ENHANCING DATA REPRESENTATION IN DISTRIBUTED MACHINE LEARNING

SUPERVISORS:
DR. CHRISTOS ANAGNOSTOPOULOS
DR. FANI DELIGIANNI

TAHANI ALADWANI

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING SCIENCE
COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

SEPTEMBER 30, 2024

Abstract

Distributed computing devices, ranging from smartphones to edge micro-servers—collectively referred to as *clients*—are capable of gathering and storing diverse types of data, such as images and voice recordings. This wide array of data sources has the potential to significantly enhance the accuracy and robustness of Deep Learning (DL) models across a variety of tasks. However, this data is intrinsically heterogeneous, due to the differences in users' preferences, lifestyles, locations, and other factors. Consequently, it necessitates comprehensive preprocessing (e.g., labeling, filtering, relevance assessment, balancing, etc.) to ensure its suitability for the development of effective and reliable models. Therefore, this thesis explores the feasibility of conducting predictive analytics and model inference on edge computing (EC) systems when access to data is limited, and on clients' devices through federated learning (FL) when direct access to data is entirely restricted.

The first part of this thesis focuses on reducing the data transmission rate between clients and EC servers by employing techniques such as data and task caching, identifying data overlaps, and evaluating task popularity. While this strategy can significantly minimize data offloading to the lowest possible level, it does not entirely eliminate dependence on third-party entities.

The second part of this thesis eliminates the dependency on third-party entities by implementing FL, where direct access to raw data is not possible. In this context, node and data selection are guided by predictions and model performance. The objective is to identify the most suitable nodes and relevant data for training by clustering nodes based on data characteristics and analyzing the overlap between query boundaries and cluster boundaries.

The third part of this thesis introduces a mechanism designed to support classification tasks, such as image classification. These tasks present significant challenges when building models on distributed data, particularly due to issues like label shifting or missing labels across clients. To address these challenges, the proposed method mitigates the impact of imbalances across clients by employing multiple cluster-based meta-models, each tailored to specific label distributions.

The fourth part of this thesis introduces a two-phase federated self-learning framework, termed 2PFL, which addresses the challenges of extreme data scarcity and skewness when training classifiers over distributed labeled and unlabeled data. 2PFL demonstrates the capability to achieve high-performance models, even when trained with only 10% to 20% labeled data compared to the available unlabeled data.

The conclusion chapter underscores the importance of adaptable learning mechanisms that can respond to the continuous changes in clients' data volume, requirements, formats, and protection regulations. By incorporating the EC layer, we can alleviate concerns related to data

privacy, reduce the volume of data needing offloading, expedite task execution, and facilitate the training of complex models.

For scenarios demanding stricter privacy-preserving measures, FL offers a viable solution, enabling multiple clients to collaboratively train models while adhering to user privacy protection, data security, and government regulations. However, due to the indirect access to data inherent in FL, several challenges must be addressed to ensure the development of high-performance models. These challenges include imbalanced data distribution across clients, partially labeled data, and fully unlabeled data, all of which are explored and demonstrated through experimental evaluations.

Contents

1 Introduction **1**

1.1 Overview & Contributions 1

1.1.1 Research Questions & Solution Overview 3

1.2 Background 6

2 Edge Computing, Data & Tasks Offloading and Caching **9**

2.1 Introduction 9

2.2 Related Work 10

2.2.1 Data Caching 12

2.2.2 Task Caching 12

2.2.3 Fuzzy Logic Inference System 13

2.3 Preliminaries 14

2.3.1 Target Data Types and Application Scope 14

2.3.2 Service Architecture 15

2.3.3 Problem Statement 16

2.4 Methodology 19

2.4.1 Task Management Factors 19

2.5 Task Management Reasoning 23

2.5.1 Fuzzy Logic Inference Modeling 23

2.5.2 Two-stage Fuzzy Logic-based Reasoning 24

2.6 Performance Evaluation 28

2.6.1 Experimental Setup for Tasks' Popularities and Data Overlapping . . . 28

2.6.2 Experimental Setup 31

2.6.3 Comparative Assessment 32

2.7 Conclusions 34

2.8 Limitations & Directions of Enhancement 35

3 Node and Relevant Data Selection in Distributed Predictive Analytics: A Query-centric Approach **36**

3.1 Introduction 36

3.1.1 Regulatory Constraints and Privacy in Distributed Data Systems 36
3.1.2 Background 37
3.1.3 Motivation & Challenges 38
3.1.4 Contributions 40
3.2 Related Work 41
3.2.1 Node Selection in DPA 41
3.2.2 Data Relevance in DPA 43
3.3 Problem Fundamentals 44
3.3.1 Preliminaries & Definitions 44
3.3.2 Problem Formulation 46
3.4 Node & Relevant Data Selection 49
3.4.1 Data Relevance Factors: Overview 49
3.4.2 Data Relevance based on Factor F1 51
3.4.3 Data Relevance based on Factor F2 53
3.4.4 Data Relevance based on Factor F3 54
3.4.5 Ranking of Suitable Nodes 55
3.5 Query-centric DML Mechanisms 56
3.5.1 Best Node Model Learning 56
3.5.2 Aggregate/Weighted Aggregate Model Learning 58
3.5.3 Ranking-based Federated Learning 59
3.5.4 Ring-based Incremental Learning 60
3.6 Performance Evaluation 62
3.6.1 Experimental Setup 62
3.6.2 Baselines & Mechanisms under Comparison 66
3.6.3 Performance Metrics & Evaluation 68
3.6.4 Limitations & Directions of Enhancement 81
3.7 Conclusions 82

4 Cluster-based & Label-aware Federated Meta-Learning for On-Demand Classification Tasks 84
4.1 Introduction 84
4.2 Related Work 86
4.3 Target Data Types and Application Scope 87
4.4 Preliminaries 88
4.5 The CL-FML Framework 89
4.5.1 Overview 89
4.5.2 Label-aware Client Clustering 90
4.5.3 Cluster-based Multiple Meta-model Learning 93
4.5.4 Task-tailored Distributed Meta-model Learning 93

4.6 Experimental Evaluation 95
4.6.1 Experiment Setup 95
4.6.2 Main Results 97
4.7 Conclusions 102
4.8 Limitations & Directions of Enhancement 103

5 The Price of Labelling: A Two-Phase Federated Self-Learning Approach 104
5.1 Introduction 104
5.2 Related Work 106
5.3 Overview & Fundamentals 107
5.4 The 2-Phase Federated Self-Learning Framework 109
5.4.1 Local Data Augmentation 109
5.4.2 2PFL Training Phases 109
5.5 Experimental Evaluation 112
5.5.1 Experimental Set-up 112
5.5.2 Experimental Results 113
5.6 Conclusions 117
5.7 Limitations & Directions of Enhancement 117

6 Discussion and Conclusion 118
6.1 Key Contributions 118
6.1.1 Data and Task Offloading in Edge Computing 118
6.1.2 Node and Data Selection in Distributed Learning 118
6.1.3 Cluster-based & Label-aware Federated Meta-Learning 119
6.1.4 Self-Learning in Federated Learning with Minimal Labeled Data 119
6.2 Addressing Future Challenges: Increasing Complexity in Data 119
6.2.1 Data Privacy and Access Restrictions 120
6.2.2 Increasing Data Heterogeneity 120
6.3 Future Directions: Adapting to Data Scarcity and Complexity 120
6.4 Final Remarks 121

List of Tables

2.1 FLI rules inputs and the expected outputs. 26
2.2 T_k Information updating according to suitable node. 28
2.3 table: Tasks popularities. 29
2.4 Queries generation and percentages of data overlapping 29
2.5 S_2 decision making based on three factors. 31
2.6 Application types used in the simulation. 32
2.7 Simulation parameters. 32
2.8 The probability of offloading u_k for each task T_k according to our mechanism compared to the other two mechanism. 33
2.9 Table of Symbols for Chapter 2. 35

3.1 Summary of Complexities 62
3.2 Experimental parameters 65
3.3 Predictive Model & DML Mechanisms Hyper-parameters 66
3.4 Performance Metrics 66
3.5 Percentage of data accessed per query across node selection mechanisms in static and dynamic environments over DS1 and DS2. 73
3.6 MSE (loss) of predictive models based on all the node selection mechanisms over **DS1** across all the query-centric DML mechanisms (static data environment). 74
3.7 MSE (loss) of predictive models based on all the node selection mechanisms over **DS1** across all the query-centric DML mechanisms (dynamic data environment). 75
3.8 MSE (loss) of predictive models based on all the node selection mechanisms over **DS2** across all the query-centric DML mechanisms (static data environment). 76
3.9 MSE (loss) of predictive models based on all the node selection mechanisms over **DS1** across all the query-centric DML mechanisms (dynamic data environment). 77
3.10 Table of Symbols for Chapter 3. 83

4.1 Impact of Labelled-data Augmentation on Model Performance 98
4.2 Comparison of Methods 100
4.3 Impact of Overlapping/Similarity between Tasks & Clusters on Fine-tuned Models Performance. 101

4.4 Impact the Label Imbalance on Meta-model Accuracy 102

5.1 Impact of high-confidence pseudo-labels on test accuracy across phases. 114
5.2 Model test accuracy, LDR, and no. of training rounds across all methods. . . . 115

6.1 Acronyms & Abbreviations. 122

List of Figures

1.1 This diagram illustrates the three main methodologies for building models on distributed data: a) Centralized training: where all clients send their data to a central server to build the model. b) Edge computing training: where each subset of clients sends their data to an EC node instead of a remote server to build models.c) FL: where clients exchange models rather than data to collaboratively build global models. 7

2.1 A three-layer architecture of EC system. Arrows indicate the data and tasks transmission between the end-users, the EC, and the cloud layers. 16
2.2 Example of the availability of Cached data. 18
2.3 Tasks offloading decisions. 18
2.4 Tasks demand clusters using subtractive clustering (λ = 5, $\mathbf{W}$ = 50). Cluster-heads are marked with $\mathbf{X}$, and circles represent task requests. 20
2.5 Implementation of FLI on our three factors. 24
2.6 The probability of offloading. 27
2.7 The effect of (p_k, o_k, u_K) on the probability of offloading (r_k). 30
2.8 Data uploading speed and tasks execution Time. 34

3.1 Node & relevant data selection based on a query-centric paradigm. Nodes, e.g., edge servers, roadside units, receive DPA queries from DPA Apps to train ML models for predictive analytics. Generated data by Internet of Things (IoT) devices, e.g., sensors, are *selectively* accessed from only selected nodes (encircled in a dashed line) avoiding accessing irrelevant data for each DPA query. 39
3.2 Local models' prediction errors (loss) over ten nodes compared with the centralized model, the average model across all nodes, and the average model across only the most suitable nodes. 49
3.3 An illustration example of the impact of supportive clusters per query on predicting the relevant data on a node (data are projected onto a 2-dimensional space for illustration purposes only). 51
3.4 Comparison of BN model (top-1 node) vs. incremental model engaging the top-3 nodes. 57

3.5 The spectrum of the query-centric DML mechanisms from (a) Best Bode (BN), to (b) Aggregation (AM/WAM), (c) Ranking-based Federated Learning (RFL), and (d) Ring-based Incremental Learning (RIL). 58
3.6 Node selection accuracy α for DS2 and DS1 datasets. 69
3.7 Node selection frequency for our mechanism and the optimal one in static data environments. 71
3.8 Node selection frequency for our mechanism and the optimal one in dynamic data environments. 72
3.9 Percentage of relevant data accessed per proportion. 73
3.10 Distribution of MSE against proportion (%) of DPA queries (from the query workloads) using RFL in static and dynamic data environments (DS1). 80
3.11 Distribution of MSE against proportion (%) of DPA queries (from the query workloads) using RFL in static and dynamic data environments (DS2). 81

4.1 CL-FML instance (clockwise). **(i)** All clients' labels $\mathcal{L} = \{a, b, c, d, f\}$ and task's labels $\mathcal{T} = \{a, b, f\}$. **(ii)** Label-aware clustering into C_1, C_2 groups and decentralized cluster-based meta-learning. **(iii)** Decentralized training of g_ℓ for data augmentation. **(iv)** Decentralized fine-tuning for task-tailored meta-model among suitable clients (shaded). 90
4.2 Multiple meta-models' top-1 accuracy (%) of CL-FML against global meta-model (G-FML) vs. convergence (samples of two groups). 99
4.3 CL-FML against G-FML vs. convergence (samples of two groups). 99

5.1 The Phases 1, 2 and 2+ of the 2PF framework progressively engaging labelled, partially-labelled and unlabelled clients in distributed self-learning. 108
5.2 Accuracy vs. training rounds for all methods and datasets (the two vertical dotted lines correspond to T_1 and $T_1 + T_2$ round milestones of 2PFL's phases). 116
5.3 pseudo-labelling ratio of unlabelled samples across datasets and phases. 116

And pray, “My Lord!
Increase me in knowledge.”

Acknowledgements

AlhamduliLlahi Robil 'Alamin!

First and foremost, all praise is due to **Allah**, who guided me to embark on this journey and granted me strength and support throughout.

I am deeply grateful to my supervisors, **Dr. Christos Anagnostopoulos** and **Dr. Fani Deligianni**, for their unwavering support, invaluable guidance, and immense patience over the years. Their mentorship has been instrumental in shaping my academic journey and helping me achieve my goals.

I would like to express my deepest gratitude to my examination panel for their time, effort, and invaluable feedback. I am particularly thankful to my external examiner, **Dr. Nikos Tziritas** from the University of Thessaly, and my internal examiner, **Dr. Emma Li**, for their thorough review and insightful comments. Their constructive feedback and thoughtful discussion were instrumental in enhancing the clarity, structure, and overall quality of this thesis.

I am also sincerely grateful to the viva convener, **Dr. Tanaya Guha**, for her efforts in coordinating and facilitating a smooth, well-organized viva.

I would like to express my sincere appreciation to the **Saudi Ministry of Education** for providing me with a scholarship and generously funding my travels during my studies. Additionally, I am thankful for the continued support from the **Saudi Arabian Cultural Bureau in the UK**, whose assistance has been crucial in facilitating my academic and professional growth. I would like to extend my sincere gratitude to **TRACE HORIZON**, where I joined as a research assistant in June 2024. The opportunity to collaborate with them has been invaluable to my research journey.

A heartfelt thank you to my dear friend, **Hessa Al-Thabeti**, for standing by me and supporting me during the early stages of my scholarship and travels. her friendship made the transition smoother, and I will always be grateful for her kindness. I extend my gratitude to my colleagues and friends, particularly **Ghadeer Alsharif**, with whom I shared an office. I am also grateful to **Nesreen Alareef** and **Arwa Alsubhi**, whose camaraderie and insightful discussions enriched my PhD experience. A special thanks goes to **Ibrahim Alghamdi** from **Al-Baha University**, whose guidance in the early stages of my PhD and publication journey was invaluable. I would also like to acknowledge my friends, **Munira Al-Khashan**, **Alaa Al-Ghamdi**, and **Ahd Al Jaidi**, for their friendship and encouragement, which have been a source of comfort and strength during my studies.

Finally, and most importantly, I owe my deepest gratitude to my late father, my lifelong role model, who did everything in his power to give me the best opportunities. His memory continues

to inspire and motivate me every day. To my beloved mother, I offer my sincerest thanks for her boundless love, encouragement, and unwavering support. You have both been my greatest pillars of strength.

To my siblings—**Asmaa**, **Nadia**, **Sumaya**, **Abdullah**, **Mohammed**, **Walid**, and **Abeer**—thank you for your constant encouragement and support. Your belief in me has been an incredible source of motivation, and I am forever grateful for our shared journey.

Chapter 1

Introduction

1.1 Overview & Contributions

Motivation and Thesis Statement

The rapid advancement of end-user devices, such as smartphones, laptops, and smartwatches, has equipped them with sensors like cameras, microphones, and GPS. As these devices are ubiquitously carried by users, they generate an immense volume of private and personal data, which is stored locally on the devices [1]. This data represents a valuable resource for deriving actionable insights, supporting dynamic decision-making across diverse domains, and enabling the development of intelligent applications leveraging deep learning (DL). Such applications span a range of fields, including autonomous driving, video analytics, surveillance, augmented and virtual reality, facial and speech recognition, anomaly detection, and natural language processing [2, 3]. The effectiveness of deep and complex models depends on access to large, varied datasets. Initially, centralized training on cloud servers was the predominant method for training large-scale models, which require substantial computational resources for efficient execution [2]. However, the increasing demands of data-driven applications—such as real-time processing, rapid decision-making, frequent user requests, and extensive data requirements relative to available bandwidth—have made centralized approaches increasingly impractical. Concurrently, growing user awareness regarding data privacy and the risks associated with centralized data storage has further driven this shift. As a result, there is a notable transition towards decentralized learning methodologies, leveraging **edge computing (EC)** or directly processing data on end-user devices through **federated learning (FL)**.

Incorporating an EC layer between the cloud and end-user devices to train models and execute analytical tasks can significantly mitigate concerns associated with centralized training while offering multiple benefits. These benefits include reduced latency and communication delays, lower energy consumption, enhanced network stability, improved security, increased reliability, and support for real-time applications [4, 5]. Despite these advantages, EC resources remain

constrained, particularly given the rapidly increasing volume of requests from end-users. This constraint presents challenges in the efficient selection of EC servers and the decision-making process for each analytical task. Effective execution of tasks on EC servers involves more than merely offloading computations and returning results; it requires minimizing redundant task offloading and reducing the amount of data that needs to be offloaded. Thus, it is crucial to not only facilitate task offloading but also to cache frequently accessed content to reduce response times and decrease network load by minimizing repeated transmissions [6].

Consequently, it is imperative to consider both offloading and caching strategies concurrently, leading to the formulation of **RQ1** (1.1.1). The proposed solution for this research question is outlined in **S1** (1.1.1) and elaborated in **Chapter** 2.

While EC can alleviate some challenges associated with executing analytic tasks in the cloud, it cannot entirely resolve these issues. For instance, EC facilitates data processing closer to end-users via local, trusted edge servers, thereby avoiding the traversal of data over the public Internet [2]. This approach can mitigate privacy and security risks by reducing exposure to potential attacks. However, it does not eliminate the reliance on third parties to centralize scattered data. Although data offloading can be reduced to as low as 10% or less, complete elimination (0% offloading) is unattainable, leaving the data disclosure issue unresolved. This limitation renders EC an impractical solution in scenarios where data cannot be shared due to privacy concerns or data protection regulations, such as with medical records, financial data, and personal information [7]. To address these challenges, significant efforts have been directed toward enabling DL model training directly on end-user devices through the adoption of FL. FL allows multiple end-user devices to collaboratively train a DL model without exchanging their local private data. By adopting FL, data offloading can be reduced to 0%, relying primarily on model exchange rather than data exchange. FL reduces the need for data centralization, thus, alleviating concerns about data privacy, decreasing the load on central servers, and minimizing communication overhead. However, in many real-life scenarios, data heterogeneity is prevalent, with each user potentially exhibiting unique data distributions, volumes, access patterns, and feature spaces. This heterogeneity poses significant challenges to developing accurate models in a distributed environment. Many state-of-the-art methods rely on random node selection as a straightforward approach. However, this method increases the risk of selecting end-user devices with low-quality or irrelevant data for a specific task. Consequently, only after models are trained on randomly selected users can the most suitable nodes be identified based on predictive performance. This process results in increased time and resource consumption, as well as a higher network load. Therefore, a comprehensive understanding of the data characteristics and access patterns of nodes is essential. Such knowledge facilitates the pre-selection of an optimal subset of nodes for each task prior to model training. Thus, it is crucial to consider the most suitable nodes by evaluating the availability and relevance of data for each task, leading to the formulation of **RQ2** (1.1.1). The proposed solution for this research question is outlined in **S2**

(1.1.1) and detailed in **Chapter** 3.

Another primary challenge in FL related to the classification tasks is the heterogeneity of data and class labels. This issue arises due to various types of distribution shifts among clients, including feature, label, and concept distribution shifts. A key aspect of this challenge is the uneven distribution of data across classes: majority classes dominate the dataset, while minority classes are represented by only a small fraction of the data [8]. The limited availability of labeled data, exacerbated by disparities in label distributions across clients, hinders the convergence of classifiers and degrades their performance. This problem is particularly detrimental when classification involves minority classes [9] or any arbitrary set of labels. Distributed analytics involving classification tasks necessitate robust models training, particularly for real-time, arbitrary classification tasks across distributed clients. Federated (Meta)-Learning (FML) has emerged as a solution for global distributed (meta)-model training, offering generalization across diverse data and classification tasks. However, current FML approaches often assume fixed labels, balanced class proportions, uniform data distributions, and consistent task distributions. As a result, global meta-models are limited to tasks that do not require addressing arbitrary out-of-distribution label issues. In real-world scenarios, class imbalance and label shifting are pervasive challenges in clients' data, and on-demand tasks often involve previously unseen labels. Consequently, a 'one (meta)-model-fits-all' approach is insufficient. To address these challenges, we formulate **RQ3**1.1.1, and the proposed solution for this research question is outlined in **S3**1.1.1 and elaborated in **Chapter** 4.

In more changeable realistic FL scenarios, where client data may be unlabelled due to factors such as labeling costs, time constraints, or a lack of expertise and resources, addressing these challenges becomes crucial. To tackle this issue, self-learning FL methods have been proposed, which utilize both labelled and unlabelled data. However, self-learning approaches typically rely on the availability of ground truth labels for a subset of the data, either on the server or distributed among clients. Moreover, these methods generally assume that data are independently identically distributed (IID). In contrast, real-world scenarios often involve non-IID data, presenting significant challenges in generating high-quality pseudo-labels and managing data heterogeneity effectively. To address these challenges, we formulate **RQ4**1.1.1, and the proposed solution for this research question is outlined in **S4**1.1.1 and elaborated in **Chapter** 5.

1.1.1 Research Questions & Solution Overview

The thesis focus is on developing methods that increase the likelihood of training models with the most relevant nodes and suitable data. This is accomplished through a range of techniques, including assessing data availability and relevance, clustering similar clients, utilizing multi-meta models, and leveraging self-supervised learning. This section highlights the key research questions addressed in the thesis and outlines the proposed solutions for each.

RQ1: How can we optimize EC server selection to minimize data offloading and reduce redundant task execution?

S1: This solution introduces a robust approach to optimizing the decision-making process for offloading and caching analytic tasks by utilizing a proposed EC server selection mechanism. The approach is designed to *efficiently allocate data-driven analytic tasks between EC and cloud environments, minimizing execution delays and maximizing EC server resource utilization*, based on task management factors and strategic reasoning (elaborated in **Chapter** 2).

RQ2: How can we efficiently select only the relevant data from each chosen node per analytics query when access to data is limited?

S2: Due to the growing concerns surrounding data privacy, access to data is becoming increasingly restricted. To address this challenge, this chapter proposes a query-centric approach to Distributed Predictive Analytics (DPA). For each query, the method privately identifies the most suitable subset of nodes to participate in the analysis. This selection process leverages each node's statistical predictions based on its local data to estimate and rank their relevance. By focusing only on the most effective nodes, the approach improves model accuracy per query while minimizing unnecessary access to irrelevant training data. Further details are provided in Chapter 3.

RQ3: How can we effectively mitigate label shifting and address missing label challenges in FL, while ensuring that the meta-model generalizes to different data distributions?

S3: The solution introduces a cluster-based approach to federated meta-learning (CL-FML). This method groups clients by label distributions to better handle label shifts and imbalances. By using multiple meta-models, each tailored to specific clusters, we ensure that the models can generalize to different data distributions. Additionally, lightweight data augmentation techniques are used to improve performance on class-imbalanced tasks (elaborated in **Chapter** 4).

RQ4: What is the 'price of learning' a global model using scarce, skewed, and distributed labelled data, while capitalizing on partially labelled and fully unlabelled data across clients?

S4: When data is limited or skewed, it becomes difficult to train accurate models. To solve this, we propose a two-Phase Federated self-Learning framework, coined **2PFL**, that addresses both extreme data scarcity and skewness in training classifiers over distributed labelled and unlabelled data. 2PFL demonstrates the ability to achieve high-performance models when trained with only 10% to 20% labelled data compared to the unlabelled data (elaborated in **Chapter** 5).

Academic Contributions to the Research Community

The work presented in this thesis is primarily derived from published research conducted by the author in collaboration with other researchers. The chapters and the corresponding papers on which these chapters are based are listed below.

1. Chapter 2: Edge Computing, Data & Tasks Offloading and Caching

    - **Tahani Aladwani**, Christos Anagnostopoulos,Konstantinos Kolomvatsos, Ibrahim Alghamdi,"Data-driven analytics task management reasoning mechanism in edge computing." Smart Cities 5.2 (2022): 562-582.
    - **Tahani Aladwani**, Christos Anagnostopoulos,Konstantinos Kolomvatsos, Ibrahim Alghamdi, "Data-Driven Analytics Task Management at the Edge: A Fuzzy Reasoning Approach." 2022 9th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, 2022.

2. Chapter 3: Node and Relevant Data Selection in Distributed Predictive Analytics: A Query-centric Approach

    - **Tahani Aladwani**, Christos Anagnostopoulos,Konstantinos Kolomvatsos, "Node and Relevant Data Selection in Distributed Predictive Analytics: A Query-centric Approach." Journal of Network and Computer Applications, Elsevier, 2024: 104029.
    - **Tahani Aladwani**, Christos Anagnostopoulos,Konstantinos Kolomvatsos, Ibrahim Alghamdi, "Query-driven Edge Node Selection in Distributed Learning Environments." IEEE 39th International Conference on Data Engineering Workshops (ICDEW), IEEE, 2023.

3. Chapter 4: CL-FML: Cluster-based & Label-aware Federated Meta-Learning for On-Demand Classification Tasks

    - **Tahani Aladwani**, Christos Anagnostopoulos,Shameem P. Parambath,Fani Deligianni, "CL-FML: Cluster-based & Label-aware Federated Meta-Learning for On-Demand Classification Tasks." 11th IEEE International Conference on Data Science and Advanced Analytics (DSAA 2024), IEEE, 2024.

4. Chapter 5: The Price of Labelling: A Two-Phase Federated Self-Learning Approach

    - **Tahani Aladwani**, Christos Anagnostopoulos,Shameem P. Parambath,Fani Deligianni, "The Price of Labelling: A Two-Phase Federated Self-Learning Approach." European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML-PKDD 2024.

Thesis Structure

The thesis is structured into six main chapters. The Introduction provides an overview of the topics discussed in subsequent chapters and summarizes key contributions to the research community. The second part of the Introduction introduces foundational concepts that are referenced in later chapters. Chapters 2, 3, 4, and 5 form the core of this thesis. Specifically: Chapter 2 focuses on improving task execution on EC servers. Chapter 3 addresses the selection of optimal nodes and relevant data through statistical predictions. Chapter 4 concentrates on mitigating label shifting and addressing missing label challenges in FL. Chapter 5 explores approaches to managing extreme data scarcity and skewness in training classifiers over distributed labeled and unlabeled data. Finally, Chapter 6 offers concluding remarks, discusses limitations, and outlines potential directions for future work.

1.2 Background

This section explains the key concepts that may be unfamiliar to the reader, offering a concise overview of the methodologies referenced throughout this thesis. By acquainting the reader with these fundamental principles, the ensuing discussions and applications within the thesis will be rendered more comprehensible.

Definition 1 (Computation Offloading)**.** *Computation offloading is a concept that applies to both cloud computing and EC, involving the transfer of some or all computational tasks from end-user devices to more powerful infrastructure, such as cloud servers or nearby EC servers [4], as in Figure (a-1.1) and Figure (b-1.1).*

Definition 2 (Federated Learning)**.** *Federated Learning (FL) is a distributed learning paradigm that allows multiple clients (e.g., smartphones, laptops, hospitals) to collaboratively train deep learning (DL) models without sharing their private raw data [10], as in Figure (c-1.1).*

Consider a central server $\mathbf{S}$ and $\mathcal{N}$ client devices. Each client $i \in \mathcal{N}$ possesses its own local dataset D_i for $i = 1, 2, \ldots, \mathcal{N}$. Each client $i \in \mathcal{N}$ holds m_i samples, where each sample is characterized by features $x \in \mathcal{X}$ and associated labels $y \in \mathcal{Y}$. These samples are drawn from either IID or non-IID distributions, depending on the degree of data heterogeneity among the clients. Meanwhile, they utilize similar data features, which can vary based on the type of data:

- **Tabular data**: For instance, weather-related features such as humidity, temperature, and pressure, or vehicle parking sensor data like occupancy duration, sensor type (magnetic/ inductive), and air quality, typically used in regression models.
- **Image data**: For classification models, clients might use popular datasets like MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100.

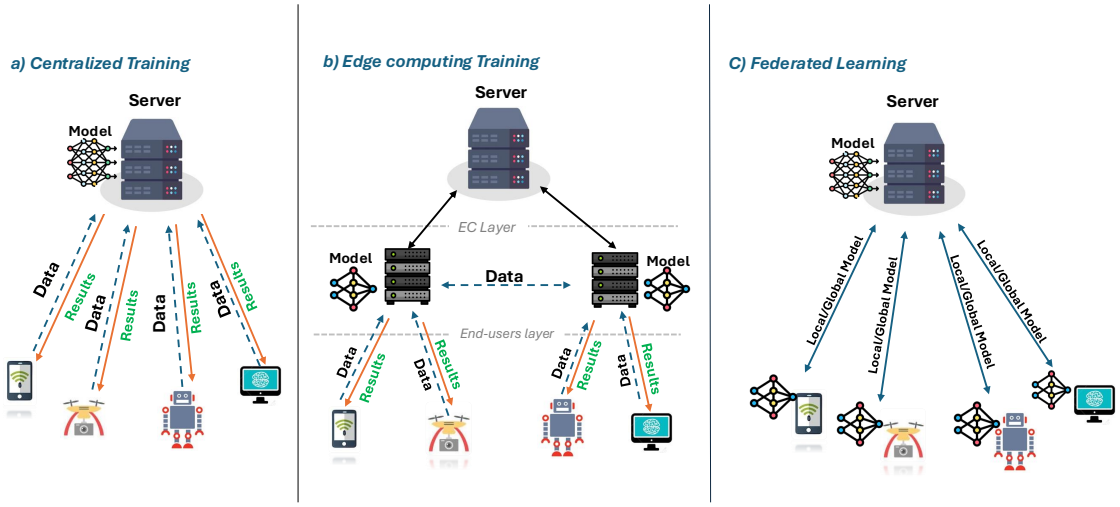


Figure 1.1: This diagram illustrates the three main methodologies for building models on distributed data: a) Centralized training: where all clients send their data to a central server to build the model. b) Edge computing training: where each subset of clients sends their data to an EC node instead of a remote server to build models.c) FL: where clients exchange models rather than data to collaboratively build global models.

The primary objective is to train an efficient model θ using the combined dataset $D \triangleq \bigcup_{i\in[N]} D_i$, with the assistance of a server **S**. This process can be facilitated either through direct access to the data (computation offloading) or via indirect access to the data (FL).

In the context of computation offloading 1, each client n_i transmits its local data D_i to a central server. The server aggregates all received datasets into a single dataset and trains a global model θ with the objective of minimizing a global loss function $\ell(\theta)$. Formally, the global optimization problem can be expressed as:

$$\min_{\theta} \ell(\theta) = \frac{1}{\sum_{i=1}^{\mathcal{N}} |D_i|} \sum_{i=1}^{\mathcal{N}} \sum_{x\in D_i} \ell(\theta; x), \tag{1.1}$$

where $\ell(\theta; x)$ represents the loss associated with a data point x in the unified dataset, and $|D_i|$ denotes the size of the dataset D_i.

Once the server has trained the model θ that minimizes the global loss $\ell(\theta)$, the resulting model θ is then returned to each client for further use.

