliminary study on indoor localization using smartphone-based IEEE 802.11mc," in *Proceedings of the 15th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '19 Companion, 2019.

[173] J. Cobb., "Testing WiFi RTT on Android P for indoor positioning." https://www.crowdconnected.com/blog/testing-wifi-rtt-on-android-p-for-indoor-positioning/, Accessed on 02/10/2025.

[174] B. K. Horn, "Doubling the accuracy of indoor positioning: Frequency diversity," *Sensors*, vol. 20, no. 5, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/5/1489

[175] B. K. P. Horn, "Observation model for indoor positioning," *Sensors*, vol. 20, no. 14, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/14/4027

[176] M. Ibrahim, H. Liu, M. Jawahar, V. Nguyen, M. Gruteser, R. Howard, B. Yu, and F. Bai, "Verification: Accuracy evaluation of WiFi fine time measurements on an open platform," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18, 2018.

[177] R. Zhang, F. Höflinger, and L. Reindl, "Inertial sensor based indoor localization and monitoring system for emergency responders," *Sensors Journal, IEEE*, vol. 13, pp. 838–848, February 2013.

[178] J. Choi, G. Lee, S. Choi, and S. Bahk, "Smartphone based indoor path estimation and localization without human intervention," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 681–695, 2022.

[179] J. Jiang, C. Pan, H. Liu, and G. Yang, "Predicting human mobility based on location data modeled by Markov chains," in *2016 Fourth International Conference on Ubiquitous Positioning, Indoor Navigation and Location Based Services (UPINLBS)*, 2016.

[180] W. Mathew, R. Raposo, and B. Martins, "Predicting future locations with hidden markov models," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ser. UbiComp '12, 2012.

[181] A. B. Adege, H.-P. Lin, G. B. Tarekegn, and S.-S. Jeng, "Applying deep neural network (DNN) for robust indoor localization in multi-building environment," *Applied Sciences*, vol. 8, no. 7, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/7/1062

[182] C.-H. Hsieh, J.-Y. Chen, and B.-H. Nien, "Deep learning-based indoor localization using received signal strength and channel state information," *IEEE Access*, vol. 7, pp. 33 256–33 267, 2019.

[183] C. Chen, P. Zhao, C. X. Lu, W. Wang, A. Markham, and N. Trigoni, "Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4431–4441, 2020.

[184] Q. Wang, H. Luo, J. Wang, L. Sun, Z. Ma, C. Zhang, M. Fu, and F. Zhao, "Recent advances in pedestrian navigation activity recognition: A review," *IEEE Sensors Journal*, vol. 22, no. 8, pp. 7499–7518, 2022.

[185] D. Schepers and A. Ranganathan, "Privacy-preserving positioning in Wi-Fi fine timing measurement," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, pp. 325–343, April 2022.

[186] C. Chen, X. Lu, A. Markham, and N. Trigoni, "Ionet: learning to cure the curse of drift in inertial odometry," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[187] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[188] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[189] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[190] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[191] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," *IACR Cryptology ePrint Archive*, vol. 2011, p. 535, January 2011.

[192] D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," in *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, ser. Lecture Notes in Computer Science, vol. 293. Springer, 1987.

[193] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, "Personalized cross-silo federated learning on non-iid data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021.

[194] C. T. Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 394–21 405.

[195] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits," in *Computer Security – ESORICS 2013*, J. Crampton, S. Jajodia, and K. Mayes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–18.

[196] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *Computer Vision – ECCV*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 746–760.

[197] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[198] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[199] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[200] O. Catrina and S. Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*. Springer, 2010.

[201] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[202] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. CONF, pp. 291–311, 2021.

[203] M. Keller, E. Orsini, and P. Scholl, "Mascot: faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[204] L. Xiong, W. Zhou, Z. Xia, Q. Gu, and J. Weng, "Efficient privacy-preserving computation based on additive secret sharing," *arXiv preprint arXiv:2009.05356*, 2020.

[205] E.E.A. .com, "Modular multiplicative inverse," https://www.extendedeuclideanalgorithm.com/multiplicative_inverse.php, Accessed on 02/10/2025.

[206] H. Ghodosi, J. Pieprzyk, and R. Steinfeld, "Multi-party computation with conversion of secret sharing," *Designs, Codes and Cryptography*, vol. 62, pp. 259–272, 2012.

[207] Z. Xia, Q. Gu, W. Zhou, L. Xiong, J. Weng, and N. Xiong, "STR: Secure computation on additive shares using the share-transform-reveal strategy," *IEEE Transactions on Computers*, pp. 1–1, 2021.

[208] M. Blanton, A. Kang, and C. Yuan, "Improved building blocks for secure multi-party computation based on secret sharing with honest majority," in *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2020.

[209] M. Haiyan, Y. Jinyao, P. Georgopoulos, and B. Plattner, "Towards SDN based queuing delay estimation," *China Communications*, vol. 13, no. 3, pp. 27–36, 2016.

[210] S. Ivanov, S. Sviridov, and E. Burnaev, "Understanding isomorphism bias in graph data sets," *Computing Research Repository (CoRR)*, vol. abs/1910.12091, 2019. [Online]. Available: http://arxiv.org/abs/1910.12091

[211] F. Bogo, J. Romero, M. Loper, and M. J. Black, "FAUST: Dataset and evaluation for 3D mesh registration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[212] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International Conference on Machine Learning*. PMLR, 2016.

[213] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[214] Y. Xuan, R. Sengupta, and Y. Fallah, "Making indoor maps with portable accelerometer and magnetometer," in *2010 Ubiquitous Positioning Indoor Navigation and Location Based Service*, 2010.

[215] B. K. Horn, "Indoor positioning using time of flight with respect to wifi access points," https://people.csail.mit.edu/bkph/FTMRTT, Accessed on 02/10/2025.

[216] S. Aminikhanghahi and D. Cook, "A survey of methods for time series change point detection," *Knowledge and Information Systems*, vol. 51, May 2017.

[217] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *KDD Workshop*, 1994.

[218] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[219] J. Xie, B. Chen, X. Gu, F. Liang, and X. Xu, "Self-attention-based BiLSTM model for short text fine-grained sentiment classification," *IEEE Access*, vol. 7, pp. 180 558–180 570, 2019.

[220] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural Computation*, vol. 8, no. 3, pp. 643–674, 1996.