In the context of FL 2, the objective is to optimize a global model θ using the server's data

$D_{\mathbf{S}}$ and the data from the clients $[D_1, \ldots, D_{\mathcal{N}}]$ such that:

$$\arg\min_{\theta \in \mathbb{R}^d} \left[\ell(\theta) \triangleq \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} L_i(\theta) \right] \tag{1.2}$$

where $L_i(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}_i} [\ell_i(\theta; (x, y))]$ denotes the empirical loss for the kth client, and θ represents the parameters of the neural network. During each round $t \in [T]$, a subset of clients $P_t \subset [m]$ is selected at random, and the server $\mathbf{S}$ transmits its current model θ^{t-1} to these selected clients. Each client then performs optimization to minimize a local empirical risk objective. The local update for the ith client at round t can be expressed as:

$$\theta_i^t = \theta_k^{t-1} - \eta \nabla L_i(\theta_i^{t-1}), \tag{1.3}$$

Where θ_i^t is the updated model parameter for client i at round t, θ_i^{t-1} is the model parameter from the previous round $t-1$, η is the learning rate and $\nabla L_i(\theta_i^{t-1})$ is the gradient of the local loss function L_i with respect to the model parameters at round $t-1$.

Chapter 2

Edge Computing, Data & Tasks Offloading and Caching

2.1 Introduction

In recent years, advancements in technology have led to the growing popularity of computationally intensive applications in fields such as surveillance, augmented and virtual reality, facial and speech recognition, image processing, and natural language understanding[3]. These applications often rely on DL models, which require substantial computational resources and large datasets to perform effectively. However, in practice, data is typically dispersed across different clients (e.g., mobile devices, sensors, companies, etc.). Meanwhile, despite advances in smart device hardware, many of these devices still cannot handle such intelligent, delay-sensitive, computationally-intensive applications [11]. Computational offloading on Cloud Computing (CC) has been proposed as a beneficial way to address the clients' resource limitations and execute the intensive tasks remotely [12]. However, CC is inherently centralized, which means it is often located far from the clients. Consequently, relying solely on CC can result in significant communication delays and high latency for delay-sensitive applications, low Quality of Service (QoS) level, as well as increased privacy concerns [3, 5]. To mitigate these drawbacks, considerable efforts have been made to push the processing of DL models to edge servers. These servers, positioned closer to end-user devices at the edge of the network, to enhance computational capabilities by introducing an additional layer of resources known as EC between CC and clients [3].

This approach addresses the critical requirements of low latency, reduced communication delays, minimized energy consumption, enhanced network stability, improved security, increased reliability, and real-time application support [4, 5]. Even though EC servers offer numerous advantages, their resources are still limited, especially with the rapidly increasing number of requests from end-users. This limitation presents challenges in selecting efficiently EC servers and making decisions for each analytic task. Executing analytic tasks on EC servers involves

more than just offloading computations and returning results. It also requires reducing the need to offload the same tasks multiple times and minimizing the data required for offloading. Thus, it is crucial not only to support task offloading but also to cache popular content for future use [13]. Caching popular content or computation results of frequently requested analytic tasks improves information retrieval efficiency and reduces response time. It also decreases network load by minimizing repeated transmissions [6]. Therefore, it is crucial to consider both offloading and caching decisions in parallel. Key decisions include whether a task should be offloaded, which EC server should handle the task, and whether the task should be cached. Offloading decisions depend on various factors such as EC server resource availability, the number of tasks, data requirements for each task, and task priorities (e.g., immediate execution or delay to prioritize the higher-priority tasks) [12, 14, 15]. Similarly, caching decisions are based on various factors, such as task popularity, data size, and required computing resources [16]. Therefore, designing an efficient EC selection mechanism that balances task offloading and caching remains a challenging issue. Therefore, this chapter presents a robust approach for determining task caching, offloading decisions, and resource allocation mechanisms aimed at maximizing service provider revenue while minimizing overall task latency. Our mechanism specifically addresses the following critical issues: (i) meeting the stringent requirements of latency-sensitive IoT applications, (ii) efficiently utilizing resources within Edge-Cloud environments, and (iii) scheduling offloading tasks to reduce overall service time and enhance resource utilization [15]. The chapter's optimization objectives can be summarized as follows:

1. **Maximize Offloading Probability to Optimal Servers:** Ensure tasks are directed to the most suitable EC server, considering factors like task popularity and data availability.

2. **Minimize Data Offloading Rate:** Reduce data transfer during offloading by implementing data caching, thereby enhancing efficiency and lowering bandwidth usage.

3. **Increase EC Server Resource Utilization:** Maximize use of resources at EC servers to improve overall system efficiency and performance.

4. **Maximize the chance of selecting the right EC Server for each task:** Fuzzy Logic Inference has been adapted to enhance decision-making in server selection.

2.2 Related Work

As discussed in Section 2.1, EC servers are designed to optimize service latency, energy consumption, and communication overhead, while also maximizing total revenue and resource utilization. However, due to the inherent limitations in resources and storage capacity of EC servers, it is impractical to offload and cache every task. This challenge necessitates selective mechanisms whereby certain tasks are executed and cached on EC servers, while others are

offloaded to the cloud. These decisions introduce several trade-offs between the requirements of clients—such as reduced latency and lower energy consumption—and those of EC servers, including optimized resource usage. As a result, extensive research has been directed towards developing efficient resource management strategies that can balance these competing needs [12, 14]. Current studies in this field typically concentrate on three core aspects: task offloading, data caching, and task caching.

Tasks Offloading

Task and data offloading refers to the process of transferring a computational task or workload from an end-user device to a remote entity, such as an EC server or the cloud, to enhance the performance and efficiency of the end-user devices while reducing computing delays [16, 17]. Regarding task offloading options, a client can offload either part or all of the computing tasks, as well as the associated data, to EC servers. Offloading tasks to an EC server, which possesses greater computing resources and power, can significantly accelerate task execution, conserve the device's energy, and decrease response times [18]. Consequently, offloading decisions can be summarized as follows: **First**, can a task be executed entirely on the local device, or is offloading necessary. **Second**, if offloading is required, should the task be fully offloaded—where the entire task is migrated to the EC server for execution—or partially offloaded, where part of the task is processed locally while the remainder is offloaded. **Third**, when should a task be offloaded, how many tasks should be offloaded, and to which EC server should they be directed?

These decisions are influenced by various factors such as required computational power, data volume, task dependency, and task priority [12]. Given the resource constraints of EC servers, selecting appropriate server selection mechanisms and offloading strategies is crucial, particularly as EC servers often manage a high volume of requests from end-users [19].

Extensive research has been conducted to enhance offloading strategies for diverse scenarios and optimization goals [5].

These studies can be categorized into three groups based on their objectives related to data and task offloading: **The first category** emphasizes application characteristics, such as computational and communication demands, as well as latency sensitivity, as discussed in [14, 20, 21, 22, 23]. **The second category** focuses on the characteristics of EC resources, including EC service latency, energy consumption, and cost, as highlighted in [24, 15, 25]. **The third category** examines both application and EC resource characteristics, aiming to minimize trade-offs between application requirements (e.g., latency constraints and computational demands) and system requirements (e.g., maximizing resource utilization) [26, 24, 27, 28, 29, 30].

2.2.1 Data Caching

Another important feature provided by EC servers is caching capacity, which represents a promising approach to enhancing the execution of remote tasks and the QoS. Data caching in EC servers is a widely adopted technique for storing copies of data to expedite retrieval for future request [31]. This technique serves various purposes, including improving efficiency in information retrieval, reducing data access latency, enhancing QoS, improving energy efficiency, and reducing network load [13]. There has been extensive research aimed at improving data caching in EC servers, with a focus on optimizing system performance, enhancing energy efficiency [22, 32, 33], and optimizing network resource utilization while reducing transmission costs and response times [31, 34, 35, 36]. For instance, researchers have applied data selection mechanisms to cache popular data and content at EC servers, although measuring data popularity for each data point can be computationally intensive [37, 33]. All these studies aim to design an optimal caching strategy that maximizes cache performance and delivers popular content efficiently to clients. However, many of these solutions overlook the heterogeneity in analytic task requirements. For instance, in an EC server when 50% of data may be popular across all tasks, the remaining 50% could be popular for specific analytic tasks only, the second 50% should be cached or not? There is also a need to address unpopular data that may be filtered out due to lack of popularity. Moreover, these solutions often neglect scenarios where required data is missing for example, when a client needs data that is not cached on the selected EC server. In such cases, it is crucial to determine whether the client should retrieve this data from another EC server or directly from end-users devices or the task should be migrated to another EC server (when the amount of missing data is significant and it can be available on another EC server, or to an EC server with low load at this moment). This practical scenario is overlooked in current research efforts.

2.2.2 Task Caching

In this subsection, we introduce task caching as a service provided by EC servers. Task caching involves storing the task application, related data, and associated running environment in EC servers or the cloud [16, 38]. When an end-user device requests an analytic task that needs to be offloaded to the EC server, the server checks if the task is already cached. If the task has been cached, the EC server informs the end-user that it exists and directly computes the result. This approach eliminates the need for the end-user to offload the task again. Conversely, if the task is not cached, the end-user must offload it to the EC server for processing. Task caching offers numerous advantages for the network, end-user devices, and EC servers, including reduced energy consumption, costs, offloading delays, and task completion times [39]. However, as mentioned earlier, EC servers have finite computation resources and storage capacity. Therefore, making caching decisions becomes challenging due to task heterogeneity, data sizes, and varying

computational and storage requirements [16]. Previous research has proposed various caching strategies to determine which tasks should be cached. Some strategies prioritize caching based on task popularity [40]. For example, in [41, 42], EC servers assume knowledge of end-users' demand patterns by observing their contextual information. However, this approach necessitates maintaining a significant amount of context information within EC servers.

While other works focused on the computing capabilities. [38] introduced a task caching algorithm that considers both the constraints of EC servers' caching and computing capabilities, as well as user mobility, under the assumption of unknown user requests. Both [16, 43] proposed task caching algorithms for heterogeneous computing tasks, considering the required computing resources and data size for each task.

In previous studies, the concept of caching has been integrated with analytic tasks, wherein the application and its associated data are entirely offloaded as a single unit to the EC server. However, as discussed in Section 2.2.1, the amount of data required by an analytic task that is already cached on the EC server has often been overlooked. This oversight can lead to excessive data uploading if there is no careful consideration of the match between cached data and the data needed for a new analytic task. Therefore, it is crucial to identify the overlap between cached data and the incoming analytic task that needs to be offloaded and cached.

Moreover, most related works tend to focus on one of three aspects independently: task offloading, data caching, or task caching. Nevertheless, an efficient task offloading mechanism must consider the consistency between the cached data on the EC server and the requirements of the new tasks. Additionally, determining whether a task should be cached is contingent upon the likelihood of it being requested again by another client. Finally, when a client needs to offload an analytic task, it is imperative to decide whether to offload the task entirely from scratch along with all its requirements or to leverage related data that is already cached on the EC server.

In this chapter, unlike previous studies, we integrate task and data offloading and caching under the assumption that user requests are dynamic but can be tracked, and that EC servers have a certain amount of cached data. We propose a data-driven decision-making mechanism for task offloading and caching. The goal is to improve the task completion rate while minimizing communication costs and efficiently meeting the clients' demands, all within the constraints of EC servers' resources.

2.2.3 Fuzzy Logic Inference System

Fuzzy Logic Inference System (FLI) has been utilized in offloading and caching decision-making. As defined by Welstead in [44], it is a set of rules and regulations that defines boundaries and provides guidelines for successfully solving problems within these boundaries. This type of logic tries to mimic human behavior in decision-making by avoiding strict boundaries between categories, in contrast to crisp logic. FLI depends on studying the degree of membership and belonging, making it an excellent option for managing real-world uncertainty due to rapidly

changing scenarios [45]. Consequently, it has been employed for solving online and real-time problems. For example, it has been used in decision-making processes for heaters according to the weather, defining a person's relative age group, and determining security levels in online shopping or trading. In our context, FLI has been applied in numerous studies to make execution decisions in EC models.

In [46], FLI has been employed to reduce the number of failed tasks resulting from transmission collisions and to support real-time applications. The proposed FLI system is designed to determine the optimal execution environment for each task—whether it should be processed by its own resources, a local edge server, or the cloud. The decision-making model is divided into two stages: the first stage focuses on determining where to place the incoming task, while the second stage identifies the most suitable location for task processing. However, this study primarily emphasizes resource availability, without taking into account data assessment and task popularity. Similarly, in [14], FLI is utilized to make processing decisions for tasks, either in the EC server or in the cloud. Their FLI system is based on three key parameters: CPU utilization, WAN bandwidth, and delay sensitivity. This model demonstrates favorable results in terms of reducing the average number of unsuccessful tasks and improving resource utilization when compared to benchmark algorithms. However, despite considering task sensitivities, the model does not address data access.

In general, FLI has been discussed in many studies to improve EC QoS. However, to the best of our knowledge, the incorporation of task popularity and data overlap as inputs to FLI systems to optimize offloading and caching mechanisms in EC has not yet been explored within this domain.

2.3 Preliminaries

This section mainly focuses explain the target data types and application scope, service architecture, and the problems formulation.

2.3.1 Target Data Types and Application Scope

The Chapter primarily focuses on distributed data with minimal privacy constraints—data that can be offloaded and cached on external servers for task execution. This includes sensor data, edge-generated data, and other forms typically handled within edge computing environments. The proposed framework is particularly well-suited for data-driven analytics tasks such as weather prediction, pollution monitoring, and healthcare diagnostics, which often involve spatio-temporal features like temperature and humidity. Such data are typically distributed across multiple edge nodes or centers. In our experiments, we used real-world datasets collected by Unmanned Sur-

face Vehicles (USVs) GNFUV ([1] that gathered time-series environmental data (e.g., sea surface temperature and humidity) from various coastal regions. The framework is also applicable to other domains such as smart agriculture, where edge sensors monitor soil moisture and temperature; intelligent transportation systems, where roadside units collect traffic flow and vehicle telemetry; and smart buildings, where IoT devices track environmental conditions for energy optimization. These types of tasks can be significantly accelerated and improved by adopting our fuzzy inference-based offloading framework, as they involve continuous data collection, demand timely processing, and benefit greatly from efficient edge resource utilization.

2.3.2 Service Architecture

The EC general architecture consists of a three-layer service architecture for data-driven task offloading. These layers are the End-Users/Applications layer, the EC servers layer, and the Cloud Computing layer, as shown in Figure 2.1.

- **End-Users/Applications Layer**: This layer encompasses data generated by a myriad of devices, including mobile phones, computers, wearable smart devices, cameras in smart cities, and various sensors (e.g., speed sensors in smart cars, environmental sensors deployed in specific areas) [37]. These devices typically possess limited and varying computing and storage capabilities [47]. Consequently, the analytic tasks required by the end-users or applications are processed by EC servers or cloud infrastructure, with the results subsequently relayed back to the end-users or applications.

- **EC servers Layer**: The EC servers layer is an intermediate layer located between the cloud and End-users/Applications layers. This layer includes a set of collaborative EC servers that provide computing services for computationally intensive, real-time, and delay-sensitive tasks [48]. It contains computing, storage, and network resources such as a micro data center with a virtualized environment, gateways, control units, storage units, and computing units [25]. This enables it to perform some tasks that were originally offloaded to the cloud. Since this layer has been placed close to end-user devices, which means the transmission distance is significantly shortened, leading to a considerable reduction in transmission time [21].

- **Cloud Layer**: This layer has unlimited computing and storage capabilities. Beside its ability to macro-control the entire EC architecture. Thus, tasks can be offloaded to the cloud through, e.g., Base Stations (BSs) from the EC layers due to limited resources or failure in completing tasks. However, its centralized and remote infrastructure leads to substantial communication delays, which are cannot satisfy the requirements of the delay-sensitive analytic tasks [5].

[1] `https://archive.ics.uci.edu/dataset/452/gnfuv+unmanned+surface+vehicles+sensor+data`

However, the decision regarding which layer the analytic tasks should be executed in must be based on a node selection mechanism. This mechanism should consider various factors such as task size, time constraints, task popularity, data access rate, and resource availability to make the offloading decision. As we will discuss in the next section.

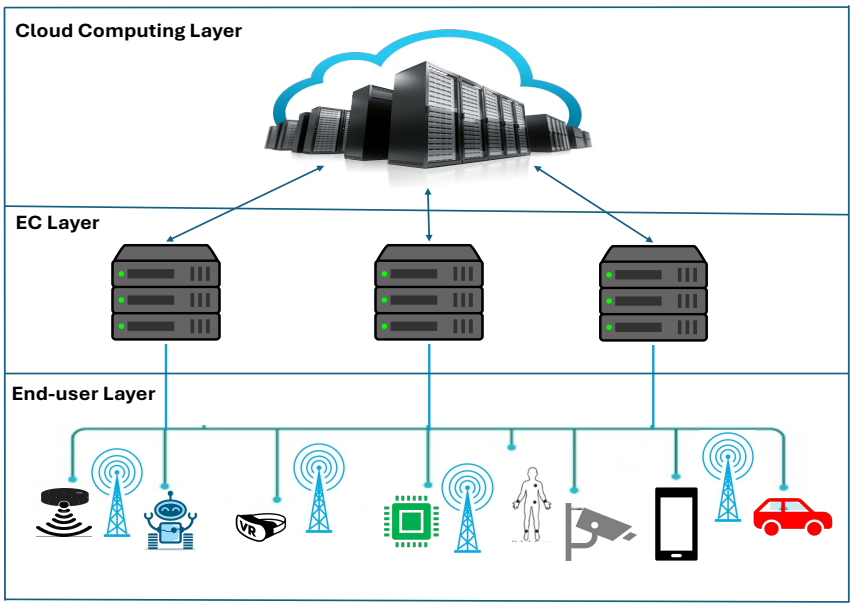


Figure 2.1: A three-layer architecture of EC system. Arrows indicate the data and tasks transmission between the end-users, the EC, and the cloud layers.

2.3.3 Problem Statement

We consider an EC system comprising a set of $\mathbf{N}$ EC servers, denoted by $\mathbf{N} = \{n_1, n_2, \ldots, n_N\}$. Each EC server n_i collects N_i real-valued contextual data points, represented as $\mathbf{x} = [x_1, x_2, \ldots, x_d]^\top \in \mathbb{R}^d$. Here, $\mathbf{x}$ is a d-dimensional vector where each dimension corresponds to a specific feature, such as temperature or humidity. The n_i server stores them locally in the dataset $\mathcal{D}_i = \{\mathbf{x}_k\}_{k=1}^{N_i}$. Each server n_i has a neighborhood $\mathbf{N}_i \subset \mathbf{N}$ of directly communicating servers $n_j \in \mathbf{N}_i$. Moreover, server n_i communicates with the end-users/applications and the cloud. These EC servers have specific computing capabilities, which in turn forms (and contributes to) the resources available in the EC servers layer [15]. Meanwhile, end-users, denoted as $\mathbf{U} = \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$, generate computationally intensive analytic tasks with stringent real-time data processing requirements, represented as $\mathbf{T} = T_1, T_2, \ldots, T_N$. These tasks vary in the number of requests (i.e., task popularity), data size, and computational capacity requirements. Consequently, they consider as computational intensive tasks for the limited computing resources available on end-user devices $\mathbf{U}$. Therefore, these $\mathbf{N}$ servers can provide the end-users/ applications with computing resources to help them process certain analytic tasks [12]. In this study, we address the scenario where the number of end-users $\mathbf{U}$ exceeds the number of tasks $\mathbf{T}$ ($\mathbf{U} > \mathbf{T}$). This scenario arises because certain computing tasks are highly popular and may be repeatedly requested by different

end-users. Consequently, we assume that different end-users can request the same task based on their preferences [39]. Additionally, the number of tasks $\mathbf{T}$ exceeds the number of EC servers $\mathbf{N}i$ ($\mathbf{T} > \mathbf{N}i$). As a result, tasks $\mathbf{T}$ are queued until a decision is made regarding their execution or offloading [20]. Given the limited capacity of EC servers to handle all tasks $\mathbf{T}$, it is imperative to make informed decisions regarding whether to execute tasks locally or offload them to other infrastructure. To determine the appropriate offloading decision for each $T \in \mathbf{T}$, each EC server n_i must obtain specific information regarding the analytic tasks T based on several key factors. **First**, the rate of task requests: A server n_i monitors the number of requests for each analytic task T_k, which arrive at different request rates λ_k from end-users or applications. By analyzing these request rates λ_k, the EC server n_i can identify the most popular tasks. Tasks with higher request rates are preferably executed locally to minimize delays. This also aids in predicting future task requests based on current demand and popularity. Additionally, server n_i can cache popular tasks for future reuse, potentially reducing response time and resource consumption. Less popular tasks may be offloaded to neighboring servers or the cloud. **Second**, according to the task request rate λ_k, some tasks are either extremely popular compared to other tasks or extremely rare. These are referred to as outlier tasks. Each server n_i can locally identify its own outlier tasks, as will be elaborated later. Generally, outlier tasks with high popularity (i.e., highly demanded tasks) will be executed locally. On the other hand, outlier tasks with very low popularity can be offloaded to an available neighboring node n_i or, if none are available, then they are offloaded to the cloud. The non-outlier tasks have all three options (local execution, offloading to available neighbor(s), or to the cloud). **Third**, as previously stated, each server n_i collects real-valued data and stores it locally in $\mathcal{D}_i$. It's worth noting that the type and amount of data in each EC server n_i significantly impact whether a task should be executed locally or offloaded. Since we primarily focus on analytical tasks (e.g., ML model training and inference), such tasks require a specified amount of data from $\mathcal{D}_i$ to be executed.

Imagine, for instance, an analytic task as a series of value-range queries, which define a specific data subspace over the server n_i's available data in $\mathcal{D}_i$. In this case, the analytic task T_k might require a large portion of the necessary data (e.g., >75%) found in server n_1, while only a small fraction (e.g., >10%) is available in another server n_2, as illustrated in Figure 2.2. Consequently, offloading such a data-driven task to server n_j may result in additional latency, increased resource usage, and the need for data transmission from n_1 to n_2.

In Figure 2.2, task T_1 is shown to be assigned to server n_2, which holds only 20% of the required data. This decision reflects a *non-optimal offloading scenario*, often caused by the lack of global knowledge about data distribution across the system. It highlights a core challenge in distributed environments: EC servers typically make decisions based on partial, local information, which can lead to suboptimal task placements and inefficient use of resources. This example motivates the need for a coordinated, context-aware offloading mechanism that jointly considers task popularity and data availability factors when selecting an execution node.

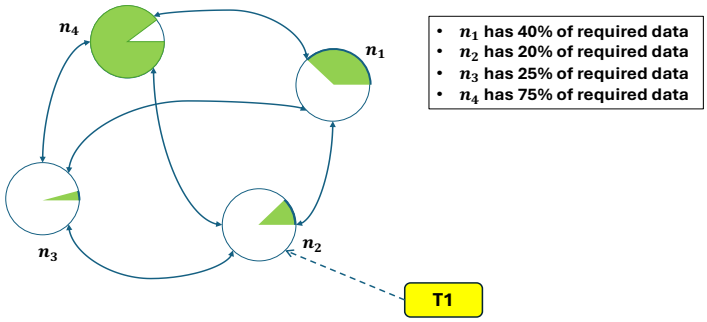


Figure 2.2: Example of the availability of Cached data.

Therefore, our mechanism considers the amount of accessible data required for a given task to make the offloading decision. Hence, given an incoming task T_k at EC server n_i, the server locally estimates the probability of this tasks to be outlier (based on a recent history of demand rates) and the percentage of available data required. This information is used by EC server n_i to come up with the first two decisions/actions: a_0 = 'local tasks execution' or a_1= 'task offloading', and if action a_1 is selected, then the server n_i should swiftly decide in which neighboring EC server $n_j \in \mathbf{N}_i$ the task T_k should be offloaded (action a_{11}) or offload that T_k to the cloud (action a_{12}), as in Figure 2.3.

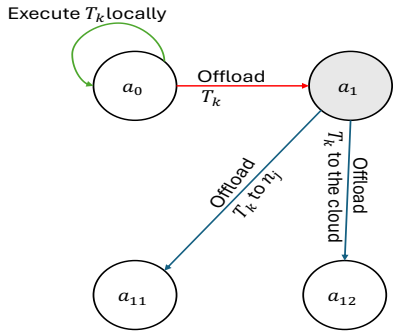


Figure 2.3: Tasks offloading decisions.

2.4 Methodology

In this section, our goal is to optimize the decision-making process for offloading and caching analytic tasks based on the proposed EC servers selection mechanism. That aims to *efficiently assign data-driven analytic tasks to EC/cloud to minimize the task execution delay and increase EC servers resource utilization*. We firstly introduce the tasks' management factors that have been adopted in this study. Then, the task management reasoning.

2.4.1 Task Management Factors

We detail the core factors of the proposed task management mechanism, which determines the optimal execution decision for each task T_k on each server n_k. These factors include *the popularity of tasks*, *outliers*, and *the corresponding data access availability*.

Task Popularity

We first elaborate on a methodology for determining the popularity of a task T_k in a specific server n_i within a sliding time window of size W. Specifically, we assume a discrete time domain $t \in \mathbb{T} = \{1, 2, \ldots\}$. At each time instance t, the server n_i observes a number of demands for each task T_k from end-users/applications. The demand for task T_k is associated with a request rate λ_k as requested by end-users/applications and monitored within the time window (horizon) W. Thus, given server n_i, a series of tasks $\{T_1, T_2, \ldots, T_k, \ldots\}$ arrive with rate λ_k. The demands for each task T_k in the W recent time instances are recorded in the task requests vector $\mathbf{v}_k = (v_{t-1+W}, v_{t-2+W}, \ldots, v_t)$, where v_{t-l+W} element indicates the number of the incoming requests of task T_k by end-users to server n_i at time instance $l = 1, \ldots, W$. The requests vector $\mathbf{v}_k$ over the time window W plays a significant role in storing the recent historical trends of each task T_k's demands, which will be used for estimating the popularity of the task T_k in a server n_i. To derive the popularity of T_k, the server n_i groups the corresponding task demands within the time window by adopting lightweight unsupervised clustering. The clustering algorithm divides the task demands of $\mathbf{v}_k$ into groups (clusters). Each cluster contains a set of demands that are similar to each other according to the task arrivals within the time window. Specifically, we adopt **Subtractive Clustering** [49] as unsupervised clustering algorithm. It is widely used in decision-making problems to categorize data into meaningful groups. In this chapter, Subtractive Clustering is used because it does not require the number of clusters to be known beforehand, making it well-suited for dynamic demand trends. As shown in Figure 2.4, this method clusters task requests over a time window into several groups, where each cluster has a centroid (cluster-head). The number and density of these clusters help quantify the task's popularity.

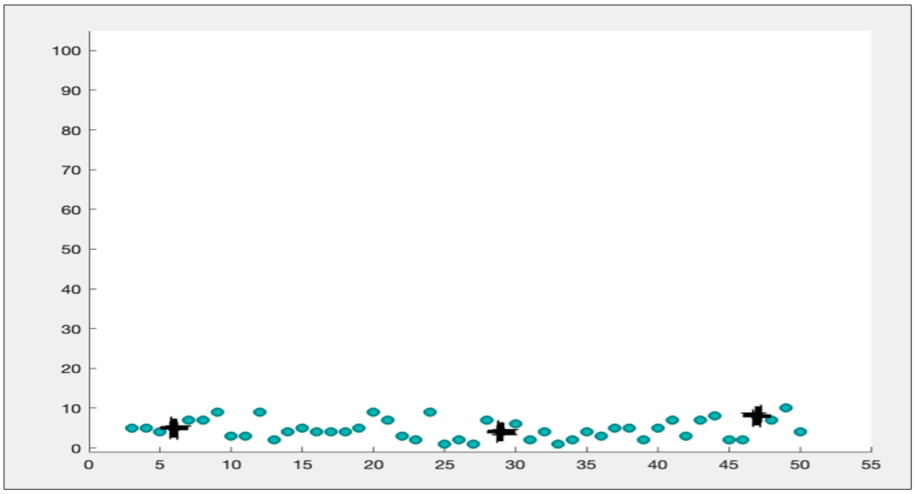


Figure 2.4: Tasks demand clusters using subtractive clustering ($\lambda = 5$, $\mathbf{W} = 50$). Cluster-heads are marked with **X**, and circles represent task requests.

To derive the set of clusters C over the task demand vector $\mathbf{v}_k$, we apply the subtractive clustering algorithm, which identifies high-density regions in the input space without requiring the number of clusters in advance. Specifically, for each observed demand value v_j in the time window W, the algorithm computes a potential score that reflects its closeness to other demand values. The potential of each point is given by:

$$P_j = \sum_{q=1}^{W} \exp\left(-\frac{(v_j - v_q)^2}{r_a^2}\right) \tag{2.1}$$

where r_a is a radius that defines the neighborhood size. The point with the highest potential is selected as the first cluster-head c_1. After selecting a cluster-head c_ℓ, the potentials of neighboring values are reduced to avoid choosing nearby points as subsequent cluster-heads:

$$P_q \leftarrow P_q - P_j \exp\left(-\frac{(v_q - c_\ell)^2}{r_b^2}\right) \tag{2.2}$$

where $r_b > r_a$ controls the suppression radius. This process continues iteratively until all cluster centers $\{c_1, \ldots, c_{|C|}\}$ are identified. Each cluster C_ℓ is then formed by assigning nearby demand values to the closest cluster-head c_ℓ, enabling the estimation of demand patterns for task T_k during the recent time window W.

After the subtractive clustering derives a set C of $|C|$ clusters over the demands of task T_k across the most recent $\mathbf{W}$ time instances, each cluster is represented by a task demands clusterhead $C_\ell \in C$, $\ell = 1, \ldots, |C|$. The clusterhead will help us in estimating the demands density for the arriving task T_k during the time window W and the amount of requests per cluster as will be elaborated below. Given these (recently historical) statistics, we can define the popularity of the task T_k based in its demanding behaviour within the time window. If the task is associated with

relatively many clusters within the specific time horizon W, then it is (statistically) considered more popular than other tasks associated with less clusters. Moreover, the more clusters are derived from the clustering, the higher the variability and amount of task demands with different rates occur during the time window. Therefore, we define the cluster density, which indicates the amount of demands of the T_k within a specific time duration. Specifically, consider the ℓ-th cluster C_ℓ, which maintains the demand values $v_j \in C_\ell$ and represented by the cluster-head demand c_ℓ. We then define the cluster variance $\sigma_\ell^2 = \frac{1}{|C_\ell|}\sum_{v_j \in C_\ell}(v_j - c_\ell)^2$. Hence, we introduce the cluster density d_ℓ as the amount of the task demand values being within a squared distance of the cluster variance from the corresponding clusterhead (centroid), i.e.,

$$d_\ell = |v_j \in C_\ell : (v_j - c_\ell)^2 \leq \sigma_\ell^2|. \tag{2.3}$$

The density d_ℓ represents the number of task demands within a distance from the centroid that is less than the deviation. The deviation σ_ℓ is adopted to define our strategy concerning; if we want to be very 'strict' and require many demand values to be very close to the centroid to conclude a high density. Within a cluster, there are historical demand values for task T_k observed over the most recent W time instances. We pay significant attention to the clusters exhibiting a high density around the centroid. This density is strong evidence that multiple task demand values are concentrated around the centroid. To aggregate the demand information conveyed by the clusters, we define a weighting scheme that assigns a high weight to clusters with high density. Specifically, based on the derived clusters, the popularity index p_k for the tasks T_k is defined as the linear combination of the derived clusters weighted by their normalized densities.

$$\tilde{d_\ell} = \frac{d_\ell}{\sum_{j=1}^{|C|} d_j} \tag{2.4}$$

and thus, the popularity demand index p_k for task T_k within the recent W time instances across all the derived clusters is defined as:

$$p_k = \sum_{\ell=1}^{|C|} \tilde{d_\ell} c_\ell. \tag{2.5}$$

Outlier Tasks

To support the decision-making process in the proposed mechanism, we introduce the concept of the task outlier. Classifying a task as an outlier helps identify statistically extreme (unusual) demands of tasks T within the recent W time instances. This mechanism is responsible for annotating some tasks as outliers based on their popularity compared to other tasks on a server. The outlier tasks are divided into two classes: outliers that have relatively very high popularity than the usual trend, and outliers that have relatively very low popularity. These classifications are based on a lightweight process using the Median Absolute Deviation (MAD) around the

median across the popularity indices of the tasks in $\mathcal{T}$ requested in node n_i over the last W time instances. The median of the popularity indices $\tilde{p}$ is used as a separating point between the high and low popularity tasks. Based on the popularity median $\tilde{p}$ over the popularity values $\{p_1, \ldots, p_M\}$, we can then calculate the MAD of the tasks $\mathcal{T}$:

$$\text{MAD}(\mathcal{T}) = median_{k=1,\ldots,M}(|p_k - \tilde{p}|) \tag{2.6}$$

Given this statistic, we define the outlier indicator I_k for task T_k based on its popularity p_k as:

$$I_k = \frac{|p_k - \tilde{p}|}{\text{MAD}(\mathcal{T})}. \tag{2.7}$$

The task T_k is outlier if I_k is greater than the empirically derived threshold $\phi = 2.5$, i.e., the outlier indicator is:

$$o_k = \begin{cases} 1 \text{ (outlier)} & \text{if } I_k \geq \phi \\ 0 \text{ (non-outlier)} & \text{if } I_k < \phi \end{cases} \tag{2.8}$$

Based on the outlier tasks identification and the associated popularity indices of these tasks, the server n_i can get more certain decisions, either locally executing a very popular (outlier) task or offloading a very low popular (outlier) task. Nonetheless, the amount of data required for those outlier tasks (and of course of all the tasks), will further help the server to proceed with a right offloading decision as it will be elaborated later. As an informal guideline, the outliers filter selects those tasks with high popularity while having a high data overlapping with the servers thus can select action a_0. In contrast, very low popular tasks with low data overlapping will select action a_{12}.

Task's Data Overlapping

To further support the decision-making process in the proposed mechanism, we introduce the concept of data overlapping, which indicates an estimation of the percentage of data required for executing the analytic task T_k out of the entire dataset $\mathcal{D}_i$. In this research, we focus on analytic tasks such as training machine learning models on distributed data, which has gained popularity and proven beneficial in recent years for applications like weather forecasting, traffic prediction, and more [12]. In this context, for instance, data points x in a node n_i represent real-values, such as sensed data collected from IoT devices. These data form the basis for determining the suitability of executing an analytic task T_k locally in node n_i. However, the availability of required data for each task varies from node n_i to node n_j. Therefore, if a task T_k offloaded to n_i has only 20% of the data it needs for execution, it implies that 80% of the required data would need to be brought in to execute the task locally. This situation can lead to increased resource consumption, network load, and response time. Given the representation of an analytic task T_k via a (range) selection query $\mathbf{q}_k = [q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}]$ over a data sub-space defined be

the dataset $\mathcal{D}_i$, we define the data overlapping as the ratio of the data points satisfying the task query $\mathbf{q}_k$ out of the data points stored in node's dataset $\mathcal{D}_i$. That is, a data point $\mathbf{x} \in \mathcal{D}_i$ satisfies the range query $\mathbf{q}_k$ if the following statement $\mathcal{S}(\mathbf{q}_k, \mathbf{x})$ holds true:

$$\mathcal{S}(\mathbf{q}_k, \mathbf{x}) \equiv (q_1^{\min} \leq x_1 < q_1^{\max}) \wedge \ldots \wedge (q_d^{\min} \leq x_1 < q_d^{\max}) \tag{2.9}$$

Hence, the degree of data overlapping u_k of task T_k represented via the query $\mathbf{q}_k$ is defined as:

$$u_k = \frac{|\mathbf{x} \in \mathcal{D}_i : \mathcal{S}(\mathbf{q}_k, \mathbf{x}) \equiv \texttt{TRUE}|}{|\mathcal{D}_i|} \tag{2.10}$$

2.5 Task Management Reasoning

2.5.1 Fuzzy Logic Inference Modeling

Given a EC server n_i receiving demands for the analytics task T_k over the EC server's dataset $\mathcal{D}_i$, we derive the corresponding factors: popularity p_k, outlier o_k, and data overlapping u_k. This section introduces a reasoning mechanism that integrates these factors to guide task offloading decisions, balancing between the task demands, EC servers' capability and data availability. To facilitate decision-making under uncertainty, FLI inference has been adapted to handle the inherent uncertainty and approximation of these factors in dynamic environments. Since it is one of the most prevalent strategy for dealing with rapid change in uncertain systems [14, 50]. This is achieved by adapted fuzzy inference rules with linguistic variables, effectively modeling the uncertainty associated with these factors. Performing such inference locally on a EC server offers several advantages. **First,** it can easily cope with multi-criteria decision-making models by incorporating multiple factors in the same model. **Second,** it is capable of dealing with uncertainty in a dynamic context without complex mathematical models. **Third,** it offers lightweight computational complexity while providing an explainable decision-making methodology [48]. This explainability is based on the linguistic variables which reflect the uncertainty derived from the values of the factors p_k, o_k, and u_k. In this context, the popularity (fuzzy variable) is associated with three linguistic fuzzy values $\{\text{High}, \text{Medium}, \text{Low}\}$ reflecting a high, medium, and low value of popularity for a specific task. Similarly, the data overlapping (fuzzy variable) is associated with the linguistic values $\{\text{High}, \text{Medium}, \text{Low}\}$ reflecting a high, medium, and low value of data overlapping derived from the task's query data subspace over node's data space. The outlier indicator o_k (as a fuzzy variable) takes two linguistic values $\{\text{Yes}, \text{No}\}$ reflecting whether T_k is outlier or not, as depicted in Figure 2.5. Given a linguistic value linked to a fuzzy variable, a membership function $\mu : \mathbb{R} \rightarrow [0, 1]$ is defined in order to indicate the possibility that a value of the variable belongs, at certain degree, to the linguistic value. Specifically, given a data overlapping value $u_k = x$, we associate this value with the linguistic value High via the membership function $\mu_u^H(x) \in [0, 1]$. For instance, if the data

overlapping $u_k = 0.7$ for task T_k, then this can possibly be considered as a High data overlapping with possibility $\mu_u^H(0.7) = 0.88$. We similarity define these membership functions for the rest of the linguistic values for all the factors. There are different membership functions forms that can be adapted for fuzzy based reasoning, such as trapezoidal, piecewise linear, singleton, triangular, and Gaussian [51]. In our context, we consider the triangular form to represent the membership functions, which is considered as the most common form according to [46]. Summarizing, we obtain the next sets of membership functions of the fuzzy linguistic values for task popularity p_k, data overlapping degree u_k and outlier indicator o_k, respectively:

$$F_{u_k}(x) = \{\mu_{u_k}^L(x), \mu_{u_k}^M(x), \mu_{u_k}^H(x)\} \tag{2.11}$$

$$F_{o_k}(x) = \{\mu_{o_k}^{No}(x), \mu_{o_k}^{Yes}(x)\} \tag{2.12}$$

$$F_{p_k}(x) = \{\mu_{p_k}^L(x), \mu_{p_k}^M(x), \mu_{p_k}^H(x)\}$$

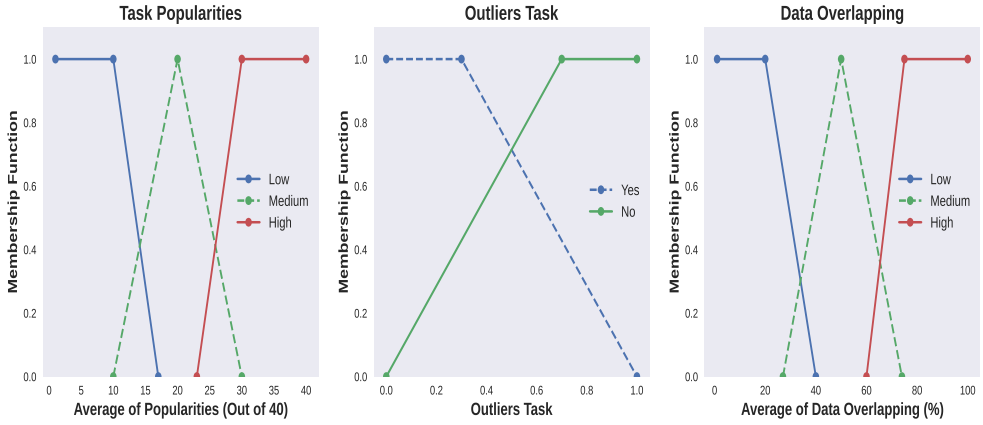


Figure 2.5: Implementation of FLI on our three factors.

2.5.2 Two-stage Fuzzy Logic-based Reasoning

Given the set of membership functions, we introduce a novel two-stage FLI reasoning engine that makes the decision of task execution locally (actions a_0), offloading to another EC server $n_j \in \mathcal{N}_i$ (action a_{11}) or offloading to the cloud (action a_{12}). Handling all these decisions in a single stage FLI is a complicated operation [48]. Therefore, we have adapted a two stage FLI system in order to reduce the system complexity . The first inference stage (S1) deals with the decisions (actions) a_0 ='local task execution' and a_1 ='task offloading'. The output of S1 is the offloading probability for a task T_k given the input p_k, u_k, and o_k, as will be elaborated later in this section. The second inference stage (S2) is based on the S1's output. In particular, if a_1 action is selected (having the highest probability), then EC server n_i swiftly decides in which neighboring EC server $n_j \in \mathcal{N}_i$ the task T_k should be offloaded (action a_{11}), or offload T_k to the cloud (action a_{12}). The proposed tow-stage reasoning mechanism runs on a specific EC

server n_i which plays the role of the 'leader' in the neighborhood $\mathcal{N}_i$. This role is periodically assigned to EC servers from the neighborhood when certain criteria are met, e.g., remaining energy, computational capacity and communication availability. This assignment is achieved via certain leader election mechanisms. We do not elaborate on these mechanisms, since it is beyond of the scope of this chapter. In the remainder, for simplicity of notation, we assume that node n_i is assigned with this leadership role to execute the Two-stage reasoning engine, where all neighboring EC servers $n_j \in \mathcal{N}_i$ directly communicate with their leader n_i. Both stages of the FLI system essentially follow the same steps, but with varying numbers of tasks. The first stage handles all tasks, whereas the second stage focuses exclusively on the subset of tasks that could not be executed locally.

First Stage Reasoning (S1)

The S1 reasoning engine on n_i for each task T_k goes through the following steps: The first step of FLI is fuzzification of the inputs (p_k, o_k, u_k) into their fuzzy linguistic terms via the membership functions. It takes all these factors as numerical values (crisp values), then it assigns each value to the corresponding fuzzy values (e.g., Low, Medium, High) [14, 46]. The second step is the activation of the Fuzzy Inference Rules (FIRs), which interpret the logic behind the decision making for the offloading probability. The obtained fuzzy values are then used to activate a set of FIRs, a.k.a., fuzzy knowledge base. Each FIR is represented via an IF-THEN statement [48]. The antecedent part ('IF' part) is a set of logical conjunctions over the fuzzy linguistic variables. The consequent part ('THEN' part) of the FIR is a fuzzy term from the set of linguistic terms {Low, Medium, High} that expresses the offloading probability r_k. The generic format of the FIR statements used in our S1 engine is as follows:

$$\begin{aligned} &\text{IF } p_k \text{ IS } X_1 \text{ AND } o_k \text{ IS } X_2 \text{ AND } u_k \text{ IS } X_3 \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN } r_k \text{ IS } X_4 \end{aligned} \tag{2.13}$$

where the linguistic terms $X_1, X_3, X_4 \in \{\text{Low, Medium, High}\}$ and $X_2 \in \{\text{No, Yes}\}$. For instance, the following FIR:

$$\begin{aligned} &\text{IF } p_k \text{ IS HIGH AND } o_k \text{ IS YES AND } u_k \text{ IS HIGH} \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN } r_k \text{ IS LOW.} \end{aligned} \tag{2.14}$$

This rule expresses the decision of the task T_k to be offloaded with low probability, i.e., action a_0 is preferred more than action a_1, due to the fact that this task has very high popularity (thus being also an outlier) and the data required by this tasks can be fully available to EC server n_i (high degree of overlapping). Hence, in this case, T_k can be locally executed on EC server n_i and not being offloaded (i.e., low offloading probability). Our S1 engine requires

18 FIRs in the fuzzy knowledge base in order to cover the whole decision space; there are $3 \times 2 \times 3 = 18$ membership function involved in the three fuzzy variables: popularity, outlier and data overlapping, respectively. The FIRs of S1 engine are provided in Table 2.1, which reflects the reasoning behind the decision on the actions a_0 or a_1 represented via the offloading probability. The last step of S1 is the defuzzification of all the offloading probability values of

Table 2.1: FLI rules inputs and the expected outputs.

R_i	p_k	o_k	u_k	r_k
1	Low	Yes	Low	High
2	Low	Yes	Medium	Medium
3	Low	Yes	High	Medium
4	Low	No	Low	High
5	Low	No	Medium	High
6	Low	No	High	Medium
7	Medium	Yes	Low	Medium
8	Medium	Yes	Medium	Medium
9	Medium	Yes	High	Low
10	Medium	No	Low	High
11	Medium	No	Medium	Medium
12	Medium	No	High	Medium
13	High	Yes	Low	Medium
14	High	Yes	Medium	Low
15	High	Yes	High	Low
16	High	No	Low	Medium
17	High	No	Medium	Low
18	High	No	High	Low

the activated FIRs [46, 48], which results in a scalar probability $r_k = P(a_1)$ for the task T_k. There are certain defuzzification operators for deriving scalar output over the activated FIRs. We adapt the centroid defuzzifier, which not only is considered as the most common operator but also the defuzzified value directly represents probability, which is aligned with the notion of r_k, calculated as:

$$r_k = \frac{\int_{x\in[0,1]} x\mu^{\nu}_{r_k}(x)}{\int_{x\in[0,1]} \mu^{\nu}_{r_k}(x)}, \tag{2.15}$$

where ν is a {Low, Medium, High} linguistic terms of the offloading probability. The defuzzified offloading probability r_k ranges between 0% and 100%. In order to transform this probability to a decision, as Figure 2.6 depicts, we define the decision threshold to be 30%, that is, if $r_k = P(a_1) \leq 0.3$ (i.e., $P(a_0) > 0.7$), then, node n_i locally executes the task T_k. That means T_k will be processed locally by the leader n_i.

The second stage of inference is introduced to deal with decision making on those tasks which have been determined to be offloaded as suggested by the S1 inference engine, i.e., their probability of offloading $r_k > 0.3$. After finishing S1, the leader EC server n_i has two types of tasks: those that should be locally executed belonging to the set $\mathcal{T}_0 \subset \mathcal{T}$ (associated with the action a_0), and tasks belong to the set $\mathcal{T}_1 \subset \mathcal{T}$ (associated with the offloading action a_1). The aim of the $S2$ inference engine is to proceed with decision over the tasks in $\mathcal{T}_1$ under the actions:

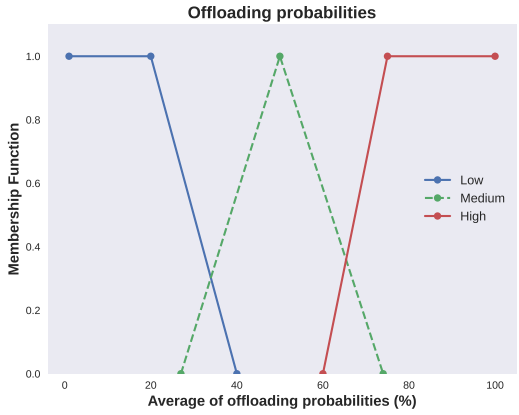


Figure 2.6: The probability of offloading.

a_{10} (offload to another node) or a_{11} (offload to the cloud). Hence, $S2$ passes through two steps: tasks information updating and determine offloading probability. If the leader EC server n_i has not had enough resources for $S2$ inference, then another EC server $n_j \in \mathcal{N}_i$ can be elected as a leader according to specific leader election mechanisms. In this case, the old leader n_i just sends only the tasks $\mathcal{T}_1$ to the new leader n_j in order to make the offload decisions.

Tasks information updating: The leader EC server n_i collaborates with its neighbors to update the information regarding the tasks in $\mathcal{T}_1$ based on the $S2$ engine. For each task, a neighboring available and suitable EC server can be assigned based on the following reasoning. Firstly, the leader n_i considers the task contextual information (p_k, o_k, u_k) for each task $T_k \in \mathcal{T}_1$ from each neighbouring $n_j \in \mathcal{N}_i$. The goal is to determine how popular a task $T_k \in \mathcal{T}_1$ is and how much data access it requires in EC sever n_j. Once n_j receives the request from leader n_i, it sends over (p_k, o_k, u_k) for each task in $\mathcal{T}_1$ according to its dataset $\mathcal{D}_j$. In turn, n_i compares between the its tasks information and neighbor's tasks information. When n_i receives information from n_j, it then has two tables: the main table that it gets from S1 and a new one that it receives from n_j. It then updates the $\mathcal{T}_1$ tasks information from n_j based on the following rule given a task. The rule states that the task T_k's information $(p_{k.i}, u_{k.i})$ in leader EC server will be updated if the corresponding values from the neighbouring n_j are greater; otherwise, the task's information will not be updated. The rationale behind updating tasks information is based on achieving lower r_k. Therefore, if the p_k and u_k are not greater than the ones in leader EC server, that means r_k will increase, and this leads to increase the probability of offloading task to unsuitable EC server or to the cloud. In order to avoid this, they will not be updated. This process will be repeated for all the tasks getting information form each neighboring EC server $n_j \in \mathcal{N}_i$ sequentially. Hence, with each received task information from the next node $n_{j+1} \in \mathcal{N}_i$, if the rule is fired, the leader's task information keeps updating. By the end of this process, leader n_i will have updated all the required information as shown in Table 2.2 according to the most suitable node.

Table 2.2: T_k Information updating according to suitable node.

Task	Old information	Update to	New information
T1	8.37, Yes, 15%	n2	9.64, No, 31.5%
T3	2.69, Yes,45.5%	n4	2.01, Yes, 85.4%
T4	14.8, Yes, 67.8%	n2	13.71, No, 52.89%
T6	7.563,Yes, 31.8%	n3	5.88, No, 52.88 %
T7	14.848, Yes, 15.67%	n2	26.45, No, 70%
T10	8.5,Yes, 21.3%	n4	7.25, No, 66.2%

Determine the offloading probability: Even though leader has updated with the most suitable EC server for each task, still there is a need to execute the $S2$ fuzzy inference engine for those tasks in $\mathcal{T}_1$, since this updating process only determines the best place for a task across neighboring EC servers $n_j \in \mathcal{N}_i$ regardless of the r_k. Meanwhile, $S2$ is applied in order to get a specific r_k for each task in $\mathcal{T}_1$. If r_k is low, the (action a_{11}) is decided, otherwise, (action a_{12}) is preferred. Finally, the updated task information will be treated as input for $S2$ and it will pass through the same steps as in $S1$, i.e., fuzzification, activation of FIR and defuzzification. By introducing $S2$, it helps the leader n_i to decide clearly *where* each task T_k should be executed according to the corresponding r_k comparing with $S1$ inference engine.

2.6 Performance Evaluation

In this section, we employ synthetic datasets to simulate task popularities, while the data overlapping experiment is conducted using real datasets and analytics queries. Additionally, we utilize the CloudSim Plus simulator to assess the impact of our mechanism on upload/download data rates for each task and resource utilization.

2.6.1 Experimental Setup for Tasks' Popularities and Data Overlapping

This experiment has been applied on two types of datasets: real datasets and synthetic datasets. The real datasets were collected by four Unmanned Surface Vehicles (USVs) operating as nodes $\mathbf{N}$, which gather sensor data from a coastal area GNFUV ([2], accessed on 20 October 2020). Each USV node n_i maintains a neighborhood $\mathcal{N}_i \subset \mathcal{N}$ of directly communicating nodes $n_j \in \mathcal{N}_i$. Furthermore, node n_i interacts with end-users or applications to collect data, storing it locally in their datasets $\mathcal{D}_i$ for predictive analytic tasks. The data comprises two features: sea surface temperature and humidity, i.e., $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^d$. One node $n_i \in \mathcal{N}$ acts as the leader, receiving a set of analytic tasks T_k and deciding whether T_k should be executed locally (action a_0) or offloaded (action a_1).

[2] https://archive.ics.uci.edu/dataset/452/gnfuv+unmanned+surface+vehicles+sensor+data

At the beginning, the leader n_i has received ten analytic tasks and is evaluating three factors: popularity (p_k), overlap (o_k), and utility (u_k)—for each task T_k. For task popularity (p_k), a synthetic dataset of task demands for each T_k is generated using Poisson distribution with different rates (λ_k) over a time window of size $W = 150$. The Poisson distribution is a commonly used tool for generating a set of requests according to a specific rate.

After constructing the request vector $\mathbf{v}_k = (v_{t-1+W}, v_{t-2+W}, \ldots, v_t)$ for each task T_k, the Subtractive Clustering algorithm is applied to group task demands based on their similarity. The cluster density d_ℓ for each cluster C is then calculated using Equation (2.3). The leader n_i determines T_k's popularity, as shown in the second column of Table 2.3.

Table 2.3: table: Tasks popularities.

T_k	p_k	o_k	u_k
T_1	8.37	No	67.8%
T_2	28.31	No	15.67%
T_3	2.69	Yes	15.46%
T_4	14.848	No	22.88%
T_5	29.977	No	81%
T_6	7.563	Yes	6.8%
T_7	26.848	No	31.8%
T_8	39.49	Yes	21.3%
T_9	34.399	Yes	69%
T_{10}	8.5	No	45.5%

Regarding the outliers indicators o_k, the statistical threshold $\phi = 2.5$ is applied, allowing n_i to generate the two sets of outliers and non-outliers, as shown in the third column of Table 2.3. To obtain the task's data overlapping u_k, each local dataset $\mathcal{D}_i$ is defined by the feature boundaries max and min values: $\mathcal{D}_i = [x_1^{\min}, x_1^{\max}, x_2^{\min}, x_2^{\max}]$. Then, queries $\mathbf{q}_k$ are generated uniformly at random for ten tasks, such that $\mathbf{q}_k = [q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}]$ for each task T_k, in order to obtain the data subspace needed for the execution of analytic task T_k, as shown in Table 2.4.

Table 2.4: Queries generation and percentages of data overlapping .

Task	$q_1^{\min}$	$q_1^{\max}$	$q_d^{\min}$	$q_d^{\max}$	Points Including	Per (%)
T_1	19	32	49	57	130/899	14.46%
T_2	19	29	44	46	164/899	18.24%
T_3	26	28	43	58	75/899	8.3%
T_4	22	32	42	53	160/899	17.79%
T_5	20	32	38	58	885/899	98.44%
T_6	20	29	41	55	310/899	34.48%
T_7	21	25	48	53	48/899	5.33%
T_8	22	33	38	55	600/899	66.74%
T_9	20	32	50	57	470/899	52.28%
T_{10}	19	28	36	50	251/899	27.91%

Evidently, there are some tasks T_k with high data overlap (e.g., T_5); u_k reaches 98%, while there are tasks with low u_k, such as T_3 and T_7. Therefore, by executing tasks with high u_k locally, such as T_5, it is expected to reduce the percentage of data offloading from 100% to 2%. In

contrast, executing tasks with low u_k locally, such as T_7, is expected to increase data offloading percentages to almost 95%, which is clearly inefficient. The FLI engine has been developed in MATLAB considering the popularity p_k of tasks T_k between [1, 40] and outlier o_k either 0 or 1, while the percentages of data overlapping u_k are between [0%, 100%]. All these are inputs to the FL system, while the probability of offloading r_k is the output in [0%, 100%]. As shown in Figure (A- 2.7), increasing p_k and u_k for task T_k leads to a decrease in the probability of offloading r_k. This implies an increase in the probability of executing this locally (action a_o). On the other hand, in Figure (B- 2.7), decreasing p_k and u_k for task T_k leads to increasing the probability of offloading r_k. This means that increasing the probability of offloading T_k either to another node (action a_{11}) or to the cloud (action a_{12}).

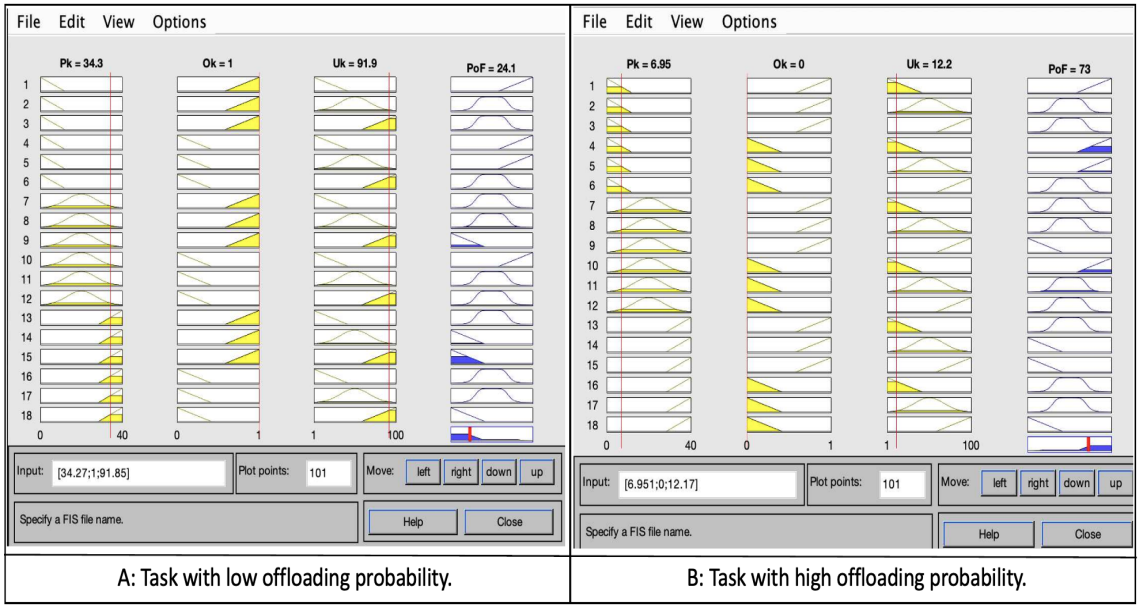


Figure 2.7: The effect of (p_k, o_k, u_K) on the probability of offloading (r_k).

For task information update, ten tasks T_k are considered, six of which require offloading (action a_1) according to Table (2.5). To determine the most suitable node $n_j \in \mathcal{N}_i$ for each task T_k, the information (p_k, o_k, and u_k) is updated.

During $S2$, leader n_i will have the task information as shown in Table 2.5. The output of $S2$ is r_k for each task in $\mathcal{T}_1$. S_2 is applied with the same steps as in $S1$. According to the results, T_1, T_3, T_6, and T_7 will offload to $n_j \in \mathbf{N}_i$ (action a_{11}), while T_4 will offload to $n_j \in \mathbf{N}_i$ (action a_{11}) if there are available resources. However, tasks T_4 and T_{10} have a very high offloading probability; therefore, they will be offloaded to the cloud (action a_{12}).

Table 2.5: S_2 decision making based on three factors.

Task	Node	P_k	o_k	u_k	r_k	Rule
T1	n_2	Low	No	Low	82%, High	rule 4.
T3	n_4	Low	Yes	High	49%, Medium	rule 3
T4	n_3	Medium	No	medium	58%, Medium	rule 11
T6	n_2	Low	No	Medium	84.4%, High	rule 5
T7	n_3	High	No	Medium	44%, Medium	rule 11
T10	n_2	Low	No	Medium	51%, Medium	rule 5

2.6.2 Experimental Setup

CloudSim Plus has been utilized to create the considered scenarios and to evaluate the performance of our mechanism. In this experiment, two types of parameters have been considered: data-driven task characteristics and EC/cloud parameters. Data-driven tasks characteristics vary according to the nature of tasks. Some tasks are affected by delays, while others are not; some tasks could execute on EC servers, while others are beyond EC servers' capabilities and should be offloaded to the cloud. To simulate real-life scenarios, ten different data-driven tasks (applications) have been used. To decide the application types, we looked at the most common data-driven tasks (weather prediction, air pollution prediction, traffic jam prediction, compute-intensive tasks, and health apps, etc.).Table 2.6 contains tasks information chosen based on [48]. The upload/download data size represents the type of data sent/received from EC/Cloud since it could increase or decrease according to data overlapping percent, and this is what distinguishes our mechanism against other task offloading mechanisms. For instance, (50,000 MB, 100 MB) denotes the size of uploaded data (humidity, temperature, wind, etc.) that will be used to build a ML model, and downloads depict the model that the application will receive as a result of data collection and training in EC/cloud computing. According to our mechanism, if the data overlapping percentage is high (e.g., 90% or more) the uploading data could be reduced from 50,000 MB to 10 MB. Task length determines the number of Million Instructions (MI) and the required CPU resource to complete a data-driven task. We considered ten tasks arriving at n_i with specific features, which include task length and upload/download data. According to data overlapping, we made the range of this parameter fluctuate from low values with some tasks to high values with others, while resource consumption and task delay sensitivity have been set up according to the applications indicated in [48].

Table 2.6: Application types used in the simulation.

Task	Application	Task Length	Upload/Download Data
T1	Deep learning	10,000	50,000/100
T2	Traffic jam prediction	20,000	200,000/300
T3	Air pollution prediction	15,000	200,000/400
T4	Healthcare diagnosis	30,000	80,000/100
T5	Weather prediction	8500	50,000/50
T6	Compute-intensive task	20,000	300,000/500
T7	Fraud detection	18,000	300,000/250
T8	Virtual assistants	25,000	20,000/50
T9	Alerting And Monitoring	14,000	100,000/300
T10	Social Media Analysis	21,000	60,000/80

Other simulation parameters that reflect the computational capabilities of EC/Cloud servers, such as bandwidth, the number of Virtual Machines(VM), and host MIPS, are listed in Table 2.7.

Table 2.7: Simulation parameters.

Parameters	EC	Cloud
Bandwidth	WAN 500 MB/s	LAN 10 GB/s
Number of VM	2	8
Number of cores	2	8
VM CPU speed	10 MB	100 MB
HOST MIPS	1000	10,000

2.6.3 Comparative Assessment

First, we evaluate the efficiency of the proposed mechanism by considering both tasks' polarities (p_k) and data overlapping (u_k). We compare its effectiveness against two alternative mechanisms designed to handle the same tasks and datasets. The first alternative mechanism (M_2) [41] considers only tasks' popularity (p_k) and outliers (o_k) when making offloading decisions. The second mechanism (M_3) focuses solely on the percentages of data overlapping (u_k) between the tasks (T_k) and EC servers ($n_j \in \mathbf{N}_i$). The experimental results in Table 2.8 demonstrate the performance of the proposed mechanism, referred to as (M_1). According to the optimal solution (OS) shown in the last column of the same table, (M_1) achieves the highest performance. As shown, (M_2) focuses on task popularity (p_k) in each EC server while completely ignoring data overlapping (u_k). This means that the (M_2) mechanism will distribute tasks (T_k) among the servers ($n_j \in \mathbf{N}_i$) without considering whether the selected EC server can reduce response time and resource consumption. On the other hand, (M_3) only considers the percentages of data overlapping (u_k). It uses the node's data efficiently, regardless of task popularity. Consequently, popular or urgent tasks (T_k) could be offloaded to the cloud (action 12) or remain in the execution queue if they have lower data overlapping with the servers ($n_j \in \mathbf{N}_i$). (M_1) aims to balance both task popularity (p_k) and data overlapping (u_k). The results show that (M_1) provides accurate

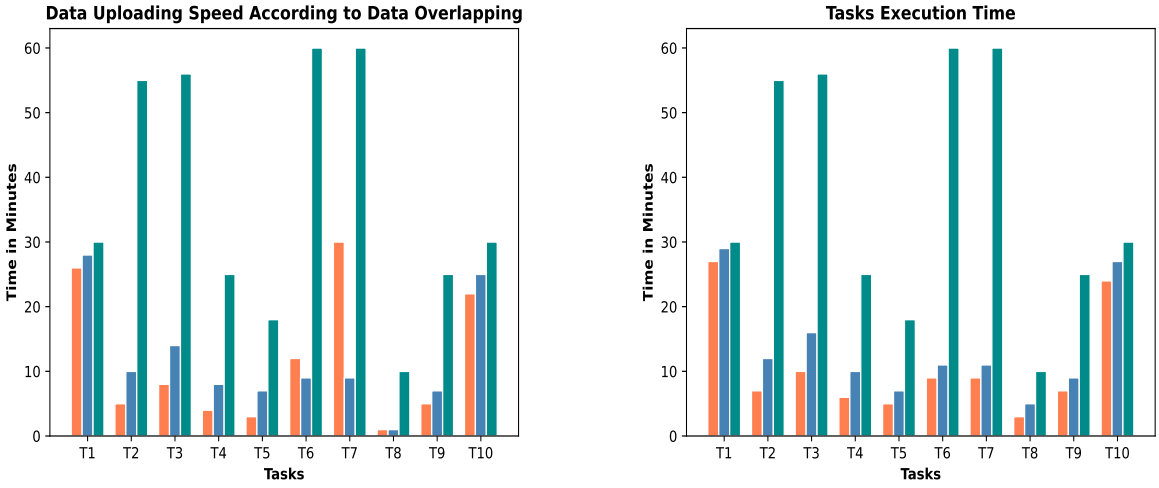
offloading probabilities (r_k) close to 90% according to the OS boundaries, whereas (M_2) and (M_3) achieve accuracies of 70% and 60%, respectively.

Table 2.8: The probability of offloading u_k for each task T_k according to our mechanism compared to the other two mechanism.

P_k	o_k	u_k	r_k	M_1	M_2	M_3	OS
T1	Low	Yes	Low	83%	84%	57%	High
T2	Med	No	High	30.4%	32%	42%	Med
T3	Low	No	Low	85%	86%	72.92%	High
T4	Med	No	med	65%	68.9%	53%	Med
T5	High	No	High	23%	17.6%	35%	Low
T6	Low	Yes	Low	85%	86.5%	72.7%	High
T7	Med	No	Med	44.7%	37%	47%	Med
T8	High	Yes	High	14.4%	14.5%	17.7%	Low
T9	High	Yes	High	15%	15.8%	27.1%	Low
T10	Low	Yes	Low	83%	85%	67.2%	High
-	-	-	-	10/10	8/10	6/10	-

Second, in terms of resource utilization, we compare the effectiveness of our mechanism against two alternative mechanisms under the same task simulation conditions. The first alternative is a cloud-based mechanism [37], where EC servers collect sensor data and send it to the cloud to reduce the energy consumption of sensors, which would otherwise occur if the data were sent directly to the cloud. The second mechanism is an EC-based approach, as suggested in several studies [14, 48], where tasks are sent to the EC server with the highest availability, bandwidth, and sensitivity to task delay.

Simulating our mechanism resulted in high data upload speeds, ranging from one to ten minutes. In contrast, the upload speed in the cloud-based model ranges from 28 to 60 minutes, while the upload speed in the EC-based mechanism, which does not consider data overlapping, is nearly double the speed achieved with our mechanism (see Figure 2.8 (a)) .



(**a**) Data uploading speed. (**b**) Data-Driven tasks execution time.

Figure 2.8: Data uploading speed and tasks execution Time.

In terms of execution time, we have considered both data offloading time and the main execution time. This is because, in data-driven tasks, data are an integral part of task execution. (Figure 2.8(b)) shows that the execution time is significantly reduced compared to the cloud-based model.

2.7 Conclusions

In this work, we introduced a mechanism for data-driven analytics tasks in an EC environment to efficiently exploit resources and reduce response time. This mechanism focuses on three factors: task popularity, outliers, and data overlapping to make execution and caching decisions for each task. Task popularity investigates each task's demands, while outliers identify statistically extreme (unusual) task demands. Data overlapping examines the percentage of data overlap between tasks and access nodes. These three factors are input into a two-stage FLI (Fuzzy Logic Inference) system to make the final decision for each task. The performance of our mechanism has been evaluated based on the probability of offloading data-driven analytics tasks to the appropriate nodes, compared to the optimal solution and two other mechanisms. The results demonstrate that our mechanism significantly outperforms the benchmark mechanisms in decision-making accuracy. Furthermore, this mechanism can reduce the probability of a task being offloaded to an unsuitable node by up to 95%.

Our method has also been evaluated in terms of resource utilization, showing that it provides higher data uploading speeds compared to EC-based and cloud-based methods. Consequently, data-driven analytic task execution times have been minimized.

2.8 Limitations & Directions of Enhancement

This mechanism offers significant benefits in reducing data offloading rates and addressing data privacy concerns. However, its applicability is limited in scenarios where data privacy is strictly regulated, such as with medical records, financial data, or personal information, where data sharing with third parties is prohibited. Privacy regulations in these areas restrict the use of this mechanism, a limitation that is expected to become more pronounced as awareness of data privacy continues to grow across various domains.

Moreover, this mechanism may not achieve high generalization in final models since models are trained independently without collaborative learning. Additionally, the mechanism focuses on specific end-user tasks that must be executed within a defined timeframe, leaving devices with only two options: either having sufficient resources to execute the task or not. This binary approach overlooks the potential for devices to contribute partially to task execution, such as training local models on local data.

To address these limitations, FL and Distributed Learning have been proposed. These approaches eliminate the need for data offloading, enable complete task execution on end-user devices, and enhance model generalization through the aggregation of local models.

Therefore, the subsequent three chapters of this thesis concentrate on distributed learning and federated learning. These approaches align with the growing emphasis on data privacy, the rapid expansion of data, the demand for models with greater generalization and efficiency, and the necessity for real-time task execution.

Table 2.9: Table of Symbols for Chapter 2.

Symbol	Definition
N	Number of EC servers
n_i	i-th EC server
D_i	Dataset stored locally on server n_i
x_k	A real-valued data point from dataset D_i
λ_k	Request rate for task T_k
W	Sliding time window
C_ℓ	Cluster head for demand cluster
σ_ℓ^2	Cluster variance
d_ℓ	Cluster density
a_0	Local task execution decision
a_1	Task offloading decision
a_{11}	Offloading task T_k to a neighboring EC server
a_{12}	Offloading task T_k to the cloud
T_k	A task in set of computational tasks T
$\mathcal{N}$	Set of neighboring EC servers
P_k	Task Popularity
o_k	Outlier Tasks
u_k	Task's Data Overlapping
r_k	Probability of offloading

Chapter 3

Node and Relevant Data Selection in Distributed Predictive Analytics: A Query-centric Approach

3.1 Introduction

3.1.1 Regulatory Constraints and Privacy in Distributed Data Systems

In recent years, the landscape of data accessibility has been significantly reshaped by the emergence of stricter privacy regulations and sector-specific governance frameworks. Legislation such as the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA) in the United States has introduced substantial constraints on how data can be collected, stored, and processed—particularly in sectors such as healthcare, finance, and telecommunications [52, 53]. For example, in healthcare, the Health Insurance Portability and Accountability Act (HIPAA) restricts the sharing of patient data across institutions without explicit consent, making centralized data aggregation impractical[54]. Similarly, financial institutions must comply with data localization and anti-money laundering regulations, which prohibit cross-border data transfers. Although these laws protect privacy, they create significant barriers to collaborative model training, often resulting in models with reduced generalization and robustness. To overcome these limitations, it is essential to design machine learning methods that preserve privacy while enabling the effective use of decentralized data. In response, this chapter proposes a query-centric approach to DPA. The method selectively engages only the most relevant nodes for each query, based on their local statistical evaluations, without requiring raw data sharing. This adaptive and privacy-aware mechanism ensures that sensitive data remains local, in compliance with data protection regulations, while still enabling high-quality predictive modeling. It also minimizes unnecessary communication and exposure, reducing the risk of privacy breaches. Importantly, this approach is compatible with a wide range of decentralized

learning paradigms—including FL [10], Incremental Learning, and other distributed learning frameworks—by focusing data access only on nodes most likely to contribute meaningfully to a given task.

3.1.2 Background

Distributed Machine Learning (DML) pushes model training, inference and prediction tasks to the network edge addressing concerns related to data privacy, centralized data transfer, and high communication costs [55]. Traditionally, *predictive analytics* relies on collecting vast amounts of data in Cloud infrastructure before training ML models for tasks like classification and regression [56]. A contrasting approach, *distributed predictive analytics* (DPA) [57], where code and models for training and inference are distributed to places where data are collected, has been emerged due to (i) increased processing power and memory capacity available in e.g., distributed micro-/edge-servers and road-side units at the network edge, and (ii) increased demands for improved data privacy.

DPA rely on a federated learning based infrastructure that enables many disparate sources of data owned by many organisations to contribute to training and using large-scale DML models for inference by exchanging only models among distributed *nodes*. Models are trained using only data that are requested for specific DPA tasks, referred to as *queries*. These queries include tasks such as regression, classification, outlier and novelty detection, and missing value substitution.

DPA amalgamates distributed knowledge *without* requiring data sharing, thereby eliminating the dependency on centralized training, which necessitates data sharing [58]. However, DPA is still in the early stages of developing models with quality that can be compared to centrally trained models for analytics queries. DPA copes with heterogeneous nodes in terms of data distribution, data size, and access patterns. Such heterogeneity hinders the development of accurate models. Understanding nodes' data characteristics like distribution, outliers, and quality, along with query access patterns across nodes, is essential. Such holistic knowledge enables the tailored selection of a subset of nodes for *each* DPA query. Such nodes should be involved in (i) predicting the relevant distributed data for each query and (ii) learning ML models for each query in a distributed fashion. However, this approach is neither directly applicable nor trivial in DML environments due to concerns about violating nodes' data privacy as elaborated in [59].

The straightforward approach of selecting random nodes to engage in model training per query is not sufficient and effective especially when dealing with heterogeneity in query access patterns. Fundamentally, random node selection neither considers the aforementioned data characteristics nor the query requirements. A federated infrastructure for DPA based on random selection of nodes' data has a significant impact on massively distributed ML models [60]. With the growth in real-time analytics tasks and the variability of distributed data, a comprehensive DPA task scheduling policy is required to ensure timely and efficient model training.

Ignoring this knowledge increases the likelihood of selecting arbitrary nodes with irrelevant

data compared to *what* the model actually needs for training, as evidenced by [57]. This inevitably leads to situations where nodes' local models negatively impact the final predictive performance due to potentially irrelevant training samples, as opposed to the data required by the queries. Consequently, aggregating irrelevant local models deteriorates the overall performance and quality of the model for each analytics query, as discussed by [61], [62], and [63].

3.1.3 Motivation & Challenges

In DML environments, nodes (e.g., edge servers and road-side units in smart cities) collect large-scale contextual data as shown in Figure 3.1. In practice, not *all* of a node's data are needed for a given query; however, some parts may be relevant and useful for improving model performance [64]. Our challenge is that these relevant data parts are usually distributed across nodes. Therefore, it is essential to leverage DPA to train a holistic model using *only the relevant data for each query* aiming to (i) improve prediction performance and (ii) avoid accessing irrelevant training data. Consequently, we must ensure that the model is trained exclusively with the required and relevant data *for each* query.

The objective of DPA on *query-centric node selection*, which significantly impacts the global model's quality, has not been thoroughly investigated. Most existing approaches use the random selection paradigm to engage nodes in distributed training, irrespective of the analytics queries, as in [65, 66]. Such approaches do not consider the specific requirements of queries that demand engaging only the most relevant and appropriate nodes. Our research question is: *How can we select only the relevant data from each chosen node per query given limited access to the data?*

Two fundamental strategies are to be established in node and data relevant selection challenge: **(S1)** Engage only those nodes in DML model training hosting the most appropriate data for each query *without* transferring data; **(S2)** Migrate data (or part of them) to nodes or the Cloud so DPA queries can be centrally processed [67], [68]. However, S2 strategy is not applicable in edge computing environments due violation of nodes' data privacy and data transfer. Whereas, S1 strategy seems promising in DML environments, nonetheless, the key challenge still lies in selecting *which* data to be selected thus avoiding accessing irrelevant data for DPA query processing. This *data selectivity* challenge requires a DPA paradigm to *learn* and *predict* the relevant data that are expected to be accessed and processed in each query.

We tackle this challenge by learning the relevant data across the data features exploiting the access patterns of queries issued by DPA applications (see Figure 3.1). Learning of relevant data is achieved by extracting dependencies between queries and local datasets, while rejecting data that are irrelevant to model learning.

Example: Consider several smart city DPA applications for real-time monitoring of on-street parking, surface parking lots in shopping malls, train stations, and corporate campuses in a city as shown in Figure 3.1. Such applications initiate DPA queries across nodes installed in e.g., parking areas and road-side units for e.g., (i) ranking the places with the highest parking occupancy in the

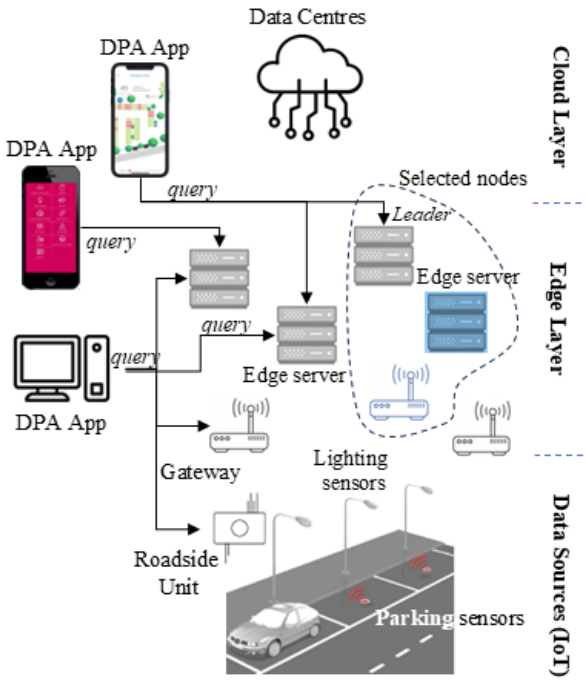


Figure 3.1: Node & relevant data selection based on a query-centric paradigm. Nodes, e.g., edge servers, roadside units, receive DPA queries from DPA Apps to train ML models for predictive analytics. Generated data by Internet of Things (IoT) devices, e.g., sensors, are *selectively* accessed from only selected nodes (encircled in a dashed line) avoiding accessing irrelevant data for each DPA query.

last 30 minutes, (ii) local (re)training of time-series forecasting models to predict the expected parking capacity (*per* parking area) in the next hour, (iii) updating city council recommender ML applications that inform drivers about available parking spaces in real-time (way-finding & interactive parking maps), (iv) updating local predictive maintenance ML models with the status of the surface sensors installed in each parking area, and (v) train traffic ML models for updating parking duration pricing based on city drivers' parking habits and daily schedules. The nodes deployed across diverse areas in the smart city should be able to predict which data they need to access for DML model training for each query improving e.g., urban decision making and planning. Awareness of *relevant* data emerged as a necessary feature of nodes [69]. Nodes benefit if they are aware of the trends and query access patterns of incoming DPA queries over their data [70], [71].

Our query-centric paradigm is proposed to enhance data selection by balancing nodes' privacy concerns with the need for high-accuracy analytics and efficient data processing. In real-world analytics applications as above-mentioned, model performance depends not only on the data volume but also on its relevance to the training process. Often, only a small subset (such as 20 percent or less) of data in each node is relevant to a specific analytic task [69]. In our example in Figure 3.1, for instance, at time 11:00am, a DPA application generates a query requesting parking sensors' data in the last 30 minutes for those sensors which have been occupied from 30

to 60 minutes (vehicles detected parked from 30 minutes to 1 hour) and stage-of-charge ranges between 60% and 80%. These data of interest need e.g., to train predictive models to forecast parking slot occupancy in a smart city or for incrementally updating pre-trained forecasting ML models. In e.g., banking sector, as another example, an analytic task normally involves nested queries, such as determining the number of customers who have both a credit score between 700 and 800 and an annual income between $\$50,000$ and $\$100,000$, and who have made at least three mortgage payments in the last six months. Additionally, the task might require understanding the impact of these customers' loan repayment behaviors on the bank's overall risk profile. By measuring the overlap between the data each node hosts (e.g., parking sensor occupancy rate within a time window and sensor battery level, or credit scores and income levels) and the specific query requirements, one can efficiently retrieve the relevant data needed for training models that predict pollution levels in a smart city or default risk in light of optimizing urban planning or loan offerings, respectively. Such DPA queries are common across various sectors, including finance [71], weather forecasting [72], healthcare [73], sports analytics [74], manufacturing [75], transportation and logistics [76], energy [77], and marketing [78], among many other fields that rely heavily on quantitative and predictive analysis. Compared to other methods [59, 79, 80, 79, 80, 81, 82, 55, 59], our mechanism reduces the need to access 100% of the data in each node for relevant data selection and training purposes.

3.1.4 Contributions

In this work, we introduce DPA query-centric mechanisms for predicting the most appropriate subset of nodes involved in DPA *per* query. By leveraging on nodes' statistical signatures over their local data helps predict and rank superior nodes that accelerate the overall model accuracy per query while avoiding redundant access to irrelevant training data. The main technical contributions of our paper are:

- We introduce a novel proactive, query-centric node and relevant data selection mechanism for DPA. This approach focuses on identifying relevant data to be accessed in advance, as indicated by each DPA query. This, in turn, determines the suitability of a node to be engaged in the DML process.
- We propose query-centric DML algorithms that utilize relevant data and node ranking for each query. We further elaborate on the complexity and effectiveness of these algorithms.
- We provide a comprehensive performance evaluation and comparative assessment using real data and query workloads. Our evaluation showcases the efficiency of our mechanism across various DML mechanisms, compared against baseline approaches and the relevant methods [57], [55], [81],and [83] found in the literature.

The rest of the chapter is organized as follows. In Section 5.2, we shed light on the related work elaborating on node and node and relevant data selection. Section 3.3 provides extensive discussion of the theoretical aspect of our problem and elaborates on the problem formulation along with preliminaries and definitions. In Section 3.4, we introduce our node and relevant data selection mechanisms, while Section 3.5 elaborates on the proposed query-centric DML mechanisms. Section 3.6 reports on the comprehensive performance evaluation, comparative assessment, discusses on the limitations of our approach and future directions for enhancement. Section 3.7 concludes the article.

3.2 Related Work

It has been evidenced in [84], [85] and [86] that the performance of a distributed learned model is greatly influenced by the quality of the distribution of data across the nodes. Therefore, many studies have focused on improving the model's performance through node selection and/or data selection. In light of this, we provide an overview of the most relevant studies on node selection. Additionally, we show how relevant data have been selected within each chosen node, according to previous approaches.

3.2.1 Node Selection in DPA

Most approaches adopt random node selection. Consequently, averaging across randomly selected nodes often leads to insufficient performance due to the heterogeneity of the nodes [59]. The heterogeneity of the nodes fundamentally determines how each individual node's data contributes to distributed model training. [55] considered various factors for node selection, such as edge device computational capability, network bandwidth, and energy consumption. The decision mechanism in [82] is based on the importance of user data by inspecting the current loss of the model at each training round. Both aforementioned mechanisms perform node selection decisions *after* model training. This means that resources are consumed in advance, and then it is decided whether engaging a node is worth it. This is not applicable in our case. For a given query, we need to predict in advance the most suitable nodes to be engaged in DML training, thus avoiding unnecessary resource consumption.

[81] introduced a mechanism where the selection criteria are based on choosing nodes with data different from what the models have previously learned. However, this approach also requires first training the model and then choosing the nodes. Additionally, it runs the risk of training a model on irrelevant data, which can be detrimental to the final model's performance. [80] proposed using a data quality score, computation score, and communication score as criteria to quantify the capabilities of the selected nodes. Meanwhile, [79] introduced a reward selection function based on the remaining energy budget (for mobile edge nodes), expected computation

load, and communication status. In both mechanisms, a model must be trained in advance to assess data quality and the computational load used.

[59] proposed the Federated Trace mechanism to track model training and data distribution across a random set of nodes. Similarly, [87] introduced a mechanism for nodes selection mechanism based on measuring each node's contribution in the previous round of the model training, which is mainly quantified as the prediction accuracy of the globally (and currently) trained model before and after aggregating it with each node's local model. In these approaches, it is mandatory a (global) model to have been trained in advance helping to choose node selection for future engagement. Finally, [88] proposed a node selection approach based on a fairness mechanism. With such mechanism, each candidate node obtains the same chance to get involved during the training process. The fairness is quantified based on a pre-trained model (before node selection).

Node selection has been investigated in time-optimized sequential decision making and on-line learning systems. There is a category of online decision-making deriving from Reinforcement Learning (RL) known as the Multi-Armed Bandits (MAB) [89] tackling specifically this problem. There has been a rise in the utilisation of MAB algorithms powered by RL examining the crucial balance between exploration and exploitation in sequential decision-making [90], [91]. The objective of MAB is to determine the best possible *arm* (node or sub-set of nodes in our case) from a group of possible arms (nodes) containing previously rewards. This is accomplished by picking a single arm sequentially and then tracking the reward that is realized [92]. The contextual MAB problem expands the core MAB by incorporating reward functions that rely on the context [60], [83] like queries in our case. Such works cast the node selection problem as a MAB problem in light of identifying the most appropriate node for maximizing expected rewards. An additional expansion of the MAB, the combinatorial MAB problem permits choice of multiples groups of arms [93], [94]. Notably, the work in [83] studies the impact of utility functions in these types of problems including node selection. In addition, the approach in [95] adopts deep RL to choose appropriate distributed node locations while utilising deep Q-learning method to ensure load balancing. The work in [96] introduced deep RL to enable actual time and low-overhead computation offloading and allocation of resources. However, it is essential to point out that these studies do not take into account the possibility of reducing data access redundancy nor select a sub-set of nodes to be engaged in a specific query, which are not their principled objectives. In addition, regarding the choice of offloading task to edge server(s), our paradigm takes into consideration the decision in *which* node(s) to offload the model training task for each query, which relates to the query access pattern over the node's data. On the contrary, the above-mentioned methods do not take into consideration any query characteristics for learning task offloading decisions while some of them involving data migration and mandatory full data access; the former is not applicable in our case, while the latter is inefficient especially when involving irrelevant training data as elaborated in our comparative assessment in Section 3.6.

Finally, the approaches in [97], [98] and [99] address the problem of selecting nodes for tasks and/or data offloading based on the Optimal Stopping Theory. However, encrypted information trading, as opposed to open access to shared data, reduces confidentiality and privacy upon data offloading. Our paradigm diverges from simple task offloading by focusing on selecting a sub-set of the most suitable nodes derived from the overlapping between any arbitrary DPA query and their data for training ML models. In our eco-system there is no data offloading while the model training tasks are offered by the selected nodes. All the decisions are tailored to the incoming DPA queries, which yields the novelty of our paradigm in DML environments.

In all the above-mentioned approaches, it is required, at least, one model training round *before* choosing the participant nodes for model learning. This evidently yields unnecessary time, irrelevant data access, and resource consumption, which in principle does not hold in our mechanism. Nonetheless, the associated node selection decisions in such approaches are not tailored to DPA queries, thus, focusing only on building generic models untied from specific tasks. Contrary to our mechanism, these approaches are unaware of the tasks requirements and needs of predictive applications. Furthermore, the node selection decisions do not take into consideration the relevance of the data residing on the nodes with the incoming DPA queries. That is, even if a node is selected, its data should also be 'scrutinised' in advance to reassure that these are relevant for being used in the DML process. On the other hand, our decision mechanism is purely driven by feeding the distributed learning with the most relevant data out of the most suitable nodes as it will be elaborated later.

3.2.2 Data Relevance in DPA

Since multiple nodes might be involved in the DML process, it is important to consider that not all data in the selected nodes may be relevant given a DPA query. It is crucial not only to select the most suitable nodes *but* also to identify the relevant data within these nodes. Including irrelevant data from nodes can negatively impact the performance of the derived predictive model. Excluding nodes whose some data are considered irrelevant might seem like a straightforward solution. However, such approach may not always be beneficial since it might lead to removal of valuable information that could have contributed to the predictive capacity of the models.

[100] proposed a mechanism to identify relevant data for model training based on a relatively small benchmark dataset. The benchmark-trained model evaluates the relevance of each individual data sample at every node to determine the relevant samples in a centralized location (e.g., a server). This mechanism requires sharing each sample's prediction errors, which can reveal specific information about each sample. [101] proposed a data relevance detection method using relevance scores for each sample, which account for inherent noise. In this approach, irrelevant (noisy) data are gradually excluded during the DML process. The data relevance is determined based on the noise of the training samples, regardless the specific DPA query. This differs significantly from our notion of data relevance. Both of the aforementioned mechanisms assess

each data sample individually based on the model predictive performance. They do not consider the relevance specific to the DPA query. In contrast, our mechanism identifies relevant data as a fundamental part of the selection decision for each query.

To the best of our knowledge, our approach is the first proactive query-centric node and relevant data selection mechanism yielding in building tailored DML models. Our work drastically departs from our recent preliminary study in [102] about the necessity and impact of identifying the relevant data in building DML models. This guided us to establish in this chapter (i) the fundamentals of the node selection problem per DPA query, (ii) identifying the minimum sufficient statistics required for selecting the most suitable nodes in advance to be engaged in the DML training phase, and (iii) introducing query-centric DML mechanisms that leverage the selected nodes for building models tailored to DPA queries.

3.3 Problem Fundamentals

3.3.1 Preliminaries & Definitions

DML aims to train a model over a distributed dataset $\mathcal{D}$ estimating the input-output mapping $y = f(\mathbf{x}; \theta)$, with d-dim. input $\mathbf{x} = [x_1, \ldots, x_d]^\top \in \mathcal{X} \subset \mathbb{R}^d$, output $y \in \mathcal{Y} \subset \mathbb{R}$, and θ being the model's parameters from a parameter space. Fundamentally, the difference with the centralized ML is that $\mathcal{D}$ is distributed over N nodes, denoted as $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$. Each node in $\mathcal{N}$ utilizes similar data features, such as weather data like humidity, temperature, and pressure, vehicle parking sensors data like occupancy duration, magnetic/inductive sensor type, air quality.

We assume a network of nodes represented via the graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ with $\mathcal{N}$ nodes (vertices) and edges $\mathcal{E}$ such that nodes n_i and n_j directly communicate (bilaterally) if there exist edges $e_{ij} = e_{ji} \in \mathcal{E}$ Each node $n_i \in \mathcal{N}$ has its own local dataset $\mathcal{D}_i = \{(\mathbf{x}, y)_\ell\}_{\ell=1}^{L_i}$, which consists of L_i samples. Each dataset $\mathcal{D}_i$ represents a subset of the entire dataset $\mathcal{D} = \cup_{i=1}^{N} \mathcal{D}_i$. When a node n_i receives a DPA task represented as a query $\mathbf{q}$, this node n_i acts as the 'leader' for that query $\mathbf{q}$. The leader node is responsible for establishing a mechanism to select the most suitable nodes $\mathcal{N}' \subset \mathcal{N}$ to execute the query $\mathbf{q}$ with the lowest possible global loss.

The loss function of the DML model f, denoted by $\mathcal{L}(f(\mathbf{x}; \theta), y) = \mathcal{L}(\hat{y}, y)$, measures the difference between the predicted output $\hat{y} = f(\mathbf{x}; \theta)$ and the actual output y given an input $\mathbf{x}$. This loss function is convex. For regression models, it can be $\mathcal{L}(\hat{y}, y) = |y - \hat{y}|$ or $(y - \hat{y})^2$. For binary classification models, it can be $\mathcal{L}(\hat{y}, y) = \max\{0, (1 - y\hat{y})\}$, with $y \in \{-1, +1\}$. Each node $n_i \in \mathcal{N}$ can have different data, which impacts the model f differently [87]. For example, some nodes $\mathcal{N}' \subset \mathcal{N}$ could improve the model f's performance, while others might negatively affect it.

We consider a discrete time domain $t \in \mathbb{T}$. At each instance $t = 1, 2, \ldots$, we are given a DPA query $\mathbf{q}_t$. A query $\mathbf{q}_t$ defines the boundaries $\mathcal{B}$ of the input subspace $\mathcal{X}$, such that the

samples contained therein are requested for training a DML model $f(\mathbf{x}; \theta(\mathbf{q}))$ given query $\mathbf{q}$. The parameters $\theta(\mathbf{q})$ of this model depend explicitly on the query $\mathbf{q}$, meaning this model should *ideally* be built from the distributed data:

$$\mathcal{D}(\mathbf{q}) = \{(\mathbf{x}, y) \in \mathcal{D} : \mathbf{x} \in \mathcal{B}\}. \tag{3.1}$$

The data space $\mathcal{B}$ is a d-dimensional space embedded in $\mathcal{X}$:

$$\mathcal{B} = \{\mathbf{x} \in \mathcal{X} : \wedge_{i=1}^{d} q_i^{\min} \leq x_i \leq q_i^{\max}\}, \tag{3.2}$$

where the query $\mathbf{q}$ specifies the min $q_i^{\min}$ and max $q_i^{\max}$ boundaries for each dimension in $\mathcal{X}$. In this chapter, we adopt a $2d$-dimensional range-based vectorial representation of a hyper-rectangle DPA query:

$$\mathbf{q} = [q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}]. \tag{3.3}$$

Note that this does not affect the analysis and generalization of our approach to define the distributed dataset $\mathcal{D}(\mathbf{q}) \equiv \bigcup_{i=1}^{N} \mathcal{D}_i(\mathbf{q})$. A query requested by a DPA application represented as a $2d$-dimensional vector results in selections of data samples across data dimensions. Range queries are integral part of real world DPA applications [103], e.g., calculating average pollution level across regions for city planning, selecting data from geo-spatial databases with GPS signals mining duration of individuals staying in locations, crime index indicators in city regions, e.g., neuroDB [104]. In this work, we deal with multidimensional data vectors representing multivariate contextual data for DPA applications (see our example in Section 3.1.2).

The lowest and highest boundary values $\{q_j^{\min}, q_j^{\max}\}_{j=1}^{d}$ are drawn from an unknown underlying joint probability distribution $P(q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}) = P(\mathbf{q})$. The geometric representation of $\mathcal{D}(\mathbf{q})$ is a hyper-rectangle defined by the lowest and highest boundary values linked to an interval range query. Other query types could be adopted to represent a query $\mathbf{q}$, which do not spoil our data retrieval from $\mathcal{D}$, e.g., k-nearest neighbors queries [105], [106], [107] or radius query [108], [103], [109]. The latter refers to as: $\mathcal{D}(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} : \|\mathbf{q}' - \mathbf{x}\|_p \leq \psi\}$ under norm $L_p = \|\mathbf{x} - \mathbf{x}'\|_p = (\sum_{i=1}^{d} |x_i - x_i'|^p)^{\frac{1}{p}}$. The geometrical representation of a radius query is a hyper-sphere in d-dimensional space defined by center $\mathbf{q}' = [q_i']_{i=1}^{d}$ with $q_i' = \frac{1}{2}(q_i^{\max} + q_i^{\min})$ and radius $\psi \in \mathbb{R}_+$. If $p = \infty$, the query represents a hyper-rectangle centered at $\mathbf{q}'$ with extent 2ψ on each dimension; for $p = 2$, the query region is a hyper-sphere.

We focus on query-based data selection for DPA represented by ranges over data dimension, e.g., ranges in date and time (temporal window of interest), city regions, vehicle' parking occupancy duration, vehicle detection indicators, and sensor types (e.g., magnetic, inductive). Query ranges can also include image features like texture, colour, and shape represented as d-dimensional intervals for image clustering and classification tasks, e.g., [110]. Selection over

other types of data like sub-graphs and moving images is beyond of the scope of this work and is left our future work.

Finally, it is worth noting that in a DML environment, the dataset $\mathcal{D}(\mathbf{q})$ is distributed across nodes and depends on the query $\mathbf{q}$. Given two different queries $\mathbf{q}_1$ and $\mathbf{q}_2$, the required datasets $\mathcal{D}(\mathbf{q}_1)$ and $\mathcal{D}(\mathbf{q}_2)$ for building the models $f(\mathbf{x}; \theta(\mathbf{q}_1))$ and $f(\mathbf{x}; \theta(\mathbf{q}_2))$, respectively, can differ. Additionally, the nodes containing the corresponding data for the two queries can also be different and dependent on the queries. We summarize our notations in Table 3.10 and abbreviations in Table 6.1.

3.3.2 Problem Formulation

Given a DPA query $\mathbf{q}$, we seek the subset of nodes $\mathcal{N}' \subset \mathcal{N}$ that can be participating in training a model f over their distributed data. However, nodes may have different data, where only the relevant ones are needed to be accessed for training f. Consider the following example, with a network of $n = 10$ nodes, each one having its own local datasets $\mathcal{D}_i$, and a query $\mathbf{q}$ that needs to access relevant data. We have three main cases for handling this query:

(C1) Centralized Case: One could gather all the data $\mathcal{D} \equiv \cup\{\mathcal{D}_i\}_{i=1}^{n}$ from all nodes in a central server and train a global model f_G over $\mathcal{D}$ for the query $\mathbf{q}$, as in [111]. We then obtain the average prediction error e_G corresponding to f_G. This means that we *do* have access to all participating nodes' data, which violates our principles for query-centric DPA in terms of data privacy.

(C2) Individual Case: Each node n_i can individually and locally train its local model f_i over local dataset $\mathcal{D}_i$. In this case, we obtain the (local) average error e_i for each node $n_i \in \mathcal{N}$ given the query $\mathbf{q}$. Evidently, in this case, no data are shared. On a first thought, one might claim that since the central server possesses complete knowledge, its predictions would be *better* (in terms of error) compared to those of each node. This is not always the case. It depends on the local data densities of each node, which are unknown in advance given a query. As it is evidenced in Figure 3.2, the performance of the global model f_G in terms of accuracy is better than some of the nodes' local models, i.e., from a sub-set of nodes $\mathcal{N}_0 \subset \mathcal{N}$, whose local errors $e_j > e_G$ for nodes $n_j \in \mathcal{N}_0$. On the other hand, there are some nodes, whose models perform better than the global one. These are the *most suitable* nodes $\mathcal{N}_1 \subset \mathcal{N}$ with local errors $e_k < e_G$, $n_k \in \mathcal{N}_1$. This means that there exists a subset of nodes whose models are more accurate than the global one. However, to identify this subset of nodes, one must first engage all the nodes *in advance* and train local models for each of them. Additionally, a global model would need to be built by transferring all the data and then selecting those nodes in $\mathcal{N}_1$. Evidently, this approach is neither practical nor applicable since we need to identify those nodes in advance given a query without training a global model.

Theorem 1. *Let e_G and e_i denote the prediction error of a central node/server and a local node*

$n_i \in \mathcal{N}$. It is not always the case that $e_G < e_i, \forall n_i \in \mathcal{N}$.

Proof. We prove Theorem 1 with two counterexamples. Consider the simple mean regression algorithm f^+, i.e., given an input $\mathbf{x}$, its predicted output $\hat{y}$ is the sample mean $\bar{y}$, and the k-nearest neighbors regression model f^*, i.e., the output is the average output of the k-th closest input points to the input $\mathbf{x}$. Consider that each $\mathcal{D}_i$ follows a Gaussian $\mathcal{N}(\mu_i, \sigma_i^2)$, with $\sigma_i^2 \to 0$ and $|\mu_i - \mu_j| >> 0, i \neq j$. Evidently, the central node's data set $\mathcal{D} = \cup_{i=1}^{n} \mathcal{D}_i$ follows the mixture $\mathcal{N}(\mu, \sigma^2)$ with $\mu = \sum_{i=1}^{n} a_i \mu_i$, $\sigma^2 = \sum_{i=1}^{n} a_i((\mu_i - \mu)^2 + \sigma_i^2)$, with $a_i > 0, \sum_{i=1}^{n} a_i = 1$. If we were told that all inputs followed $\mathcal{N}(\mu_j, \sigma_j^2)$ for some $j, 1 \leq j \leq n$ then we should have engage only node n_j, thus, yielding (i) $e_j < e_G$ in case of f^+ and (ii) $e_j = e_G$ in case of f^*, and avoiding engaging all nodes n_i or, even, constructing the global model in the central node by transferring all the data. Furthermore, consider that all $\mathcal{D}_i$ follow exactly the same distribution; consequently, the central node's data follows the same distribution. Then, regardless of any knowledge on the densities of inputs, we could randomly select one node from $\mathcal{N}$, thus, yielding $e_i = e_G, \forall n_i \in \mathcal{N}$, and avoiding constructing the central node and/or engaging all nodes. □ □

(C3) Model Aggregation Case: The query $\mathbf{q}$ can be sent to all nodes in parallel. Each node n_i locally builds a model f_i and sends it to a central node. The central node, after receiving all models, aggregates them and finally sends the aggregated model $f_A = \frac{1}{n} \sum_{n_i \in \mathcal{N}} f_i$ to all nodes, where each one stores it locally. The performance of f_A is expected to be higher than that of the global model f_G. Some of the nodes have similar or lower performance compared to the global model on the central node, while other nodes achieve much higher performance, as shown in Figure 3.2. Moreover, an aggregated model across only those nodes in $\mathcal{N}_1$ can potentially have higher performance than f_A (since models from nodes in $\mathcal{N}_0$ have been excluded), as shown in Figure 3.2 (black dash-dot line). Therefore, the models that belong to the nodes in $\mathcal{N}_1$ could minimize the expected error of the desired model, while the rest of the models do not contribute to improving the predictive performance. However, the nodes in $\mathcal{N}_1$ cannot be known in advance and are unlikely to be the same for each query.

For each incoming query, all nodes need to communicate to obtain an average estimate. This implies that all nodes are equally considered available for providing a model per query. Ideally, given a query $\mathbf{q}$, it would be preferable to identify the subset $\mathcal{N}' \subset \mathcal{N}$ of nodes whose average model's estimate $\hat{y}' = \frac{1}{|\mathcal{N}'|} \sum_{n_i \in \mathcal{N}'} \hat{y}_i$ would be (almost) the same as $\hat{y}$. This estimate $\hat{y}$ could be either the prediction of f_G or f_A. More interestingly, it would be ideal if we could select the *minimum* subset of nodes whose average estimate is as close to $\hat{y}$ as possible for each query. Evidently, we desire to obtain the *minimum* subset $\mathcal{N}' \equiv \mathcal{N}_1$ in advance *for each* query, without engaging all the nodes and/or transferring their data to a central location. This is a NP-hard problem as explained below.

Theorem 1. *Given a query $\mathbf{q}$, the problem of finding the minimum subset $\mathcal{N}' \subset \mathcal{N}$ of nodes, whose average estimate $\hat{y}'$ results to same error as $\hat{y}$ is NP-hard.*

Before proceeding with the proof of Theorem 1, let us recall the Subset Sum Problem (SSP): Consider a pair $(\mathcal{I}, s)$, where $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ is a set of $n > 0$ positive integers and s is a positive integer. SSP asks for a subset of $\mathcal{I}$ whose sum is as large as possible, but not larger than s. SSP is NP-complete. Consider now the following problem, referred to as Minimum Subset Average Problem (MSAP): Given $(\mathcal{I}, s)$, find the minimum subset $\mathcal{I}'$ with average s' such that $\lfloor s' \rfloor = s$ or $\lceil s' \rceil = s$.

Theorem 2. *The MSAP is NP-hard.*

Proof. If there is a polynomial-time algorithm for MSAP, then a polynomial-time algorithm can be developed for SSP. Suppose that there exists a polynomial algorithm, $A(\mathcal{I}, s)$ that solves MSAP, i.e., $A(\mathcal{I}, s)$ finds in polynomial time the minimum subset $\mathcal{I}'$ s.t. s. Then $A(\mathcal{I}, s)$ can be used to solve the SSP with $(\mathcal{I}, ns)$, $n = |\mathcal{I}|$. In general, any solution $B(\mathcal{I}, s)$ of SSP with $(\mathcal{I}, s)$ can be formulated as the Algorithm 1. If the polynomial-time complexity of $A(\mathcal{I}, s)$ is a polynomial $P(n)$, then the complexity of $B(\mathcal{I}, s)$ is $O(nP(n))$. But, this implies that there is a polynomial-time algorithm for SSP. Hence, no polynomial-time algorithm exists for MSAP.

□

Algorithm 1 $B(\mathcal{I}, s)$

$\mathcal{I}, s$ $\mathcal{I}'$

for $1 \leq k \leq |\mathcal{I}|$ **do call** $A(\mathcal{I}, \frac{s}{k})$ **If** a subset $\mathcal{I}'$ of $\mathcal{I}$ with k elements is found, whose elements have an average k' such that $\lfloor k' \rfloor = s/k$ or $\lceil k' \rceil = s/k$ **Then** obviously their sum is s, so exit and return $\mathcal{I}'$

Based on the outcome of the Theorem 2, we can now proceed with the proof of the Theorem 1.

Proof. Let $e = |\hat{y} - y|$ and $e' = |\hat{y}' - y|$. In order to show that the problem of finding the minimum subset $\mathcal{N}'$ with $e' = e$ is NP-hard, it suffices to show that the problem $|\hat{y}| = |\hat{y}'|$ is NP-hard. Consider the set $\mathcal{I}^R = \{|\hat{y}_i|_{i=1}^n\}$, $|\hat{y}_i| > 0, \forall i$. Since MSAP, which deals with set of positive integers, is NP-hard from Theorem 2 then a variant of MSAP with $(\mathcal{I}^R, \hat{y})$ is, also, NP-hard.

□

Nonetheless, if one is able to find the minimum set $\mathcal{N}'$ of nodes for a given query (let n be very small), then, this is meaningless. That is because, in order to obtain $\mathcal{N}'$ for a given query, once has *firstly* to query all nodes and, consequently based on their estimates, produce $\mathcal{N}'$. Hence, one has to *predict* in advance the most appropriate subset $\mathcal{N}'$, which results to same or less error than that of $\mathcal{N}$. Furthermore, out of those nodes in $\mathcal{N}'$, we should select *those* nodes whose datasets satisfy the query boundaries $\mathcal{B}$. Therefore, we propose a mechanism for proactive selection of nodes for each query, where these nodes will be engaged in training the model $f(\mathbf{x}; \theta(\mathbf{q}))$.

At any instance t, our mechanism predicts a subset $\mathcal{N}'_t \subset \mathcal{N}$ whose selected nodes provide a *good* model for query $\mathbf{q}_t$. Good model relates to (i) obtain a relatively small prediction error e_t as possible and (ii) achieve predictions $\hat{y}$ close to $\hat{y}_G$ (if the latter is able to be determined) denoting that $e_t = e_{G,t} + \varepsilon$ for some tolerance $\varepsilon > 0$ or, if possible, $e_t < e_{G,t}$.

Problem 1. *Given a query $\mathbf{q}_t$ at time instance t, predict whether a node $n_i \in \mathcal{N}$ is suitable for contributing to train the model f requested by the query, i.e., if n_i is a candidate to be added to $\mathcal{N}'_t$, based on minimum sufficient information node n_i has a-priori conveyed.*

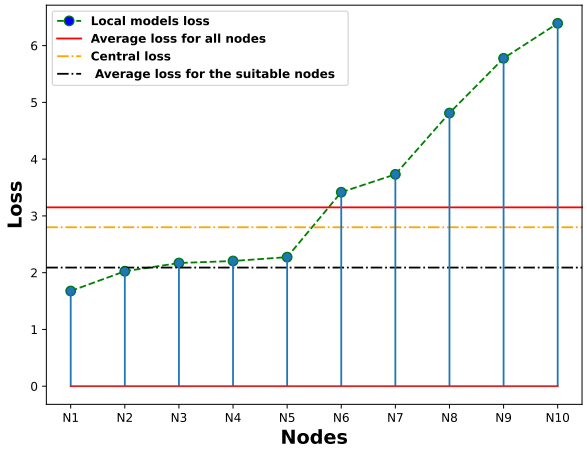


Figure 3.2: Local models' prediction errors (loss) over ten nodes compared with the centralized model, the average model across all nodes, and the average model across only the most suitable nodes.

In Problem 1, a suitable node selection decision should be achieved *before* model building given a DPA query. This entails predicting nodes' memberships to $\mathcal{N}'_t$ for each query at time t. This in turn, requires sufficient statistical information about the nodes' datasets without disclosing or accessing the actual data. Our mechanism should exploit such information to determine whether a node's data (or even parts of data) are *relevant* to query boundaries $\mathcal{B}$ in question. This enables a node to be included in $\mathcal{N}'_t$, thus participating in the DML training process of model f.

3.4 Node & Relevant Data Selection

3.4.1 Data Relevance Factors: Overview

We first introduce the factors the determine the notion of node's data relevance per query with a running example and then elaborate on the node selection mechanism based on ranking.

Consider a node n_i, which is randomly selected to participate in training model f given a query $\mathbf{q}$. For illustration purposes only, Figure 3.3 shows the node's data and the data region

defined by the query (area in a rounded rectangle) projected onto a 2-dimensional plane (e.g., the axes could refer to the first two principal components of the data feature space).

The data samples enclosed in the region defined by the query are the *relevant* samples w.r.t. query boundaries $\mathcal{B}$ for training the model f. Using the entire node's data results in incorporating *irrelevant* samples for the query. Consequently, training the model f with *all* data leads to a drift from the most accurate model [82]. Our challenge is to eliminate the impact of irrelevant samples with respect to the query on the model's training stage, especially when we do not have actual access to the node's data. The identification of a node's relevant and irrelevant data *per* query, which determines the training samples for building f, is not trivial. Our mechanism should identify only the relevant data *per* selected node *per* query by acquiring the minimum sufficient information from the data. Sufficient information can initially be conveyed by quantizing node's data. This can help separate relevant from irrelevant data per query.

We distinguish three factors determining the notion of data relevance w.r.t. query's boundaries over clustered data: **(F1)** The overlapping area between clusters' boundaries and the query's boundaries. **(F2)** The number of samples per cluster. **(F3)** The number of relevant samples per cluster. Each factor plays a specific role in assessing whether node n_i's data are relevant for model training given a query.

In our example in Figure 3.3, we assume that node n_i has $L_i = 6000$ samples clustered into six clusters $\{C_1, \ldots, C_6\}$ using, for example, k-means clustering algorithm. The samples are distributed per cluster as $\{1000, 800, 900, 1200, 1100, 1000\}$, respectively. As shown in Figure 3.3, the areas of clusters C_1, C_2, and C_3 have a higher overlap with the query's boundaries. These clusters, hereinafter referred to as *supportive clusters*, contain relatively many relevant samples w.r.t. the query. This contrasts with clusters C_4, C_5, and C_6, which have either very small or zero overlap with the query's boundaries.

Considering factor **F1**, we determine the membership of node n_i based on the degree of overlap per cluster, *without* actually accessing the data. It is sufficient to obtain the minimum and maximum values of each dimension per cluster and compare these against the query's boundaries.

Considering factor **F2**, the number of samples per cluster can further support the decision on selecting node n_i. This information can be easily obtained from the clustering process. A supportive cluster with a few samples and relatively high overlap with the query's boundary could be mistakenly considered more relevant for model building than a supportive cluster with more samples and medium overlapping rate. For instance, C_1 contains more samples than the other two supportive clusters, C_2 and C_3. Therefore, C_1 has the highest probability of providing more relevant samples for model building. This probability can be empirically calculated as the proportion of samples in each supportive cluster relative to the total number of samples across all supportive clusters. In our case, we obtain $(p_1, p_2, p_3) = (0.37, 0.29, 0.34)$ for C_1, C_2, and C_3, respectively.

By considering both factors **F1** and **F2**, we can order the supportive clusters based on their

probabilities, from the most supportive, C_1, to the moderately supportive, C_3, and finally to the least supportive, C_2. However, as shown in Figure 3.3, *all* the samples in C_2 are relevant w.r.t. the query. Therefore, C_2 should be ranked first, not C_1. This indicates that the information from factors **F1** and **F2** alone is insufficient for accurately determining node n_i's participation.

Consequently, we introduce factor **F3**, which focuses only on the samples from a supportive cluster that fall within the query's boundaries (rather than considering all samples from the cluster). A cluster's high density does not necessarily mean its samples are relevant to the query. **F3** aims to exclude irrelevant samples from supportive clusters. In our example, 25.5%, 87.5% and 22.2% of the samples in supportive clusters C_1, C_2, and C_3, respectively, fall within the query's boundary and are thus relevant. These percentages help revise the ranking probabilities based on the proportion of relevant samples. Consequently, the updated probabilities are $(p_1, p_2, p_3) = (0.22, 0.60, 0.18)$, indicating that C_2 is now the most supportive and relevant cluster, while C_3 is the least relevant. These factors lead us to a more accurate ranking of supportive clusters for node n_i, and hence a more precise node selection based on query-data relevance. Importantly, the determination of relevant samples relies on all factors work without disclosing nodes' data.

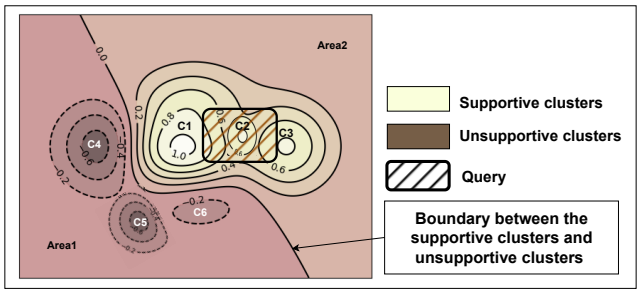


Figure 3.3: An illustration example of the impact of supportive clusters per query on predicting the relevant data on a node (data are projected onto a 2-dimensional space for illustration purposes only).

3.4.2 Data Relevance based on Factor F1

Consider a query **q** which defines the boundary $\mathcal{B}$ over the data, and a node n_i with a local dataset $\mathcal{D}_i$. The overlap between the query's boundary and node's data indicates an estimation of the percentage of data from the entire dataset $\mathcal{D}_i$ required for accessing and training the model f. This is associated with node n_i's contribution to model training.

As discussed in Section 3.4.1, F1 assesses the participation of a node and the corresponding relevance of its data from the supportive clusters. Specifically, node n_i has first quantized its input data space into K clusters $\cup_{k=1}^{K}\{C_k\} \equiv \mathcal{D}_i$, associated with d-dimensional cluster heads

$\{\mathbf{c}_k \in \mathcal{X} \subset \mathbb{R}^d\}_{k=1}^K$, which minimizes the quantization error:

$$\min_{\{\mathbf{c}_1,\ldots,\mathbf{c}_K\}} \sum_{k=1}^{K} \sum_{(\mathbf{x},y)\in\mathcal{D}_i} \|\mathbf{x} - \mathbf{c}_k\|^2. \tag{3.4}$$

Our goal in this step is to find the number of clusters K' out of K that have a high overlap with the query boundary across all dimensions. A high overlap with many clusters K' in node n_i indicates that this node has data patterns similar to those requested by the query, which is expected to increase the chance of that node being selected as a participant in the learning process.

Let us define the $2d$-dimensional vector:

$$\mathbf{c}'_k \;=\; [x_{1,k}^{\min}, x_{1,k}^{\max}, \ldots, x_{d,k}^{\min}, x_{d,k}^{\max}]^\top, \tag{3.5}$$

which contains the minimum and maximum values for each dimension of all the data inputs $\mathbf{x} \in C_k$ belonging to the cluster C_k, $k = 1, \ldots, K$. Specifically, $x_{j,k}^{\min} = \min\{x_{j,k} : \mathbf{x} \in C_k\}$ and $x_{j,k}^{\max} = \max\{x_{j,k} : \mathbf{x} \in C_k\}$, $j = 1, \ldots, d$. The vector $\mathbf{c}'_k$ represents the (hyper) rectangle boundary of the cluster C_k.

It might be the case that the underlying distribution of the data can change due to concept drift in, e.g., the input domain. This leads to potential changes in some of the cluster heads $\mathbf{c}_k$ of a node n_i, and in turn, updates the corresponding boundary vectors $\mathbf{c}'_k$. We then rely on the principles of incremental learning of the current cluster heads upon receiving a new batch of input vectors $\{\mathbf{x}^{(\tau)}, \mathbf{x}^{(\tau+1)}, \ldots\}$ by adopting the following update rule upon an incoming $\mathbf{x}^{(\tau)}$:

$$\mathbf{c}_k^{(\tau+1)} = \begin{cases} \mathbf{c}_k^{(\tau)} + \xi(\mathbf{x}^{(\tau)} - \mathbf{c}_k^{(\tau)}) & k = \arg\min_{\kappa} \|\mathbf{x}^{(\tau)} - \mathbf{c}_\kappa^{(\tau)}\|^2, \\ \mathbf{c}_k^{(\tau)} & \text{otherwise.} \end{cases} \tag{3.6}$$

This means, if node n_i receives a data $\mathbf{x}^{(\tau)}$ at time instance τ, then the closest cluster-head $\mathbf{c}_k^{(\tau)}$ is updated by update rate $\xi \in (0, 1)$ towards the new data $\mathbf{x}^{(\tau)}$.

Given an arrival of $\mathcal{T}$ new data points $\{\mathbf{x}^{(\tau)}, \mathbf{x}^{(\tau+1)}, \ldots, \mathbf{x}^{(\mathcal{T})}\}$, for those cluster-heads which have been updated using (3.6), the corresponding boundary vectors $\mathbf{c}'_k$ are updated based on the membership of the newly incoming data points to their cluster-heads. Given the query $\mathbf{q}$ and the vectorized k-th cluster boundary $\mathbf{c}'_k$, we calculate the percentage of overlap $h_k(\mathbf{q}) \in [0, 1]$ between these two hyper-rectangles (across all dimensions):

$$h_k(\mathbf{q}) = \frac{1}{d} \sum_{j=1}^{d} h_{j,k}(\mathbf{q}), \tag{3.7}$$

where

$$h_{j,k}(\mathbf{q}) = \begin{cases} \frac{q_j^{\max} - x_j^{\min}}{\max(x_j^{\max}, q_j^{\max}) - \min(x_j^{\min}, q_j^{\min})} & \text{if } q_j^{\max} \leq x_j^{\max}, \\ \frac{x_j^{\max} - q_j^{\min}}{\max(x_j^{\max}, q_j^{\max}) - \min(x_j^{\min}, q_j^{\min})} & \text{if } x_j^{\max} \leq q_j^{\max}, \\ 0 & \text{if } q_j^{\max} \leq x_j^{\min}, \\ & \text{or } x_j^{\max} \leq q_j^{\min}. \end{cases} \tag{3.8}$$

A cluster C_k is considered supportive w.r.t. query $\mathbf{q}$ if and only if $h_k(\mathbf{q}) \geq \epsilon$ based on a DPA application specific overlapping threshold $\epsilon > 0$. Hence, for node n_i, we obtain $K' \leq K$ supportive clusters:

$$C' \equiv \cup\{C_k : h_k(\mathbf{q}) \geq \epsilon\}. \tag{3.9}$$

The overall support of the supportive clusters towards the query $\mathbf{q}$ is then defined as:

$$r_i(\mathbf{q}) = \frac{1}{K'} \sum_{k=1}^{K'} h_k(\mathbf{q}). \tag{3.10}$$

So far, we argue that the samples belonging to the supportive clusters are considered relevant for the query $\mathbf{q}$, i.e., should node n_i be selected in $\mathcal{N}'$, then the samples in:

$$\mathcal{D}_i' = \{(\mathbf{x}, y) \in \mathcal{D}_i : \mathbf{x} \in \cup_{k=1}^{K'} C_k\}, \tag{3.11}$$

can be used for model training.

3.4.3 Data Relevance based on Factor F2

The overlapping degree $h_k(\mathbf{q})$ is not enough to decide whether node n_i is suitable to participate in model training, as discussed in is Section 3.4.1 in terms of F2. For instance, assume two nodes n_1 and n_2 both have $K_1 = K_2 = 3$ clusters, with node n_1 having $K_1' = 1$ supportive cluster and node n_2 having $K_2' = 2$ supportive clusters with respect to ϵ. We could rank the nodes based on their percentage of supportive clusters. For example, we obtain $\left(\frac{K_1'}{K_1}, \frac{K_2'}{K_2}\right) = \left(\frac{1}{3}, \frac{2}{3}\right)$ for n_1 and n_2, respectively. Hence, n_2 can be considered more suitable than n_1. However, the number of samples may differ across clusters. If the supportive cluster of node n_1 had more samples than the total number of samples in the two supportive clusters of node n_2, then n_1 should have been ranked first and not n_2. This will not happen unless the number of samples per supportive clusters has been taken into consideration.

The number of training samples is important to obtain a good model fit. Especially, in DPA models, by decreasing the sample size yields a model unable to capture the relationship between the input and output accurately, thus, leading to lower predictive performance as evidenced by

[112]. Let $|C_k|$, $k = 1, \ldots, K'$ be the number of samples of a supportive cluster $C_k \in C'$ of node n_i. Then, we extend (3.10) by adding the size of the supportive clusters:

$$r_i'(\mathbf{q}) = \sum_{k=1}^{K'} h_k(\mathbf{q}) a_k(\mathbf{q}), \text{ with } a_k(\mathbf{q}) = \frac{|C_k|}{\sum_{\kappa=1}^{K'} |C_\kappa|}, \tag{3.12}$$

such that $\sum_k a_k(\mathbf{q}) = 1$.

So far, given factors F1 and F2, we have obtained the overlapping degree $h_i(\mathbf{q})$ and the sizes of supportive clusters for node n_i with respect to a query $\mathbf{q}$. These indicators will be used to rank the suitability of n_i for model building, by using only the selected samples in the dataset $\mathcal{D}_i' \subseteq \mathcal{D}_i$, as derived in (3.11).

At that stage, the minimum sufficient statistic, which can be provided by each node, is the cluster boundary vectors $\{\mathbf{c}_k'\}$ associated with the clusters C_k. This can be shared among node n_i's neighborhood $\mathcal{N}_i$ in advance. Based on this, a node from a neighborhood, which receives a query $\mathbf{q}$, plays the role of a 'leader' to locally determine the suitability of each neighbor node without accessing the neighbors' data. Specifically, each neighboring node shares its cluster boundary vectors in advance, so there is no need for further communication with the leader node for each query. The leader node can efficiently compute the metric $h_j(\mathbf{q})$ for each neighboring node n_j upon receiving a query $\mathbf{q}$. This computation can be efficiently performed locally in $O(Kd)$ time.

3.4.4 Data Relevance based on Factor F3

The samples in $\mathcal{D}_i'$ for a node n_i are determined based on the boundaries of its supportive clusters, which have been identified w.r.t. the overlap between the query and cluster boundaries. However, according to factor F3 in Section 3.4.1, not all the samples in a supportive cluster can be considered relevant to the query boundaries. The fact that a supportive cluster overlaps with the query boundaries does not imply that all its samples are relevant. Relevance depends on the local density and distribution of samples within the supportive cluster itself.

To gain further insights into the relevance of the samples in supportive clusters, each node can include in its sufficient statistic the ratio of samples in supportive clusters that also falls within the query boundary. This means that each neighboring node must locally examine whether each input sample $\mathbf{x}$ from $(\mathbf{x}, y)$ in $\mathcal{D}_i'$ satisfies the query boundary $\mathcal{B}$ as well. In this case, when the leader node receives a query $\mathbf{q}$, it requests each neighbor to identify the subset of their dataset that is relevant to the query:

$$\mathcal{D}_i'' = \{(\mathbf{x}, y) \in \mathcal{D}_i' : \mathbf{x} \in \mathcal{B}\}. \tag{3.13}$$

That is, each node locally can estimate the size of relevant samples out of all the samples in a supportive cluster $|C_k'|$ such that $C_k' = \{(\mathbf{x}, y) \in C_k : \mathbf{x} \in \mathcal{B}\}$. Hence, the node can return to the

leader node the revised support information:

$$r_i^{''}(\mathbf{q}) = \sum_{k=1}^{K'} h_k(\mathbf{q}) b_k(\mathbf{q}), \text{ with } b_k(\mathbf{q}) = \frac{|C_k'|/|C_k|}{\sum_{\kappa=1}^{K'} |C_\kappa'|/|C_\kappa|}, \tag{3.14}$$

such that $\sum_k b_k(\mathbf{q}) = 1$.

In this case, the sufficient statistic requires $O(\sum_{k=1}^{K'} |C_k| d)$ time to be calculated on each member node for the query $\mathbf{q}$, which needs to be sent to the leader.

3.4.5 Ranking of Suitable Nodes

Based on factors F1 and F2, only the leader node locally estimates the metric $h_i(\mathbf{q})$ in (3.12) for each member n_i given a query. This requires that in advance each member has sent over the cluster vector boundaries to the leader once. This comes with $O(1)$ communication in the neighborhood of the leader and only local processing on the leader once, independent on the number of the incoming queries $\{\mathbf{q}_1, \ldots, \mathbf{q}_T\}$ up to horizon T. By incorporating factor F3, the leader node requires each member to estimate the metric in (3.14) locally per query. Then, each member sends this over to the leader. This requires $O(T)$ communication and local processing on each member.

Taking into account all factors, we rank the suitability of a node n_i for a query $\mathbf{q}$. Nodes are sorted by $\{r_1', \ldots, r_N'\}$ by the leader if only the support in (3.12) is used, resulting in no further communication with the leader. Alternatively, if the support in (3.14) is used, the leader, in collaboration with its neighbors, sorts the nodes by $\{r_1^{''}, \ldots, r_N^{''}\}$. In both cases, the leader determines a subset of the top-ℓ nodes ($\ell < N$) to be engaged in model learning given a query. The number ℓ of top-ranked nodes can be decided by selecting those nodes whose support r' (or $r^{''}$) satisfies a specific condition $\mathcal{A}(r')$:

$$\mathcal{N}' = \{n_i \in \mathcal{N} : \mathcal{A}(r_i') = \texttt{TRUE}\}, \tag{3.15}$$

with $\ell = |\mathcal{N}'|$. The condition depends on the DML policy adopted, which will be elaborated in Section 3.5. A node $n_i \in \mathcal{N}'$ participates in the distributed learning using its own *relevant* samples $\mathcal{D}_i^{''} \subseteq \mathcal{D}_i' \subset \mathcal{D}_i$, based on the adopted support metric $r_i^{''}$ or r_i'. Therefore, the relevant dataset $\mathcal{D}(\mathbf{q})$ for training the model f refers to the distributed relevant datasets of the selected nodes in $\mathcal{N}'$, i.e., $\mathcal{D}(\mathbf{q}) \equiv \cup_{n_i \in \mathcal{N}'} \mathcal{D}_i^{''}$.

The query-driven distributed learning minimization objective seeks the distributed model f over *only* the relevant data $D(\mathbf{q})$ across the *selected* nodes in $\mathcal{N}'$ that collaboratively minimize

the loss:

$$\begin{aligned} \mathcal{J}(f;\mathbf{q}) &= \frac{1}{|\mathcal{D}(\mathbf{q})|}\sum_{(\mathbf{x},y)\in\mathcal{D}(\mathbf{q})}\mathcal{L}(f(\mathbf{x};\theta(\mathbf{q})),y) \\ &= \frac{1}{|\mathcal{D}(\mathbf{q})|}\sum_{n_i\in\mathcal{N}'}\sum_{(\mathbf{x},y)\in\mathcal{D}_i''}\mathcal{L}(f(\mathbf{x};\theta(\mathbf{q})),y). \end{aligned} \tag{3.16}$$

Note that in our case, minimizing (3.16) specializes the generic distributed machine learning objective in [113]. Our aim is to minimize (3.16) among only the selected nodes in $\mathcal{N}'$ as expressed in Problem 1. Hence, in our case, the distributed model f is trained over the predicted relevant samples in $D(\mathbf{q})$ as requested by the DPA $\mathbf{q}$. In turn, the derived model f is tailored to the query $\mathbf{q}$ specifications and should not be confused with a derived model which would be trained over *all* nodes' data (relevant and irrelevant) regardless of any DPA query. In our case, each selected node $n_i \in \mathcal{N}'$ participating in the minimization of (3.16) contributes to the model training with only its (potential) relevant data in D_i'' and not with all its available data D_i. Section 3.6.3 elaborates on the impact of this data selection decision on the generalization capacity of the tailored model f and the corresponding implications.

In the next section, we provide the DML mechanisms to collaboratively train the model f tailored to a DPA query $\mathbf{q}$ across the selected nodes over their relevant data in a distributed fashion to minimize (3.16). Such DML mechanisms span across the spectrum of distributed learning (e.g., federated learning) as elaborated below.

3.5 Query-centric DML Mechanisms

Given the selected nodes $\mathcal{N}'$ for a query $\mathbf{q}$, we elaborate on the DML mechanisms based on the interactions among the selected participants. A node is elected as the leader based on specific criteria. In this work, a node becomes the leader if it receives a query $\mathbf{q}$. Consequently, the remaining nodes are considered members for this query. Note that any member node for a query $\mathbf{q}$ can be elected as a leader for a different query $\mathbf{q}'$ if it receives that query (see example in Section 3.1.2). The same holds true for an already leader, which can be appointed as a member w.r.t. another query received by another node.

Let us focus on a specific query $\mathbf{q}$ received to a leader node n_0. The leader node n_0 can initiate a (minimum) spanning tree to make aware all its members about its appointment for the received query $\mathbf{q}$.

3.5.1 Best Node Model Learning

According to node ranking, the leader n_0 selects only the top-ranked node n_i using the Best Node (BN) learning policy. This selection is made either directly by calculating the supports

$\{r'_j\}$ or by collaborating with the member nodes to calculate the supports $\{r''_j\}$, as discussed in Section 3.4. For simplicity, we refer to both variants of supports, r''_j and r'_j, unless otherwise specified. The selection condition $\mathcal{A}(r''_i)$ for the top-ranked node n_i is satisfied if and only if $r''_i = \max\{r''_j : n_j \in \mathcal{N}'\}$. In BN (see Figure 3.5(a)), it is assumed that node n_i has the most relevant sample among all candidate nodes. Therefore, the leader assigns node n_i to build a model $f_i(\mathbf{x}; \theta_i(\mathbf{q}))$ locally, using n_i's relevant data $\mathcal{D}''_i$.

The model f_i is sent to the leader node n_0 as requested, resulting in $O(1)$ communication with the BN policy. There might be a case where the top-ranked node has nearly all of the relevant data with respect to the query, i.e., $r''_i \rightarrow 1$. In this case, we achieve results similar to those in centralized training with respect to the query, where $\mathcal{D}''_i \simeq \mathcal{D}(\mathbf{q})$. This similarity is also reflected by the loss function of the BN, i.e., we obtain that:

$$\mathcal{J}(f; \mathbf{q}) = \mathcal{J}(f_i; \mathbf{q}) = \frac{1}{|\mathcal{D}''_i|} \sum_{(\mathbf{x},y)\in\mathcal{D}''_i} \mathcal{L}(f_i(\mathbf{x}; \theta_i(\mathbf{q})), y). \tag{3.17}$$

Further (incremental) training of f_i with additional candidate nodes (e.g., the 2nd or 3rd ranked nodes) could lead to overfitting, potentially worsening the model's performance. To avoid this problem, particularly when the top-ranked node has a relatively high support r''_i, we should be cautious about including *unnecessary* additional candidate nodes.

For illustration, Figure 3.4 shows the case of engaging more than just the top-ranked node with $r''_i = 0.9$ for training a deep neural network given a query. It demonstrates that there is no significant improvement in performance after incrementally training the model across the top-3 candidate nodes.

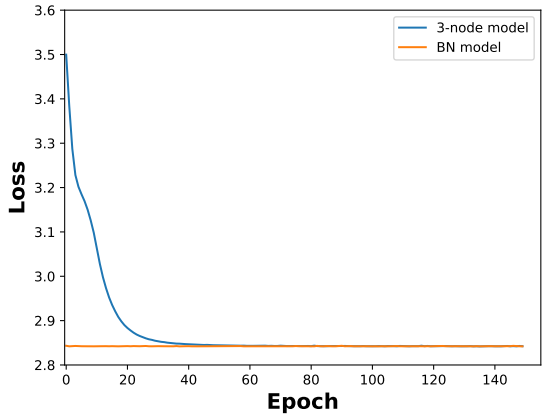


Figure 3.4: Comparison of BN model (top-1 node) vs. incremental model engaging the top-3 nodes.

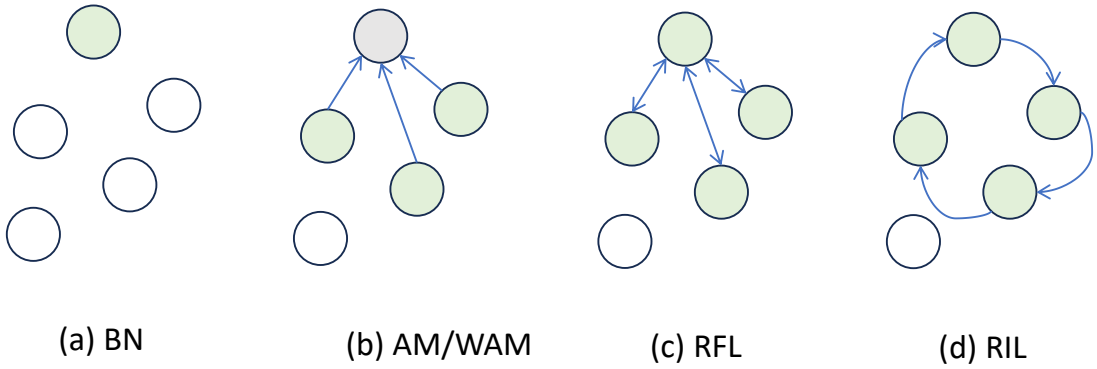


Figure 3.5: The spectrum of the query-centric DML mechanisms from (a) Best Bode (BN), to (b) Aggregation (AM/WAM), (c) Ranking-based Federated Learning (RFL), and (d) Ring-based Incremental Learning (RIL).

3.5.2 Aggregate/Weighted Aggregate Model Learning

Let us consider a case where more than one node has a relatively high support (ranking) value in the candidate set. In this case, more than one node can participate in DML by providing access to its relevant data. Given the supports $r_i^{''}$, the leader node n_0 selects nodes under a cut-off condition over $r_i^{''}$. This cut-off is a separating statistic between low and high ranking values, derived from the principle of outliers detection using the Median Absolute Deviation (MAD). MAD is defined as the median of the absolute deviations from the median $\tilde{r} = \text{median}\{r_i^{''}\}$, i.e., $\text{MAD} = \text{median}\{|r_i^{''} - \tilde{r}|\}$. MAD is the most robust measure of dispersion, separating relatively high support values under a cut-off, which is normally set to 2.5 times the MAD, as suggested in [114].

In the Aggregate Model (AM) learning, the selection condition $\mathcal{A}(ri^{''})$ for a node n_i to be selected is:

$$\mathcal{N}' \equiv \{n_i \in \mathcal{N} : \frac{|r_i^{''} - \tilde{r}|}{\text{MAD}} > 2.5\}. \tag{3.18}$$

The model f derives from aggregating the locally trained models f_i from the selected nodes $n_i \in \mathcal{N}^{''}$ determined in (3.18). The AM mechanism essentially takes advantage of the local data variety over the selected nodes (see Figure 3.5(b)). The leader n_0 communicates once with the selected nodes enabling them to locally build their models $\{f_i\}$ over their relevant data $\{\mathcal{D}_i^{''}\}$, individually. Then, the selected nodes returns their local models (parameters) $\{\theta_i\}$ to the leader. The latter aggregates the local models with equal importance:

$$f(\mathbf{x}; \theta(\mathbf{q})) \quad = \quad \frac{1}{|\mathcal{N}'|} \sum_{n_i \in \mathcal{N}'} f_i(\mathbf{x}; \theta_i(\mathbf{q})). \tag{3.19}$$

Note that we obtain $O(|\mathcal{N}'|)$ communication with the AM policy. However, selected nodes

could have different data distributions and ranges. Hence, some local models can perform better or worse than other, thus, it is not well argued to aggregate the local models equally. The model f is derived by a weighted aggregation of the local models taking into account the individual contribution of each models based on the relative ranking (support) given the query. We then obtain the Weighted Aggregate Model (WAM):

$$f(\mathbf{x};\theta(\mathbf{q})) \quad = \quad \sum_{n_i \in \mathcal{N}'} \hat{r}_i f_i(\mathbf{x};\theta_i(\mathbf{q})). \tag{3.20}$$

where $\hat{r}_i = \frac{r_i^{''}}{\sum_{n_k \in \mathcal{N}'} r_k^{''}} \in [0,1]$ and $\sum_{n_i \in \mathcal{N}'} r_i^{''} = 1$. The loss function of WAM is then:

$$\mathcal{J}(f;\mathbf{q}) = \sum_{n_i \in \mathcal{N}'} \frac{\hat{r}_i}{|\mathcal{D}_i^{''}|} \sum_{(\mathbf{x},y) \in \mathcal{D}_i^{''}} \mathcal{L}(f_i(\mathbf{x};\theta_i(\mathbf{q})), y). \tag{3.21}$$

We obtain the loss of AM by setting $\hat{r}_i = 1, \forall i$ in (3.21).

3.5.3 Ranking-based Federated Learning

Under the data relevance notion, each selected node contributes to minimizing the expected loss of the model f by adopting the principles of FL introduced in [115]. We extend the FL paradigm in our context by incorporating the weight of the relative ranking of the nodes selected in the AM/WAM mechanism, participating in a holistic (*federated*) model optimization as follows.

Firstly, the leader node n_0 selects the most suitable nodes based on (3.18). Then, the leader initiates a FL-based optimization process for $\mathcal{T}$ rounds across the selected nodes in $\mathcal{N}'$. We modify the sampling of the selected nodes at each round of the FL optimization by leveraging the relative ranking of the selected nodes. Specifically, at each round $\tau = 1, \ldots, \mathcal{T}$, the leader selects κ nodes out of the ℓ selected nodes in $\mathcal{N}'$ based on the ranking probabilities $(\hat{r}_1, \ldots, \hat{r}_\ell)$. Within round τ, each sampled node locally trains its model $f_i^{(\tau)}(\mathbf{x};\theta_i^{(\tau)}(\mathbf{q}))$ over its relevant samples $\mathcal{D}_i^{''}$.

At the end of the round τ, the model parameters $\{\theta_i^{(\tau)}(\mathbf{q})\}_{i=1}^{\kappa}$ of the κ sampled nodes are sent to the leader node. The leader aggregates the received models using the relative ranking as weights. This results in giving more attention to models trained with sampled from nodes with relatively more relevant data than the rest of the nodes (see Figure 3.5(c)). The overall objective of the Ranking-based FL (RFL) over the relevant data $\{\mathcal{D}_i^{''}\}$, $n_i \in \mathcal{N}'$ involving the ranking-based weighted aggregation by the leader is given by:

$$\min_{\theta(\mathbf{q})} \sum_{n_i \in \mathcal{N}'} \sum_{(\mathbf{x},y) \in \mathcal{D}_i^{''}} \frac{\hat{r}_i}{|\mathcal{D}_i^{''}|} \mathcal{L}(f(\mathbf{x};\theta(\mathbf{q})), y). \tag{3.22}$$

The federated model f obtained by (3.22) at the end of round $\mathcal{T}$ is used by leader node n_0 to

serve the query $\mathbf{q}$.

In RFL, we require $O(\mathcal{T}\ell)$ total communication rounds with the selected nodes per query. This derived from the principles of FL, which gradually optimizes the f's predictive performance w.r.t. our objective in (3.22).

WAM & RFL Loss Functions

In AM and WAM learning policies, a selected node n_i locally optimizes its local model parameter $\theta_i(\mathbf{q})$, $\forall i$, over the query-defined relevant samples such that:

$$\theta_i^{\star}(\mathbf{q}) = \arg\min_{\theta_i(\mathbf{q})} \sum_{(\mathbf{x},y)\in\mathcal{D}_i''} \mathcal{L}(f_i(\mathbf{x};\theta_i(\mathbf{q})), y). \tag{3.23}$$

Then, each optimized model is aggregated through (3.19) or (3.20) for AM or WAM, respectively, in the leader node n_0 in order to serve the incoming query $\mathbf{q}$. Hence, no extra communication among selected nodes is required. On the contrary, in RFL, all selected nodes try to optimize a FL model f, i.e., seek the optimal federated parameter $\theta(\mathbf{q})$ across all the relevant samples as provided in (3.22). The prediction capacity of each locally optimized model aggregated in the leader node per round is weighted based on the relative ranking of the participating nodes.

FL & RFL Loss Functions

The weighting model factor in (3.22) depends on the selected node ranking and the associated number of relevant samples within the node based on the query. This departs from the conventional FL paradigm by [115], where the weighting factor refers to all the samples residing on a node (including relevant and irrelevant samples). Moreover, the ranking value attempts to give more importance to the model parameters of those selected nodes with higher relevance to the query during the FL training process. Our proposed query-centric model averaging scheme in (3.22) is aligned with the averaging convergence scheme in [116], such that our averaging weights $\left\{\frac{\hat{r}_i}{|\mathcal{D}_i''|}\right\}$ are invariant in training time. Note that the ranking probabilities $(\hat{r}_1, \ldots, \hat{r}_\ell)$ relate to random sampling without replacement such that κ selected nodes are sampled out of the ℓ suitable nodes in $\mathcal{N}'$.

3.5.4 Ring-based Incremental Learning

In the Ring-based Incremental Learning (RIL) mechanism, given a query $\mathbf{q}$, the leader node n_0 selects the most suitable nodes as in (3.18) and then logically arranges these nodes to have a kind of dependency among them to incrementally train the model f over their datasets.

First, the nodes form a logical clockwise ring topology (initiated by the leader). The nodes are linked from the one with the relatively highest support to the node with the relatively least

support following the decreasing order $r_1^{''} \rightarrow r_2^{''} \rightarrow \cdots \rightarrow r_\ell^{''}$, $\ell = |\mathcal{N}'|$ (see Figure 3.5(d)). A node $n_i \in \mathcal{N}'$ receives the model update from the previous one n_{i-1} to further train and passes this model to the next node n_{i+1}.

In the incremental learning literature, the benefits of incremental learning are as follows. **(i)** By increasing the number of samples in model training, the accuracy and generalization ability of the model will be greatly improved. In our context, the model f is incrementally trained over the relevant samples from the suitable node in the ring. **(ii)** Based on the current model $f^{(m)}$ at the stage/node n_m, the performance of the model will be further optimized by new training samples from the next nodes $n_\kappa, \kappa > m$. **(iii)** After a stable model is obtained, the new training samples will not affect the performance of the model. This means that the predictions of new samples by two successive latest models $f^{(m)}$ and $f^{(m-1)}$ are almost the same.

Based on the above-mentioned benefits, the loss of the ring-based incremental learning of $f^{(m)}$ at node $n_m \in \mathcal{N}'$ is:

$$\begin{aligned} \mathcal{J}(f^{(m)}; \mathbf{q}) &= \sum_{(\mathbf{x},y)\in\mathcal{D}_{(m)}^{''}} \mathcal{L}(f^{(m)}(\mathbf{x}; \theta_{(m)}(\mathbf{q}), y)) \\ &+ \xi_{(m)} \sum_{(\mathbf{x},y)\in\mathcal{D}_{(m)}^{''}} \Big(f^{(m)}(\mathbf{x}; \theta_{(m)}(\mathbf{q}), y) \\ &- f^{(m-1)}(\mathbf{x}; \theta_{(m-1)}(\mathbf{q}), y) \Big)^2. \end{aligned} \tag{3.24}$$

$f^{(m-1)}$ denotes the model at stage $m-1$. ξ is a weighting factor representing the importance between the two predictions of two consecutive variants of the model during incremental learning. As the model is incrementally trained by accessing relevant samples from the suitable nodes, the factor $\xi_{(m)}$ increases. This indicates that the derived (final) model should be more generalizable over the relevant spaces of the models with higher support values.

Passes & Ring Structure in RIL

The model f can be (re)trained incrementally over more than one pass $\nu \geq 1$ of the ring. That is, the final model at the end of the first round, i.e., the model $f^{(\ell)}$ reaching the node with the highest support $r_1^{''} = \max\{r_i^{''}\}_{i=1}^{\ell}$, can undergo another pass of the ring.

We provide a process where the leader node n_0 can create the logical ring of the selected nodes in $\mathcal{N}'$. Based on the ranking list indexed by the selected nodes, i.e., $\Lambda = \{(r_1^{''}, n_1), \ldots, (r_\ell^{''}, n_\ell)\}$, the leader sends Λ to the top-ranked node n_1. Node n_1 is appointed as the *head* of the ring. It receives the list and is advised to connect with node n_2, sending the sub-list $\Lambda \setminus \{(r_1^{''}, n_1)\}$. Node n_2 then sends the updated sublist $\Lambda \setminus \{(r_2^{''}, n_2)\}$ to node n_3, and this process continues until node n_ℓ connects back to the head node n_1. When node n_1 receives a connection message from n_ℓ, the ring is complete and each node knows its *next* node. The incremental learning process then begins. At each stage, node n_m locally trains the model incrementally based on (3.24) and then

sends the updated model to the next node n_{m+1}.

The suitable node in $\mathcal{N}'$ requires $O(\ell\nu)$ communication for incremental training after ν passes. The leader node n_0 uses the incrementally trained model $f^{(\ell,\nu)}$ after $\nu \geq 1$ passes to serve the query $\mathbf{q}$.

Table 3.1 summarizes the computational and communication complexities of the node selection mechanisms and the DML mechanisms engaging the leader node (receiving a DPA query) and the involved ℓ selected nodes.

Table 3.1: Summary of Complexities

Node Selection		
Mechanism	**Computation**	**Communication**
Factor F1	$O(dK)$	N/A
Factor F2	$O(dK)$	$O(\ell)$
Factor F3	$O(d(K+\|C'\|))$	$O(\ell)$
DML Mechanism	Computation	Communication
BN	$O(1)$ (leader)	$O(1)$
A/WAM	$O(\ell)$ (leader)	$O(\ell)$
RFL	$O(\ell)$ (leader)	$O(\mathcal{T}\ell)$
RIL	$O(\ell)$ (leader)	$O(\nu\ell)$

3.6 Performance Evaluation

3.6.1 Experimental Setup

Datasets

We assess the performance and efficiency of our mechanisms using two publicly available datasets to create realistic environments. The dataset DS1[1] contains multivariate data with data heterogeneity among nodes. DS1 includes weather observations from $N = 13$ European weather stations, which correspond to the edge nodes in our context.

The dataset DS2[2] contains hourly air pollutant data from the Beijing Municipal Environmental Monitoring Center, covering $N = 12$ air-quality monitoring stations. These stations correspond to the edge nodes. The meteorological data for each air-quality station are matched with the nearest weather station from the China Meteorological Administration.

We chose DS1 and DS2 because they meet our study criteria, which include data collected from different locations. We further applied the data manipulation techniques [117], [118], [119] elaborated in on each node to achieve non-independently and identically distributed (non-i.i.d.) data over the nodes via distribution drifts where correlated features shuffled between samples

[1] https://github.com/florian-huber/weather_prediction_dataset

[2] https://www.kaggle.com/datasets/sid321axn/beijing-multisite-airquality-data-set

(the features contain both causes and effects of a target variable). Our target is to assess the impact on non-iid data and query access patterns in selecting nodes with relevant data for each DPA query. We evaluate this impact via the final DML model prediction accuracy, the node selection accuracy, and percentage of relevant data accesses as will be elaborated later. For both datasets, we conduct the same experiments following identical procedures to evaluate the best and worst-case scenarios for the proposed node selection mechanism and comparison mechanisms. In addition, in order to assess scalability of the node selection mechanisms in terms of number of nodes along with a fair comparison with the MAB-based mechanisms [120] and [83] dealing with a relatively large number of nodes for selection decisions (elaborated later), we adopted data augmentation techniques (including data blending via interpolation between samples) in [121] and [122] over DS1 and DS2. As a result, we obtained non-i.i.d. data shared among $N = 1000$ nodes. Note, the developed code and datasets are publicly available here[3].

Query workloads

We define $|Q_1| = 1100$ and $|Q_2| = 400$ DPA queries in our query workloads Q_1 and Q_2 for DS1 and DS2 dataset, respectively. Each query $\mathbf{q} \in Q$ is randomly generated across the entire data space, following the query workload method described in [123]. The process of query generation under query distribution $P(\mathbf{q})$ is introduced in [124, 123, 125], [126], and [109]. At time t, a DPA query $\mathbf{q}$ is uniformly distributed across the data space. For dimension x_j, the query *center* $q'_j = x_j^{\min} + (x_j^{\max} - x_j^{\min})\psi$ is uniformly sampled in $[x_j^{\min}, x_j^{\max}]$, with $\psi \sim U(0,1)$. Thus, the boundaries for this dimension are $q_j^{\min} = \max(x_j^{\min}, q'_j - \frac{\zeta_j}{2})$ and $q_j^{\max} = \min(x_j^{\max}, q'_j + \frac{\zeta_j}{2})$, with uniformly sampled query *length* $\zeta_j = \psi'\sigma_j$ with $\psi' \sim U(0,1)$ and σ_j being the standard deviation of the dimension x_j. Hence, the DPA generated queries cover the *entire* data space and avoids the exclusion of various sub-regions.

To provide a better understanding of the query representation, a typical DPA range query in a smart city urban analytics application requesting parking sensors' data could be: $\mathbf{q} =$ {[10:30am, 11:00am], [30 min., 60 min.],[60%, 80%]}, whose interpretation is elaborated on our example in Section 3.1.2. Such range queries are widely supported by modern database management systems as core part of aggregate queries (expressed in declarative languages like SQL) with range selection predicates [127]. Our DPA queries are queries with conjunctive selection predicates whose efficient computation has been a major research interest with methods applying sampling, synopses, and ML models to compute such queries; the interested reader could refer to [103] and there references therein for more information.

The queries are randomly assigned to nodes, with each node receiving a query with equal probability $\frac{1}{N}$. This random assignment facilitates the leader election process to identify leader nodes and their corresponding members for each issued query.

[3] `https://anonymous.4open.science/r/JNCA-752C/README.md`

ML models for DPA

Each node quantizes its data in advance using a vector quantization method into K clusters. In our experiments, we used the k-means clustering algorithm, modified with the update rules for cluster heads under data changes. To avoid bias, all nodes use the same number of clusters K. The data space quantization determines the supportive clusters for node ranking purposes. Other quantization algorithms that can dynamically update cluster heads in evolving data subspaces, such as those discussed in [128], may also be incorporated into our framework.

Our overarching goal is to train and examine the prediction accuracy of the models using only the *relevant data* predicted by incoming queries, rather than all available data. This helps in reducing the likelihood of training models on irrelevant data samples. We evaluate the predictive performance of these models compared to **(i)** the *ideal case*, i.e., accessing only the training data perfectly matched with the query boundaries in a centralized fashion, and **(ii)** the *ground truth*, i.e., accessing the predicted relevant data w.r.t. query boundaries in a distributed fashion.

To gain a deeper understanding of how our mechanism functions, we analyze its impact on the performance of ML and DL models for DPA queries. The rationale behind this selection of models is to cover the spectrum of relatively simple *but* yet powerful predictive models widely used in exploratory analytics [103], query-driven big data analytics [108], [123], decentralized data mining analytics [109], and aggregate query-based predictive analytics [124], [125] over distributed contextual data. Specifically, we conducted experiments with three different predictive models derived from the above-mentioned applications of analytics families: Multivariate Linear Regression (LR), Non-linear/Polynomial Regression (PR), and Deep Neural Network (DNN) models. In addition to the wide adoption of these families of models for predictive analytics, their choice is justified by several key factors: **(i)** LR is computationally efficient and ideal for large datasets, making it a solid choice for initial data exploration and benchmarking [123], [103]. **(ii)** PR provides the flexibility to capture more complex relationships in data selected by range and aggregate queries in large-scale databases without a significant increase in computational cost [124], [103]. **(iii)** DNN models are well-suited for modeling highly complex and non-linear query access patterns for mining data regions in large-scale datasets [109], [129] along with incremental regularization of compressed DNN models in data mining tasks [130]. Finally, LR and PR are straightforward to interpret in exploratory analytics [103] aiding in the clear understanding the relevant data learnt by query access patterns along with developing re-usable ML models for DML training efficiency in edge computing environments, e.g., [131]. The use of these diverse models, ranging from simple to complex, ensures a comprehensive evaluation of our mechanism across different applications and domains in the realm of DPA.

Model Configuration, Train & Test datasets

The tuned hyperparameters of these predictive models are provided in Table 3.3 and are assessed using the Mean Squared Error (MSE). For fairness, the prediction error is evaluated using *ground*

truth testing data for each query, which are *not* involved in any training process. Specifically, given a query $\mathbf{q}$, we ideally aim to access the (distributed) data $\mathcal{D}(\mathbf{q})$ defined in (3.1). These are the *actually* relevant data on which the model f should be trained according to the query specification. A sample of these data is used as the *test* dataset for evaluating the MSE of each DML model. Moreover, we train a DML model using our proposed mechanisms, which involves access *only* to the predicted relevant data from the selected nodes, i.e., data samples that have been identified as relevant by our mechanism to train our models. We then test the final DML model with the ground truth data for each query in question as explained above.

Consequently, given a query $\mathbf{q}$, we *train* a DML model f over the *predicted* relevant distributed data $\cup_{i=1}^{\ell}\{D_i^{''}(\mathbf{q})\}$ across all the ℓ selected nodes based on our DML mechanisms and, we *test* the model f's MSE over the actually relevant data $\mathcal{D}(\mathbf{q})$ for that query, i.e.,

$$\text{MSE} = \frac{1}{|\mathcal{D}(\mathbf{q})|}\sum_{(\mathbf{x},y)}(y - f(\mathbf{x};\theta(\mathbf{q})))^2. \tag{3.25}$$

Furthermore, in order to report on the statistical significance of the performance of the methods compared with our mechanism we adopt the non-parametric statistical Friedman's test [132]. This is a robust test that handles non-normal distributions and outliers. Friedman's test, as an extension of the Wilcoxon signed-rank test [132], is widely used in regression analysis (a.k.a. two-way analysis of variance by ranks). Friedman's test has been carried out in order to identify the best and worst-performing methods, and statistically significant differences between pairs of models based on the predicted response values. By assigning ranks on prediction residuals in using Friedman's test, the average prediction performance by all the models is first computed for each test dataset. The differences between the performances of all the models and the average are then calculated, and are subsequently ranked. We examine the reported p-value compared with the statistical level of 0.05 to assess the significance in the prediction accuracy among the methods in comparison. In our experiments, we provide the test result value p compared against 0.05.

Tables 3.2, 3.3, and 3.4 list the experimental parameters, the hyper-parameters, and the perofmance metrics used in this paper.

Table 3.2: Experimental parameters

Parameters	**DS1**	**DS2**
Nodes N	(13, 1000)	(12,1000)
Dimensionality d	11	18
Query workload size $\|Q\|$	1100	400
Environment	Static, Dynamic (concept drift).	
Number of clusters K	(5, 10, 15)	
Overlapping threshold ϵ	(0.45, 0.55, 0.65, 0.75)	
Statistical level p**-value**	0.05	

Table 3.3: Predictive Model & DML Mechanisms Hyper-parameters

Model	**LR**	**PR**	**DNN**
Local RFL rounds $\mathcal{T}$	20	20	20
Validation split	0.2	0.2	0.2
Learning rate	0.03	0.001	0.001
Number of RIL passes ν	5	5	5
RIL weighting factor ξ	0.2	0.2	0.2
Activation function	ReLU	ReLU	ReLU
Optimizer	Adam	SGD	Adam
Loss	MSE	MSE	MSE

Table 3.4: Performance Metrics

Metric	**Range**
MSE	$\mathbb{R}_+$
Node selection accuracy α	[0,1]
Node selection frequency	$[1, \|Q\|]$
Percentage of relevant data accessed	[0,100]%
Percentage of node's data accessed	[0,100]%

3.6.2 Baselines & Mechanisms under Comparison

We evaluate our query-centric DML mechanisms: Best Node Model (BN), Aggregate Model (AM), Weighted Aggregate Model (WAM), Ranking-based Federated Learning Model (RFL), and Ring-based Incremental Learning Model (RIL) in producing accurate models for DPA. These mechanisms are assessed w.r.t. factors F1 and F2 (F1-2) and all factors (F1-3), as defined in equations (3.12) and (3.14), respectively, over datasets DS1 and DS2.

We compare the performance of our mechanisms with the following baselines and relevant methods found in the literature: (i) the **Global Model (GM)**, where all the data are transferred in a centralized location used for centralized model learning per query (i.e., the Cloud) in a query-driven fashion [57]. (ii) the **Random Model (RM)**, where ℓ nodes are randomly selected to be engaged in the DML process per query [55]. (iii) the **Game Theory selection (GT)** and (iv) the **Fair Selection (FS)** mechanisms introduced in [81]. In addition, we assess the scalability of our paradigm with comparing with the query-driven Reinforcement Learning and contextual Multi-armed Bandit (MAB) mechanisms for node selection: **Max Utility-based MAB (M-MAB)** [133] and **non-stochastic MAB with Transformer-based Experts and Feedback (TEF)** [83]. M-MAB and TEF refer to online learning paradigms which are capable of selecting the subset of the most suitable nodes for engagement in DPA based on the trajectory of incoming DPA queries (contexts) as elaborated later. We chose TEF and M-MAB for comparative assessment since these mechanisms excel in incremental learning from a relatively high number of queries comparable to the number of nodes (using $N = 1000$ nodes in our comparison evaluation).

In GT mechanism, each node n_i builds its own independent local model f_i *in advance* according to its local dataset $\mathcal{D}_i$. Moreover, in GT it is assumed that each node could exchange its model with all the other nodes in the network. When a node n_i receives the trained model f_0 from the leader node n_0, it tests the f_0 model's performance locally against its data $\mathcal{D}_i$ and returns the results to the leader. The leader then targets those nodes that obtained accuracy lower than a pre-defined threshold. The rationale behind this is that a node could have data with different patterns than the leader's data. Therefore, the leader node selects those nodes n_i that have local models with relatively low accuracy in order to engage them for obtaining a global model. The learning process iteratively involves all the selected nodes' data for training the model.

The FS mechanism is a modification of the GT mechanism. While FS follows a similar approach, it incorporates additional fairness. Rather than always involving the same set of nodes, the FS mechanism ensures rotation by selecting the less frequently selected nodes at every round. This rotation ensures that over time, all nodes, irrespective of their initial performance or data, have an equal opportunity to participate and influence the global model. In essence, both the GT and FS algorithms emphasize fairness by ensuring that all nodes, regardless of the statistical patterns of their data or their models' initial performances, play an active and equitable role in the DML process.

The incremental learning mechanisms TEF and M-MAB are based on predicting the currently best sub-set of nodes to be selected via a trajectory of incoming DPA queries $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_t$. When the leader node is receiving a DPA query $\mathbf{q}_t$ (which plays the role of a *context* in these mechanisms), then the node recommends a sub-set of the most suitable nodes. The node ranking values are treated as the rewards to TEF and M-MAB for estimating the probability distribution of nodes being included in $\mathcal{N}'$. At every round t, TEF and M-MAB update such probability distribution w.r.t. current rankings $\{r_k\}_{k=1}^{N}$.

TEF is based on the MAB framework, which models the trade-off between exploration and exploitation over a sequence of actions (sub-set of suitable nodes in our case). TEF considers the query $\mathbf{q}_t$ as the context *up to* t, i.e., the sequence $(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_t)$. Such context is used for identifying the sub-set $\mathcal{N}'$ of the most suitable nodes. Given this context, the TEF uses the top-rank r'' as the reward as explained in [83]. TEF then estimates the current distribution over the set of nodes in $\mathcal{N}'$, which is the basis for recommending the most suitable node by sampling from this distribution.

M-MAB deals with the *countably many arms* in the MAB framework, i.e., the cases where there are relatively many candidate sub-sets of suitable nodes available at every query. M-MAB introduces a selection strategy based on the maximum utility of the most suitable nodes in light of reducing the sizes of the sub-sets of suitable nodes to manageable sizes. This is achieved by selecting nodes with maximum utility with respect to the currently executing query. M-MAB adopts Zooming LinUCB (linear Upper Confidence Bound) [134] to combine the upper confidence bound technique with an adaptive refinement step reducing the size of $\mathcal{N}'$. For each

currently executing query, M-MAB selects suitable nodes using LinUCB [135], where the reward is a linear function of the query $\mathbf{q}$.

Once the nodes for each query are predicted then, we compare the performance of the ML models for DPA trained by the DML mechanisms BN, AM, WAM, RFL, and RIL.

3.6.3 Performance Metrics & Evaluation

Node Selection Accuracy

We assess the efficiency of the node selection mechanisms by comparing the set of the most appropriate nodes $\mathcal{N}'$, predicted by our approach, with the ground truth (ideal) nodes $\mathcal{N}^\star$ given a random query $\mathbf{q}$. In real-life scenarios, the ideal nodes $\mathcal{N}^\star$ are entirely unknown unless we access all nodes' data and train models in advance for each query; hence, they are referred to as ideal nodes. We introduce the metric *node selection accuracy*, defined as the ratio:

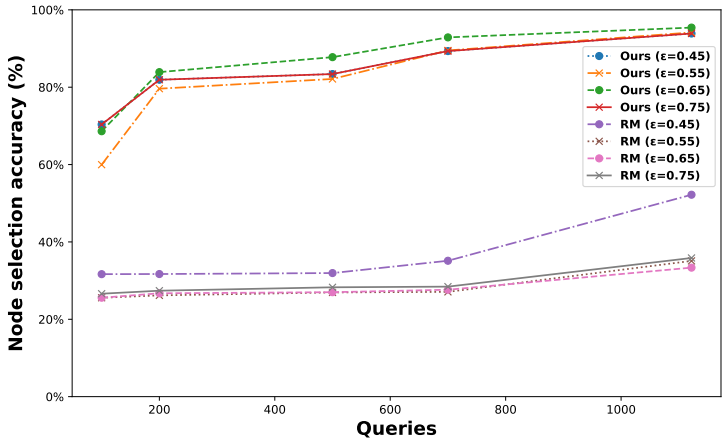
$$\alpha = \frac{|\mathcal{N}' \cap \mathcal{N}^\star|}{|\mathcal{N}^\star|} \in [0, 1]. \tag{3.26}$$

If the predicted selected nodes are included in the ideal nodes set, i.e., we accurately predict the most suitable ones for a query, then $\alpha \to 1$.

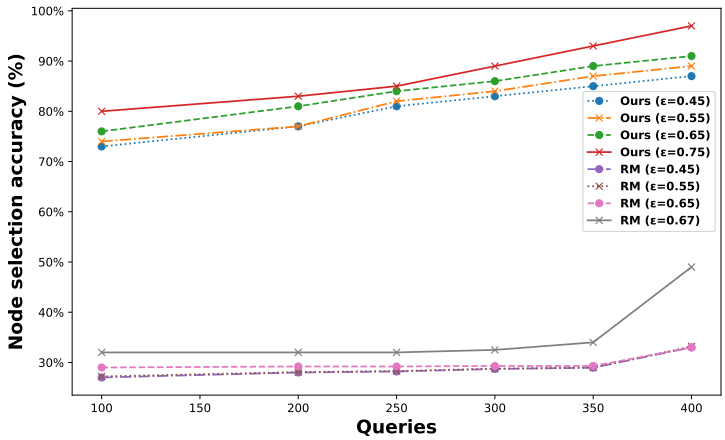
Additionally, we compare the node selection accuracy of our mechanism with that of a random node selection mechanism to highlight the importance of node selection and engagement per DPA query w.r.t. data relevance. In the random node selection, nodes are chosen entirely at random, meaning that the predicted set of nodes $\mathcal{N}'$ is a random subset of the nodes $\mathcal{N}$.

It is worth noting that our mechanism implicitly controls the number of participants through the overlapping threshold ϵ, which determines the supportive clusters. This slightly influences the node selection accuracy, as shown in Figures 3.6(a) and 3.6(b). Specifically, we present the metric α for our mechanism and the random selection with different overlapping thresholds $\epsilon \in \{0.45, 0.55, 0.65, 0.75\}$ over datasets DS1 and DS2. By assuming a relatively high ϵ (e.g., 0.75), we aim to select a few of the highest-ranking nodes using their supportive clusters (out of $K = 5$ clusters in the experiments shown in Figures 3.6(a) and 3.6(b)). Conversely, a relatively low ϵ value (e.g., 0.45) results in selecting almost all nodes with acceptable ranks.

Our selection mechanism achieves high selection accuracy ($0.7 \leq \alpha \leq 0.95$) across the range of ϵ values. In contrast, random selection yields a selection accuracy between 0.3 and 0.5 across the same range of overlapping thresholds. This clearly indicates that random selection of nodes per query is not a viable solution in our context, as outlined in our rationale. Similar results are observed for $K \in \{10, 15\}$ in both datasets.



((a)) Node selection accuracy α of our mechanism and the RM method with different overlapping threshold ϵ having $K = 5$ clusters per node; DS1 dataset.



((b)) Node selection accuracy α of our mechanism and the RM method with different overlapping threshold ϵ having $K = 5$ clusters per node; DS2 dataset.

Figure 3.6: Node selection accuracy α for DS2 and DS1 datasets.

Static & Dynamic Data Environments

We further demonstrate the effectiveness of our mechanism w.r.t. dynamic data updates. Specifically, we investigate how the node selection accuracy is affected by dynamic environments and assess the robustness of our proposed mechanism in these scenarios. We conduct experiments in both fixed and dynamic environments regarding data updates over the nodes.

In a fixed data environment, we assume that updates on the underlying data distribution over nodes occur slowly. That is, the underlying data distribution does not significantly change over time, or changes are negligible. In contrast, dynamic data environments, such as those involving time-series data from surveillance applications as described in [69], exhibit frequent data updates in the underlying distributions. In these cases, data updates occur rapidly, with potential concept drifts where mean values may shift and/or new patterns may emerge.

Static Data Environment: Consider the experiment conducted in a static environment with non-i.i.d. data over the nodes from datasets DS1 and DS2. In this scenario, the nodes' data distributions remain unchanged. Figures 3.7(a) and 3.7(b) illustrate the disparity between the *node selection frequency* with which a node is predicted to be a participant by our mechanism and that of the optimal (ideal node) selection. This selection frequency indicates the number of times a node has been selected for a DML process out of the total number of queries. The results indicate that the difference between our mechanism and the optimal selection is relatively small for each node across all the queries considered. This suggests that our mechanism performs closely to the ideal node selection, demonstrating its effectiveness in static data environments.

Dynamic Data Environment: We consider a dynamic environment, where the underlying non-i.i.d. nodes' data distributions. This case influences the node selection mechanism w.r.t. data clusters. For details on how the cluster heads and the associated cluster boundary vectors are updated in response to changes in the underlying data refer to Section 3.4. We performed four sequential changes (concept drifts) of the data distribution changing progressively 25% of the selected nodes' data distributions parameters (including mean and skewness values). For each change, we applied our node selection mechanism to predict the suitable set of nodes per query. We then compared their node selection frequencies with the frequencies of the nodes in the optimal case.

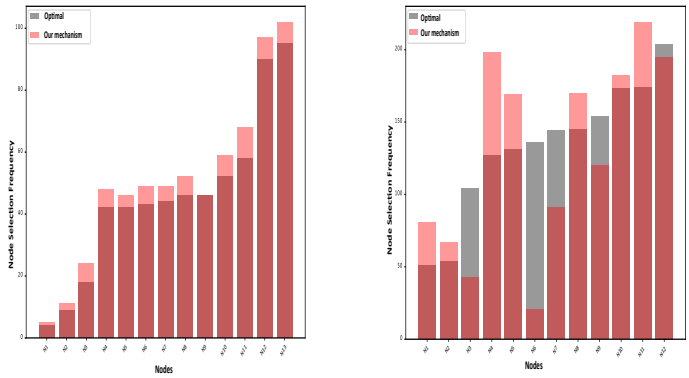
As we can observe in Figures 3.8(a) and 3.8(b), in each progressive change (from 25% to 100% total changes), our mechanism remains robust in selecting the most suitable nodes per query by gradually and incrementally updating the nodes' cluster-heads, and therefore, updating the cluster boundary vectors reflecting these data changes. We obtain similar results and behaviour of our mechanism over the datasets (similar results are obtained for $K \in \{10, 15\}$ in both datasets).

Relevant Data Access Rate

It is so far evidenced that our mechanism is capable of engaging the most suitable nodes to be involved in the DML process under static and dynamic data environments. We therefore examine the *amount* of relevant data accessed locally, which drives the efficiency of our mechanism.

After predicting the most suitable nodes, we examine the impact of the number of clusters K on the amount of relevant data used by the selected nodes during DML training. Our mechanism aims to reduce the amount of irrelevant data samples by increasing the number of clusters K in the nodes. An increase in K enhances the segregation between relevant and irrelevant samples. When K is relatively small, each cluster may contain a higher percentage of irrelevant samples. Conversely, increasing the number of clusters helps reduce the access to irrelevant samples within the supportive clusters.

Figures 3.9(a) and 3.9(b) illustrate how varying the number of clusters helps filter out irrelevant data samples accessed by the nodes over the DS1 and DS2 datasets, based on the three factors F1, F2, and F3. The vertical axis represents the percentage of data accessed across issued



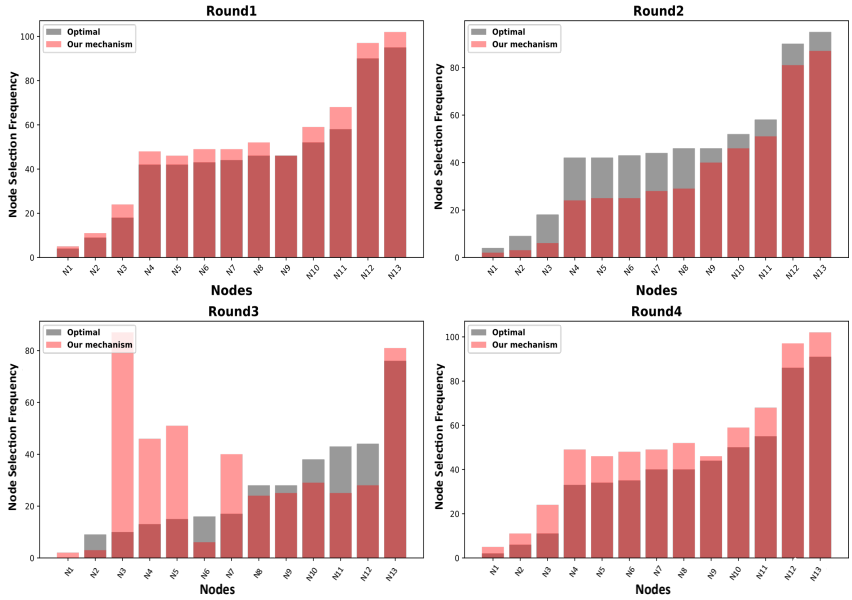
((a)) Static data environment: node selection frequency for our mechanism and the optimal (ideal node selection) one with $\epsilon = 0.65$ and $K = 5$ clusters per node; DS1 dataset.

((b)) Static data environment: node selection frequency for our mechanism and the optimal (ideal node selection) one having $\epsilon = 0.65$ and $K = 5$ clusters per node; DS2 dataset.

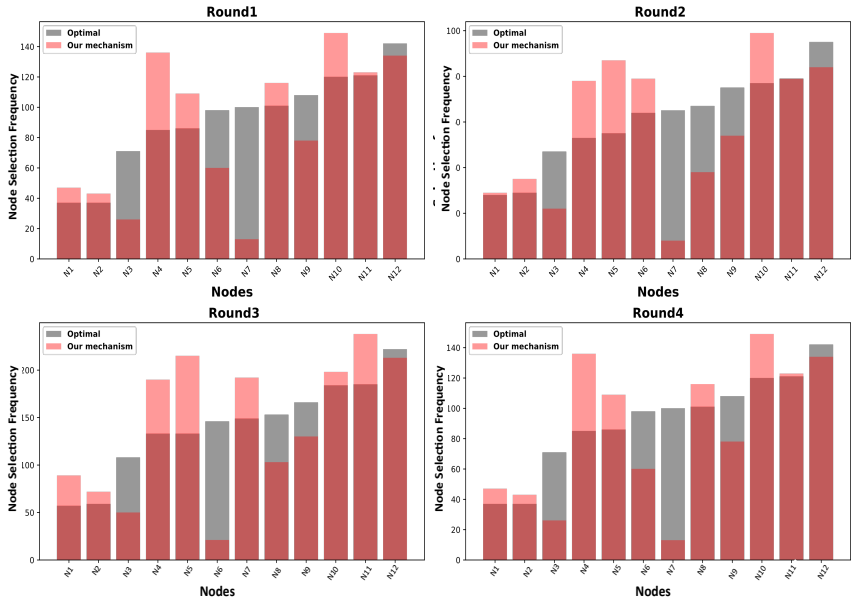
Figure 3.7: Node selection frequency for our mechanism and the optimal one in static data environments.

queries per node (on average). The bars indicate the percentage of data needed for each query from all selected nodes, using different K values per node. Evidently, for most queries, less than 20% to 40% of the total data are accessed, demonstrating the selective capability of our mechanism to involve *only* the required data per query. Accessing all available data typically deteriorates model performance, as will be shown later.

Table 3.5 summarizes the percentage of data accessed relative to number of clusters K for our mechanism, compared with benchmark methods across both DS1 and DS2 datasets in static and dynamic data environments. It is worth noting that when using the three factors F1, F2, and F3 (F1-3 in Table 3.5) for data selection per query per node, we achieve a high probability of accessing all relevant data per query from the supportive clusters (but not all the data in each supportive cluster). This effectiveness is particularly pronounced with higher K values. In contrast, the GT, FS, RM, and central GM mechanisms access all available data. Moreover, in Table 3.5 we show the percentage of data accessed by nodes selected using TEF, M-MAB and our method (using all factors F1-3 with $K = 15$ clusters) for $N = 1000$ nodes in both DS1 and DS2. Both TEF and M-MAB methods attempt to predict the most suitable nodes in light of increasing the probability of accessing relevant data over a relatively high number of nodes. However, our paradigm outperforms the MAB methods by being significantly selective in accessing all the relevant data thus avoiding redundant access of irrelevant data used to train the DML models. Our mechanism achieves 65% and 29% less redundant data access compared to TEF and M-MAB, respectively. This is attributed to the filtering mechanisms introduced in factors F1-F3 focusing explicitly on the characteristics of each incoming DPA query avoiding the summarization statistics over the queries' patterns trajectories as achieved by TEF and M-MAB.



((a)) Dynamic data environment: node selection frequency for our mechanism and the optimal (ideal node selection) one having $\epsilon = 0.65$ and $K = 5$ clusters per node over four sequential phases of data updates; DS1 dataset.

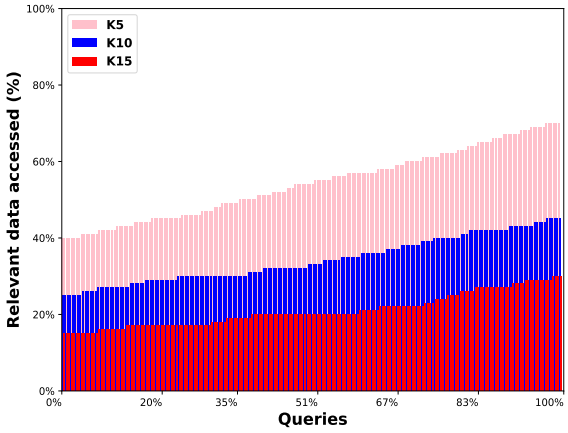


((b)) Dynamic data environment: node selection frequency for our mechanism and the optimal (ideal node selection) one having $\epsilon = 0.65$ and $K = 5$ clusters per node over four sequential phases of data updates; DS2 dataset.

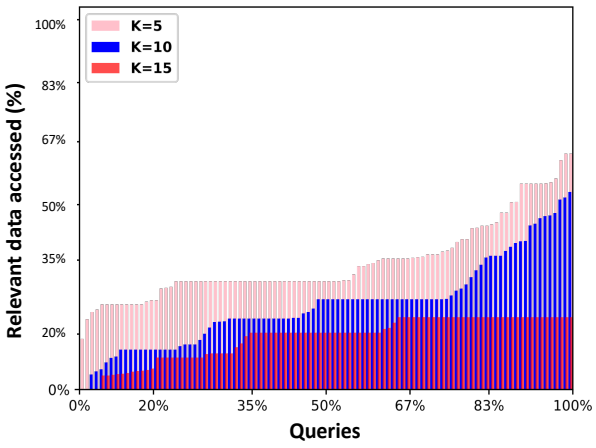
Figure 3.8: Node selection frequency for our mechanism and the optimal one in dynamic data environments.

Comparative Assessment

We compare the prediction error (MSE) over the test datasets in (3.25) of the models LR, PR, and DNN trained via the selected nodes by our mechanism against the methods GM, RM, GT



((a)) Percentage of relevant data accessed per proportion (%) of DPA queries in our mechanism adopting F1-3 factors; DS1 dataset.



((b)) Percentage of relevant data accessed per proportion (%) of DPA queries in our mechanism adopting F1-3 factors; DS2 dataset.

Figure 3.9: Percentage of relevant data accessed per proportion.

Table 3.5: Percentage of data accessed per query across node selection mechanisms in static and dynamic environments over DS1 and DS2.

	(DS1)		(DS2)	
Mechanism	**Static**	**Dynamic**	**Static**	**Dynamic**
GM	100%	100%	100%	100%
F1-3 K=5	39.88%	55.20%	69.13%	75.55%
F1-3 K=10	32.17%	52.67%	67.22%	64.90%
F1-3 K=15	20.92%	51.04%	65.27%	62.50%
RM	100%	100%	100%	100%
GT	100%	100%	100%	100%
FS	100%	100%	100%	100%
MAB Models	N = 1000		N = 1000	
	Static	Dynamic	Static	Dynamic
TEF	67%	79%	72%	79%
M-MAB	71%	78%	77%	81%
F1-3(K = 15)	19%	29%	55%	57%

and FS across all the query workloads Q_1 and Q_2 for the dataset DS1 and DS2, respectively. Moreover, we compare our mechanism adopting the three factors (F1-3) with the M-MAB and TEF models under N = 1000 nodes to study the scalability performance. Once the most suitable nodes have been predicted by each method, we apply the DML mechanisms BN, AM, WAM, RFL, and IRL over these nodes to train the predictive models.

Tables 3.6 and 3.8 show the performance of the predictive models w.r.t. MSE across all DPA

Table 3.6: MSE (loss) of predictive models based on all the node selection mechanisms over **DS1** across all the query-centric DML mechanisms (static data environment).

DS1 Static Environment						
Node Selection	**Learning Models**	**BN**	**RFL**	**RIL**	**AM**	**WAM**
Ours (F1)	**LR**	36.08	**32.44**	37.67	55.44	35.83
	PR	40.38	**32.71**	35.12	61.56	36.74
	DNN	38.05	**31.03**	33.56	56.98	34.60
Ours $K = 5$	**LR**	35.6	**31.81**	35.25	53.57	34.12
	PR	39.23	**31.83**	34.61	60.27	33.01
	DNN	35.71	31.96	**31.45**	56.71	33.17
Ours $K = 10$	**LR**	34.98	**30.74**	35.82	51.22	33.98
	PR	38.67	**30.15**	33.87	58.23	32.33
	DNN	33.56	**29.79**	30.48	54.29	30.11
Ours $K = 15$	**LR**	33.20	**30.87**	32.05	51.22	34.98
	PR	32.77	**31.77**	37.52	57.96	33.89
	DNN	31.43	**28.53**	30.68	55.75	31.51
GM	**LR (27.91)**	-	-	-	-	-
	PR(29.39)	-	-	-	-	-
	DNN (27.85)	-	-	-	-	-
GT	**LR**	**57.45**	85.43	88.78	97.59	89.10
	PR	**63.50**	72.57	73.33	98.04	91.61
	DNN	**46.01**	72.88	73.67	82.04	76.69
FS	**LR**	**60.49**	87.82	89.18	95.22	90.46
	PR	**63.88**	78.40	79.67	83.32	80.41
	DNN	**53.02**	72.48	74.25	78.39	74.95
RM	**LR**	**73.94**	98.82	97.39	112.87	98.98
	PR	73.53	85.50	**71.20**	93.61	87.26
	DNN	87.53	**77.59**	79.48	84.51	81.30
DS1 Static Environment; MAB Mechanisms ($N = 1000$)						
TEF	**LR**	53.44	48.42	47.49	**42.44**	48.66
	PR	43.43	45.15	**41.32**	53.51	57.44
	DNN	**47.55**	47.69	49.45	54.61	51.35
M-MAB	**LR**	63.54	68.62	67.69	**52.46**	58.76
	PR	**48.40**	57.52	50.52	49.15	58.84
	DNN	**49.95**	50.66	55.76	59.99	62.33
F1-3($K = 15$)	**LR**	28.04	28.28	**27.95**	28.70	29.98
	PR	30.23	20.20	**29.90**	33.71	32.25
	DNN	27.94	29.57	**27.88**	26.51	28.50

queries in static data environments over DS1 and DS2 datasets. Both tables show the comparison of the performance of the models having built based on nodes selected based on our mechanism in the cases: adopting F1 factor and F1-3 factors for different K values compared with the central GM, GT, FS, and RM approaches. Moreover, the TEF and M-MAB methods are compared against our F1-3 mechanism (with $K = 15$ clusters) selecting sub-sets over $N = 1000$ nodes. The selected nodes trained the different predictive models (LR, PR, and DNN) according to the query-centric DML mechanisms BN, RFL, RIL, AM, and WAM.

From the results, our mechanism consistently outperforms other node selection mechanisms even in the case with $N = 1000$ over the same queries ($p < .05$). Notably, in terms of the query-centric DML mechanisms, RFL demonstrates the closest approximation to the GM training outcome, while BN mostly shows good results for GT, FS and RM, however, with no comparable

Table 3.7: MSE (loss) of predictive models based on all the node selection mechanisms over **DS1** across all the query-centric DML mechanisms (dynamic data environment).

DS1 Static Environment						
Node Selection	**Learning Models**	**BN**	**RFL**	**RIL**	**AM**	**WAM**
Ours (F1)	**LR**	39.29	**33.46**	35.47	57.53	34.22
	PR	42.00	**34.76**	35.55	65.81	35.25
	DNN	44.72	**33.16**	34.51	49.35	33.26
Ours $K = 5$	**LR**	38.95	**32.78**	34.67	51.05	34.56
	PR	41.28	**33.23**	34.92	46.23	33.02
	DNN	43.54	33.40	**31.19**	47.29	32.36
Ours $K = 10$	**LR**	37.53	**32.04**	33.05	45.52	33.88
	PR	39.65	33.26	34.30	46.19	**32.72**
	DNN	37.71	32.02	**30.67**	45.56	30.04
Ours $K = 15$	**LR**	37.03	**30.43**	32.34	44.67	33.09
	PR	37.33	**31.46**	32.07	45.77	33.93
	DNN	36.02	31.52	**29.56**	45.68	31.22
GM	**LR (27.91)**	-	-	-	-	-
	PR(29.39)	-	-	-	-	-
	DNN (27.85)	-	-	-	-	-
GT	**LR**	**85.22**	96.15	101.27	111.18	96.94
	PR	**72.25**	79.27	83.81	86.92	82.53
	DNN	**71.18**	74.51	78.66	82.45	75.12
FS	**LR**	**88.61**	95.03	98.18	102.22	94.21
	PR	**61.05**	85.81	88.75	95.39	86.15
	DNN	**69.63**	73.75	76.68	88.81	74.62
RM	**LR**	**65.79**	128.21	132.79	130.31	93.27
	PR	**70.81**	93.67	96.24	97.43	95.21
	DNN	**68.63**	80.24	86.07	93.04	88.43
DS1 Dynamic Environment; MAB Mechanisms ($N = 1000$)						
TEF	**LR**	56.64	68.62	57.59	**52.54**	58.78
	PR	47.73	43.35	40.44	**40.41**	47.54
	DNN	**43.53**	57.59	59.55	64.66	61.85
M-MAB	**LR**	77.54	**70.72**	77.98	72.76	78.88
	PR	78.70	77.56	**70.77**	79.57	78.47
	DNN	69.65	70.76	75.16	**69.11**	72.66
F1-3($K = 15$)	**LR**	33.34	28.38	30.95	**28.09**	29.33
	PR	29.53	29.52	**29.45**	30.15	30.11
	DNN	28.57	**28.33**	29.28	30.15	33.22

accuracy results ($p < .05$). Specifically, the disparities between GM error and our results are on average 3.8% across all predictive models. This indicates that our mechanism achieves comparable accuracy under circumstances of not allowing data transfer to central locations including violation of data privacy apart from unconditional access to data. Moreover, given the unanimous agreement across all models regarding RFL's superiority, it is considered as the preferred choice for static data environments, especially when prioritizing model performance.

However, one must remain aware of potential communication overhead with RFL compared to the other query-centric DML schemes, e.g., BN and A/WAM. While RFL requires $\mathcal{T}$ training rounds among selected nodes, this results in more communication compared to BN, which simply engages a single node. RFL also surpasses the communication rounds in both AM and WAM, as they only demand one round of interaction between selected nodes. Moreover, RFL entails a higher communication compared to RIL, which the latter circulates a single model among the

Table 3.8: MSE (loss) of predictive models based on all the node selection mechanisms over **DS2** across all the query-centric DML mechanisms (static data environment).

DS1 Static Environment						
Node Selection	**Learning Models**	**BN**	**RFL**	**RIL**	**AM**	**WAM**
Ours (F1)	**LR**	42.53	**35.26**	40.84	61.35	52.49
	PR	43.77	**36.25**	41.41	70.49	53.04
	DNN	35.91	**32.15**	37.45	58.04	49.07
Ours $K = 5$	**LR**	40.15	**34.25**	39.31	56.88	48.82
	PR	42.00	**36.22**	43.77	60.04	47.17
	DNN	35.01	**31.75**	36.92	51.79	47.13
Ours $K = 10$	**LR**	39.20	**33.26**	38.42	55.03	47.43
	PR	42.53	**35.97**	42.21	62.18	48.00
	DNN	34.98	**31.09**	36.27	50.73	47.11
Ours $K = 15$	**LR**	38.02	**32.41**	37.42	53.38	46.24
	PR	41.88	**34.48**	41.12	60.90	47.66
	DNN	33.87	**30.78**	35.98	48.28	46.08
GM	**LR (30.57)**	-	-	-	-	-
	PR (32.34)	-	-	-	-	-
	DNN (30.29)	-	-	-	-	-
GT	**LR**	**65.45**	79.99	82.33	92.24	84.05
	PR	**67.23**	88.61	88.38	98.77	92.87
	DNN	**65.34**	76.65	77.23	95.07	82.88
FS	**LR**	**43.77**	62.26	68.20	90.12	88.81
	PR	**46.36**	67.07	71.91	94.27	86.33
	DNN	**42.78**	60.21	69.22	93.41	85.12
RM	**LR**	**51.17**	90.65	95.17	111.12	93.82
	PR	**53.34**	99.31	102.07	107.33	98.67
	DNN	**50.19**	87.00	91.44	97.29	94.15
DS2 Static Environment; MAB Mechanisms ($N = 1000$)						
TEF	**LR**	**55.66**	67.76	59.59	62.74	57.77
	PR	**49.91**	50.30	50.44	55.41	57.99
	DNN	47.77	**42.22**	52.22	60.01	62.27
M-MAB	**LR**	75.44	72.11	**70.01**	70.16	73.77
	PR	74.50	**67.66**	67.98	77.87	69.45
	DNN	66.02	70.11	**64.23**	71.11	69.96
F1-3($K = 15$)	**LR**	31.14	**30.81**	32.13	34.31	33.28
	PR	34.44	**33.01**	34.98	33.35	38.65
	DNN	31.44	**30.48**	31.77	32.55	36.34

selected nodes forming a logical ring. Notably, our mechanism F1-3 achieves better performance in the case of $N = 1000$ nodes by adopting the RIL method. This indicates the efficiency of our mechanism in selecting and engaging the most suitable nodes out of a network with a relatively high number of nodes. Furthermore, TEF and M-MAB methods produce similar predictive performance across different DML mechanisms (like BN, RIL, and AM) achieving, however, 39.02% more error (on average) compared with our F1-3 mechanism ($p < .05$).

In DS1, static data environment (Table 3.6), we obtain a trade-off between accuracy and communication overhead by adopting RFL (and RIL for $N = 1000$) against other less communication intensive query-centric DML mechanisms. We can slightly sacrifice some of the obtained accuracy, e.g., around 1% less, by adopting RIL or WAM instead ($p < .05$). Similar trade-off is obtained in DS2, static data environment, (Table 3.8). For $K = 15$, our selection mechanism

Table 3.9: MSE (loss) of predictive models based on all the node selection mechanisms over **DS1** across all the query-centric DML mechanisms (dynamic data environment).

DS1 Static Environment						
Node Selection	**Learning Models**	**BN**	**RFL**	**RIL**	**AM**	**WAM**
Ours (F1)	**LR**	47.40	**38.35**	42.11	68.28	39.49
	PR	48.47	**39.33**	39.45	70.49	40.59
	DNN	49.16	37.77	**37.13**	59.30	39.32
Ours $K = 5$	**LR**	46.60	**37.37**	41.59	63.78	38.46
	PR	47.27	38.81	**33.19**	65.29	39.62
	DNN	47.25	**36.99**	37.17	57.31	38.40
Ours $K = 10$	**LR**	46.25	**36.08**	37.22	62.10	37.37
	PR	46.06	37.25	**35.90**	63.57	38.29
	DNN	45.02	**35.05**	36.88	56.37	38.67
Ours $K = 15$	**LR**	46.55	**33.98**	35.18	62.77	37.03
	PR	45.20	36.82	33.71	62.32	**36.14**
	DNN	44.17	**32.39**	34.48	55.18	35.26
GM	**LR (30.57)**	-	-	-	-	-
	PR (32.34)	-	-	-	-	-
	DNN (30.29)	-	-	-	-	-
GT	**LR**	**64.18**	79.99	82.32	114.00	84.75
	PR	**65.24**	88.61	92.24	116.00	85.15
	DNN	**61.52**	74.78	76.23	113.37	81.09
FS	**LR**	**73.12**	76.44	82.69	107.38	84.63
	PR	**75.15**	77.83	77.19	109.32	86.91
	DNN	**73.13**	74.46	75.63	111.50	83.41
RM	**LR**	**71.00**	90.65	95.17	111.12	93.82
	PR	**53.34**	99.31	102.07	107.33	98.67
	DNN	**50.19**	87.00	91.44	97.29	94.15
DS2 Dynamic Environment; MAB Mechanisms ($N = 1000$)						
TEF	**LR**	65.55	**51.11**	60.11	61.43	67.89
	PR	44.43	**41.22**	49.33	45.33	59.01
	DNN	57.88	**38.33**	42.44	51.21	52.88
M-MAB	**LR**	65.64	62.61	**60.44**	73.36	74.01
	PR	64.60	57.56	**57.08**	57.89	59.94
	DNN	56.22	**61.12**	67.07	66.20	58.76
F1-3($K = 15$)	**LR**	33.21	**31.11**	32.23	33.91	32.55
	PR	33.97	**33.16**	34.29	34.39	35.25
	DNN	31.11	**30.44**	32.66	32.12	34.77

adopting RFL and RIL achieves almost similar accuracy prediction levels with the GM. This indicates the capability of our mechanism to offer distributed learning over DPA queries without accessing the data in advance nor transferring the data in a central location.

In dynamic environments, similar to the static ones, our mechanism surpasses other techniques, with RFL, RIL, and WAM emerging as best performers for most of the predictive models. Tables 3.7 and 3.9 present the performance of the node selection mechanisms in dynamic data environments, where data have undergone changes described in Section 3.6.3. It is worth noting that in this context, RIL is considered as an efficient query-centric scheme to improve the model performance and reduce the communication rounds. However, this requires the communication dependency among nodes during the ring-based model training. Furthermore, our mechanism does not only select the most suitable nodes, it also reduces redundant access to irrelevant data as

discussed in Section 3.4.1. Each query, on average, requires access only to 10% $\sim$ 15% of data on a selected node. In DS2, WAM seems also another efficient candidate in some cases compared to RFL and RIL, e.g., for PR and DNN models when $K \in \{10, 15\}$ trading off accuracy with communication overhead. Furthermore, in the case of $N = 1000$ nodes, RFL is proved efficient for TEF and our mechanism, while RIL is a promising candidate for M-MAB mechanism. Even in this case, our mechanism outperforms TEF and M-MAB in terms of prediction accuracy ($p < .05$) while RFL scheme achieves 26.55% and 41.01% less error (on average) compared to TEF and M-MAB, respectively.

Discussion on Generalization Capacity of ML Models for DPA

Our preemptive mechanism ensures that the subsequent training phase leverages the most relevant data out of all selected nodes' data. This yields each DML-trained model *tailored* to each DPA query. Such training process over relevant samples might affect the generalization capacity of the derived ML model for DPA, i.e., its ability to adapt to new, previously unseen data, yet drawn from the same distribution as the one used to train the model (the distribution of the relevant data). Our mechanism attempts to balance this potential lack of generalization by introducing the factors F1, F2, and F3 in Section 3.4. Recall that each factor controls the 'degree of relevance' of the data used for training with F2 and F3 attempting to predict the highest portion of relevant data given a query. As evidenced in our experimental results in Tables 3.6, 3.7, 3.8, and 3.9 (refer to performance of F1 and F1-3 variants), the derived models obtain almost similar MSE with the ideal GM method, which uses only the most relevant data for training given a query (ground truth). Evidently, by adopting our mechanism with all factors involved (i.e., F1-3 variant), the derived model's generalization capacity is decreased (due to overfitting).

On the other hand, our mechanism's variants with factor F1 result in not aggressively filtering our irrelevant training data given a query. In these variants, the derived model is trained over a *mixture* of (predicted) relevant samples (as strictly requested by a DPA query specifications) and irrelevant samples, which could not be excluded due to the F1 factor. In this case, we obtain the following trade-offs: a highly tailored model (based on the F1-3 variant satisfying the DPA query) with low model generalization capacity, and a less tailored model (based on the F1 factor) with higher generalization capacity. Depending on the DPA application and analytics domain, once could balance between a generalized model and a model that fits the requirements of the DPA query. For instance, models for exploratory analytics applications, e.g., [103] can be obtained using our mechanism with F1 and/or F2 variants to allow for generalization, while models for aggregate predictive analytics in large-scale databases, e.g., [125], [108] can be obtained using our F1-3 variant, where model fitting over large-scale datasets is significantly required (closely match with the relevant data allowing for accurate predictions and pattern mining).

Discussion on Comparative Methods Performance

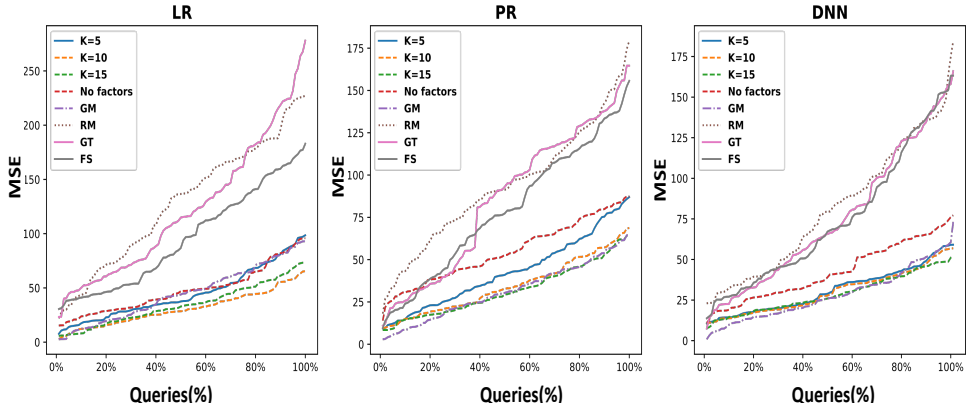
Considering now the approaches under comparison, i.e., GT, FS, RM, TEF and M-MAB, it is required to access all the data in advance to establish the most suitable nodes. As stated by [82], selection of nodes is based on a value function, which reflects the usefulness of the data during each training round. Moreover, the MAB methods in [133] and [83] require online training to achieve convergence in determining the most suitable sub-set of nodes per DPA query. During this training period (exploration-exploitation rounds), these methods require full access to the data to quantify the reward value per round.

Our mechanism is based a pre-training model principle, i.e., the nodes are predicted to be suitable for a DPA query before attempting any DML process. Whereas, GT, FS, and TEF and M-MAB methods are based on a post-training model principle, i.e., the nodes are predicted to be suitable for a DPA query after training a predictive model (for TEF and M-MAB this holds up to the conference stage). A commonality among these approaches is their dependency on building the predictive models first; then, based on these models, they determine the suitable set of nodes. Notably, even after the node selection process, they do not focus explicitly on the relevant data requested by the queries. In contrast, in our mechanism, we prioritize both the selection of suitable nodes and the identification of relevant data before initiating the training stage.

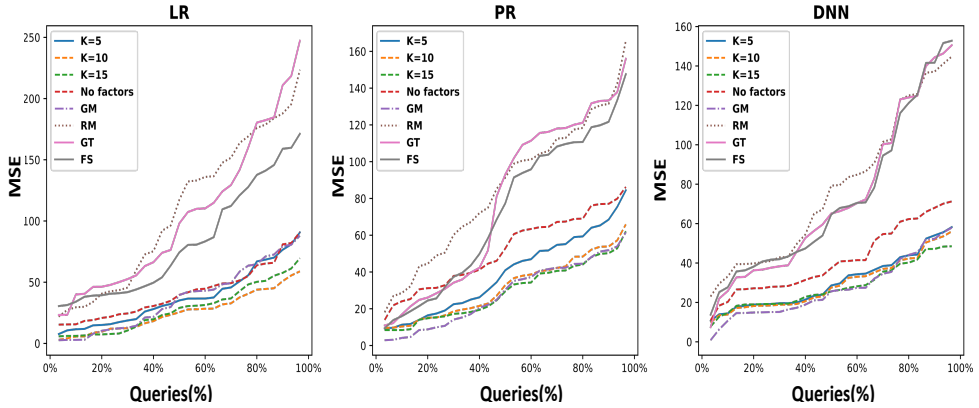
In terms of data access, as shown in Figure 3.9(b), the number of irrelevant samples has significantly reduced when considering the clustering approach compared to the comparison methods. In addition, we improved the model performance by surgically identifying the relevant data to be accessed only, for instance, in Table 3.6 the performance of RFL over DNN is increased by $\sim 10\%$ by adopting all the factors (F1-3) compared with only considering the F1 factor. This trend holds for the rest of the query-centric DML mechanisms.

In terms of communication rounds among nodes during the DML process, our mechanism can reduce the communication by avoiding engaging unsuitable subsets of nodes in one round. This is because only one round is dedicated to determining the node rankings based on supportive clusters using F1-3 factors, followed by determining the amount of data in each relevant cluster. Using the F1 and F2 factors, no communication round between the leader and the nodes is required. If the conditions in factors F1 and F2 are satisfied then the DML process commences. Subsequently, we assess the number of relevant samples in each separate cluster. Otherwise, a node will not participate in the training stage.

Given the fact that RFL are mostly superior than the other DML mechanisms schemes across static and dynamic environments, datasets, and predictive models, we further provide a detailed comparative assessment of our mechanism (both, adopring F1 and F1-3 factors) against the approaches under comparison by showing the *distribution of the MSE per query* in Figures 3.10 and 3.11. Note that similar results are obtained for the rest of the DML mechanisms. We obtain insights about the percentage of the DPA queries served w.r.t. prediction error across each



((a)) Distribution of MSE vs. proportion (%) of queries using RFL across different models in static data environment (DS1 dataset).

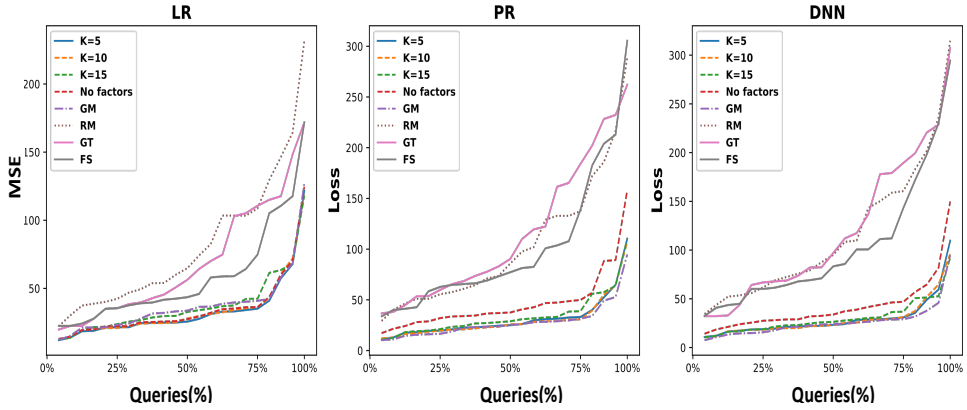


((b)) Distribution of MSE vs. proportion (%) of queries using RFL across different models in dynamic data environment (DS1 dataset).

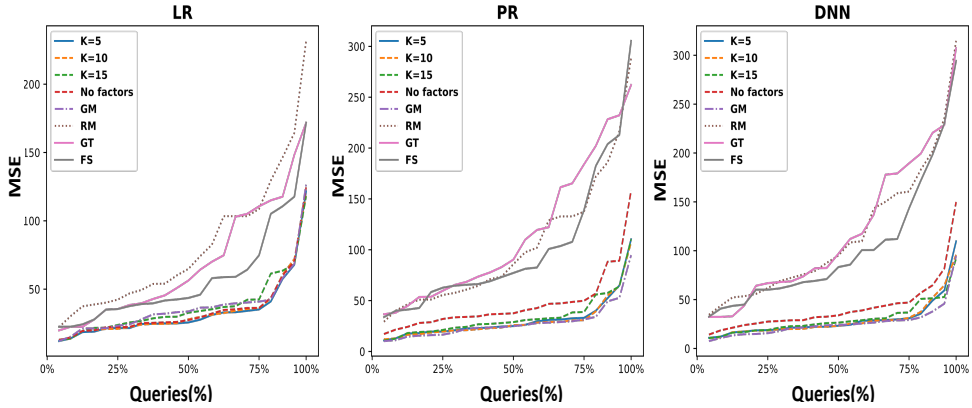
Figure 3.10: Distribution of MSE against proportion (%) of DPA queries (from the query workloads) using RFL in static and dynamic data environments (DS1).

query workload, and how all the mechanisms achieve low MSE across all incoming queries. Specifically, Figures 3.10 and 3.11 show that given a MSE value, which proportion of the issued queries (out of all queries in the workload) assumes error up to that MSE value. It is evidenced that our mechanism is efficient in terms of selecting the most appropriate nodes to train a DML model per query. As evidenced by the performance of FS, GT, and RM in both datasets for both environments, the performance of each query individually is negatively impacted when considering 100% data access during the DML process. That is, for more than 50% of the queries, the obtained MSE by FS, GT and RM is significantly higher than our mechanism.

Moreover, one can observe the robustness of our mechanism where for up to 90% of the queries (see Figure 3.11; both static and dynamic data environments), the obtained MSE remains almost constantly in low levels while being comparable with that of GM. This yields our mechanism applicable in the real of DPA.



((a)) Distribution of MSE vs. proportion (%) of queries using RFL across different models in static data environment (DS2 dataset).



((b)) Distribution of MSE vs. proportion (%) of queries using RFL across different models in dynamic data environment (DS2 dataset).

Figure 3.11: Distribution of MSE against proportion (%) of DPA queries (from the query workloads) using RFL in static and dynamic data environments (DS2).

3.6.4 Limitations & Directions of Enhancement

Node selection in DML environments has several perspectives related to DPA. Our paradigm predicts the most suitable nodes holding the most relevant data that will be used for training ML models in a distributed manner. Our mechanisms excel where there is significant heterogeneity and diversity between the query access patterns and the available data in each node. Hence, by selecting a sub-set of nodes for each incoming DPA query into the system is of paramount importance in terms of our objectives: model performance and data access load. Nonetheless, our node selection process needs to be expanded to accommodate different perspectives. Specifically, node selection can take into consideration the accessibility of nodes due to the network status and connectivity. One limitation is that DPA tasks are based on the assumption that selected nodes are also accessible and available during model training. This implies that these nodes should devote their resources and computational load when requested to be engaged in ML models distributed learning. However, 'stragglers' (nodes with limited computational capacity) and heavy loaded

nodes might hinder the entire process. Proactive mechanisms for refining the outcome of node selection e.g., [69] or providing guarantee about the capacity of the selected nodes e.g., [136] is a limitation in our paradigm.

In addition, our paradigm does not focus on the system resilience in DML environments. Even if the selected nodes have been proved to accommodate the learning process for a specific DPA query, failure rates, nodes attacks, and unexpected events yielding some of the selected nodes incapable to support the learning process would jeopardise the DPA tasks [137]. Factors including device heterogeneity, heterogeneity in network connectivity (e.g., mobile and stationary nodes), and incentives for nodes devoting their resources for training define perspectives and directions where our current paradigm does not include in the node selection process in this work.

Finally, in terms of the nature of data, our mechanism may be less effective with other forms of selection queries like object recognition over moving images, and multi-modal learning of ML models over simultaneously different types of data like text, audio, or images. Moreover, when the amount of data is similar across many nodes, then this would require our mechanism to be enhanced with methods to eliminate potential nodes hosting data with similar distributions to avoid redundancy in the learning process. The above-mentioned perspectives for enhancement of our paradigm are included in our future agenda for developing a holistic paradigm in the emerging area of distributed analytics.

3.7 Conclusions

We introduce an innovative node and relevant data selection paradigm to support DPA in DML environments. Our objective is to predict the most suitable nodes with the most relevant data to be engaged in DML process of ML models in light of achieving high predictive performance while avoiding redundant access over irrelevant data. Our principle is based on filtering out nodes whose data do not match the query access patterns. We provide comprehensive experimental evaluation and comparative assessment with other methods found in the literature over query workloads and datasets to assess the robustness, scalability and performance of our mechanisms. Our paradigm showcased to outperform baselines and relevant mechanisms, thus, demonstrating its ability to predict the most suitable sets of nodes tailored to DPA queries. Moreover, our DML mechanisms are proved efficient in reducing the training communication rounds and redundant access over irrelevant data ensuring streamlined interactions among selected nodes. We finally elaborated on the directions of enhancement of our paradigm towards a holistic eco-system in the realm of DPA.

Table 3.10: Table of Symbols for Chapter 3.

Symbol	**Definition**
$\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$, $\mathcal{N}' \subset \mathcal{N}$, $\mathcal{N}^\star \subset \mathcal{N}$	Set of nodes, selected nodes, ideal selected nodes.
$\ell = \lvert\mathcal{N}'\rvert \leq N$	Number of selected nodes.
$\mathbf{x} = [x_1, \ldots, x_d]^\top \in \mathcal{X}$, $y \in \mathcal{Y}$	Input and output.
d	Data dimensionality.
$\mathbf{q}$	DPA query.
$\mathcal{B}$	Hyper-rectangle in $\mathcal{X}$
$\mathcal{D}_i = \{(\mathbf{x}, y)_k\}_{k=1}^{L_i}$	Node n_i's local dataset with L_i samples.
$f(\mathbf{x}; \theta(\mathbf{q}))$	DML model; $\theta(\mathbf{q})$ parameters.
$\mathcal{D}(\mathbf{q}) = \cup_{i=1}^{N} \mathcal{D}_i(\mathbf{q})$, $\mathcal{D}_i^{''} \subset \mathcal{D}_i' \subset \mathcal{D}$	*Whole* distributed data and relevant data per query/node.
$\mathcal{L}$	Loss function.
$q_k^{\min}$, $q_k^{\max}$	Boundaries per query.
e_j,e_G	Error of central and local nodes.
$\{C_1, \ldots, C_K\}$	Set of K clusters.
$\mathbf{c}_k$, $\mathbf{c}_k'$	k-th cluster-head & boundary.
$r_i(\mathbf{q}), r_i'(\mathbf{q}), r_i^{''}(\mathbf{q})$	Ranking support for n_i.
$h_k(\mathbf{q}) \in [0, 1]$	Query-cluster overlapping.
$K' \leq K$	Number of supportive clusters.
$\epsilon > 0$	Overlapping threshold.
$t, \tau \in \mathbb{T}, \mathcal{T}$	Discrete time, training epochs.
α	Node selection accuracy.
ν	Ring passes (RIL).

Chapter 4

Cluster-based & Label-aware Federated Meta-Learning for On-Demand Classification Tasks

4.1 Introduction

Distributed clients like road-side units and edge micro-servers collect and store diverse forms of data like images and voice [138]. They are valuable sources of data, improving accuracy and generalization of ML and DL models for classification tasks. Model training over clients' data yields robust predictive results compared with single sources [139]. However, centralizing data on servers presents challenges, including data privacy and the sheer volume of data [140]. FL emerges as a distributed learning paradigm to address these issues. FL collaboratively trains a global model across clients facilitating access to distributed data [140].

The primary challenge in FL is data and class labels heterogeneity. This issue arises due to various types of distribution shifts among clients including feature, label, and concept distribution shifts as evidenced in [141, 9]. Such challenge revolves around uneven distributions of data across classes: majority of samples belong to majority classes while minority classes comprise only a small amount of data [8]. Limited labelled data caused by disparities in labels across clients impede the convergence of classifiers degrading their performance. This has a detrimental impact when classification involves minority classes [9] or *any* arbitrary set of labels.

Meta-learning has proved to accelerate model adaptation to arbitrary labels by allowing fine-tuning over small datasets when faced with previously unseen tasks [142]. The adoption of meta-learning in FL concentrates on what is termed as 'perfect setups', which are challenging to implement in real-world applications. These setups hinge on several key assumptions. **A1:** Classification tasks share exactly the same set of labels $\mathcal{L}$ and label distribution as those used in training meta-models. It is thus not possible to deal with any arbitrary out-of-distribution classification requests. **A2:** The labels are evenly distributed among all the clients [143]. **A3:**

Most approaches rely on limited number of labels, often $|\mathcal{L}| \approx 10$ [144, 145, 146, 147, 148, 149]. Inadequate attention is given to cases with larger number of labels (e.g., more than 20 or 100) where *not* all labels $\mathcal{L}$ are available in each client. In real settings, **A1-A3** might not hold, thus, cannot satisfy on-demand tasks requesting training classifiers for *any* arbitrary label subset of the available $\mathcal{L}$. In such settings, relying on a single, global Federated Meta-Learning (FML) model proves to be inefficient and impractical [150] to accommodate (i) any arbitrary classification tasks and (ii) out-of-distribution labels across clients.

An *on-demand classification task*, or simply *task*, requests the training of a classifier over distributed clients' data, where the data are labelled with labels from a set $\mathcal{T} \subset \mathcal{L}$. A task is associated with its set of labels $\mathcal{T}$. **Note:** in traditional FML and FL, we obtain the trivial case $\mathcal{T} \equiv \mathcal{L}$ where *all* tasks are the same; implying that *all* clients host data corresponding to *all* labels in $\mathcal{L}$ (unevenly or not). Consider, for instance, urban planning in the intelligent transportation systems domain. Here the construction department focuses on the task of training image classifiers for road surface defect detection correlated with road material over data with labels $\mathcal{L}' = \{$'gravel road', 'asphalt road', 'concrete road'$\}$. Meanwhile, the carriageway maintenance department utilizes some of the same data with defect labels $\mathcal{L}'' = \{$'transverse crack', 'block crack', 'pothole'$\}$ for maintenance tasks [151]. Consequently, data from both departments are neither locally accessible in the same clients nor data are labelled with all labels in each client individually. A real challenge is when a task requires the training of a classifier over data labelled with a subset of labels $\mathcal{T}$ out of all available labels $\mathcal{L}$ across clients. Clients may possess data with labels in $\mathcal{T}$ exhibiting varying levels of class imbalances or may have data with fewer (evenly balanced) classes than those in $\mathcal{T}$, which is not unusual in real-world problems.

A straightforward solution would be to develop separate classifiers for each individual task, ensuring optimal performance for each task within the context of FL/FML. This requires $O(2^{|\mathcal{L}|})$ classifiers to be trained and maintained to accommodate *any* possible on-demand request for classifiers associated with any label subset of $\mathcal{L}$. This is undeniably impractical and does not scale even for moderate values of $|\mathcal{L}|$. Moreover, creating models from scratch *every* time a *new* task arrives is impractical especially when dealing with imbalanced classes. Therefore, this chapter introduces a Cluster-based & Label-aware FML framework (CL-FML) that addresses such challenges, diverging from standard FL and FML paradigms. CL-FML initially groups clients together sharing *almost* overlapping subsets of labels w.r.t. holistic label set $\mathcal{L}$ derived from all available labels across clients. The label-aware clustering logically gathers clients together based on label shifting, thus, mitigating label imbalance *per* task.

CL-FML harnesses the capability of decentralized FML to perform effectively by fine-tuning any on-demand task. This chapter study the cases of training more than one (reusable) meta-model tailored to available labels $\mathcal{L}_k \subset \mathcal{L}$ of a cluster of clients C_k, hereinafter referred to as *cluster-based meta-model*. The sizes of such meta-models are compact to be stored on clients temporarily, enabling them to be reused for future tasks. Each of these meta-models serves a

specific range of label distributions ($\mathcal{L}_k$) being responsible for a task involving at least some of the labels $\mathcal{L}_k$. In our example, the task requires data within specific labels' ranges, i.e., classification requires labels $\mathcal{L}'$ from construction department and labels $\mathcal{L}''$ from maintenance department with $\mathcal{L}' \cap \mathcal{L}'' \equiv \emptyset$. **Note:** it might also be possible a task requires one or more labels that do not belong to a cluster of clients. Hence, our objective is not only limited to adapting meta-model solely to tasks with exactly the same distribution; it copes with sharing meta-models among clusters to further fine-tune in light of satisfying all requested labels from $\mathcal{T}$. We leverage lightweight data augmentation to generate data for labels that do not appear in that cluster ($\mathcal{T} \setminus \mathcal{L}_k$). Such generative data augmentation models, like [152], will be trained in advance for each cluster C_k. Selected clients per cluster train such models w.r.t. their capacity and label availability. These models are then selectively shared among clusters to augment data within a cluster with labels from $\mathcal{T} \setminus \mathcal{L}_k$.

To the best of our knowledge, CL-FML is the first approach that tackles on-demand classification tasks introducing distributed label-aware client clustering and multiple cluster-based meta-models. Our technical **contributions** are:

- The CL-FML framework leverages label distribution based clustering of clients to identify the *most suitable* clients to be engaged *per* task.
- Multiple cluster-based meta-models are introduced, one per cluster. A task-tailored FML model is trained to handle on-demand classification tasks with overlapping labels in the cluster, enabling fast model adaptation and reusability for new tasks through selective fine-tuning.
- A novel generative model training process is introduced among selected clusters to address data unavailability for tasks.
- A comprehensive comparative assessment of CL-FML is conducted against baselines DFedAvg [153], cluster-based DFedAvg (C-DFedAvg) [154], and group-based FML (G-FML) [155] over benchmark datasets under various tasks and label distributions. The experiments showcase the effectiveness and efficiency of CL-FML in tackling on-demand tasks in distributed learning environments.

4.2 Related Work

In FL, clients are randomly selected per round, while global model aggregation is centrally performed. This is effective over independent and identically distributed (i.i.d.) data, however, unsuitable for non-i.i.d. data, where each client owns statistically heterogeneous local data. The latter rises *concept*, *label*, and *feature* distribution shifts that hinder the whole performance due to local model drifts [147]. Prior efforts [156, 148] addressed concept and feature shifts.

Conversely, label shifts arise when marginal label distributions $\mathcal{P}(\ell), \ell \in \mathcal{L}$, differ across clients even if conditioned distributions $\mathcal{P}(\ell|x)$ remain the same. Several approaches tackle label skews where clients' data have fixed number of labels $\mathcal{L}$, e.g., [149]. A few studies have recently investigated label shifting due to variations in label distribution range [157]. As discussed in Section 5.1, some clients own data with labels only from a subset of $\mathcal{L}$ and with significant label imbalance, which is part of our study along with the size $|\mathcal{L}|$. To deal with label shifting, [157] introduced multiple global models that improved performance compared with a single global FL model. This motivated the cluster-based FL: each global model per cluster. [158] and [155] focus on clustering clients based on models' similarity and data distribution, respectively. However, such clustering demands significant communication overhead due to data sharing. CL-FML, instead, introduces light-weight clustering based only on sharing label distributions. [159] clusters clients based on similarities among labels to tackle label range differentiation. However, such approach assumes that after clustering, labelled balanced clients are obtained ideally for training a global model. Therefore, inherent label imbalances still exist in clustered clients' data, which signifies fundamental statistical heterogeneity in FL. CL-FML's strategy is deemed crucial to develop multiple models and label-aware clustering, which addresses label shifting over such heterogeneity.

Meta-learning and multi-task learning expedite the training of tasks leveraging other related tasks. In FML, meta-learning trains a global meta-model for only specific, pre-determined, tasks [142], [160], while multi-task learning trains a generalized model across clients, facilitating knowledge transfer from different tasks [161]. However, as the number of clients increases, thus, data heterogeneity grows, employing single meta-learning and/or multi-task learning models becomes infeasible and unscalable. Even if approaches, e.g., [162, 163] introduce client grouping to address scalability, they merely focus on transferring knowledge in non label-shifting cases, i.e., all clients share the same labels to equally utilize meta-knowledge across all tasks [143]. Nevertheless, we may still encounter label imbalances within clusters. CL-FML, instead, leverages transfer learning to initially build multiple models capturing intra-cluster heterogeneity capable of accommodating on-demand tasks involving arbitrary label subsets from $\mathcal{L}$.

4.3 Target Data Types and Application Scope

The proposed CL-FML framework is designed to operate in environments where data is highly privacy-sensitive and cannot be shared across entities. This includes domains such as healthcare, finance, and organizations dealing with proprietary or confidential data. Although our experimental evaluation uses standard image benchmarks—MNIST, Fashion-MNIST, EMNIST, and CIFAR-100—these datasets were selected to demonstrate the method's performance across grayscale and colored images, varying dataset sizes, and a broad spectrum of class counts, ranging from a few to hundreds of labels.

Beyond these benchmarks, CL-FML is broadly applicable to a wide range of tasks and application scenarios. It can be used to support critical sectors by enabling the construction of supervised classification models, regression models, or even unsupervised clustering models, depending on the nature of the task. In this chapter, we specifically focus on classification tasks under the challenging conditions of label distribution shift and data heterogeneity, which are common in real-world decentralized federated learning settings where clients hold non-IID and unbalanced labeled data. For example, CL-FML can be effectively applied in healthcare diagnostics, where different institutions may record distinct sets of diseases or medical conditions. Similarly, in autonomous driving, each vehicle may locally observe a unique subset of road scenes or traffic signs. The label-aware clustering mechanism in CL-FML ensures that clients with similar label distributions are grouped to collaboratively build cluster-specific meta-models. These models are then fine-tuned using only a small fraction of labeled or augmented data, enabling efficient and accurate adaptation to on-demand tasks. This design allows CL-FML to minimize communication rounds while maintaining high classification performance, making it a practical and scalable solution for federated learning environments with strict privacy constraints.

4.4 Preliminaries

Centralized & Decentralized Federated Learning: Consider a distributed learning system with N clients $\mathcal{N} = \{n_1, \ldots, n_N\}$. Let D_i be the local dataset of a client $n_i \in \mathcal{N}$. In Centralized FL (CFL) [10], given a subset of $N' < N$ clients $\mathcal{N}' \subset \mathcal{N}$, the local loss is:

$$\mathcal{R}_i(\theta) = \frac{1}{|D_i|} \sum_{(x,y)\in D_i} \mathcal{J}(f(\theta; x), y) \tag{4.1}$$

where θ is the model parameter, f is the discriminant function/model mapping input x to class y, $\mathcal{J}$ measures cross-entropy loss. The global loss for all selected clients $n_i \in \mathcal{N}'$ is:

$$\mathcal{R}(\theta) = \sum_{n_i \in \mathcal{N}'} \rho_i \mathcal{R}_i(\theta), \text{ where } \rho_i = \frac{|D_i|}{\sum_{n_j \in \mathcal{N}'} |D_j|}. \tag{4.2}$$

The model training process spans periodically over T global rounds with E local rounds. Let $t \in \{1, \ldots, T\}$ be a discrete training round and $\tau = \lfloor \frac{t}{E} \rfloor E$ be the start time of the current global epoch. At τ, the clients receive updated aggregated weights $\bar{\theta}^\tau$ from a centralized server, which aggregates the clients' model parameters. The local training at n_i at epoch $e = 1, \ldots, E$ is:

$$\theta_i^{(\tau+e)+1} = \theta_i^{\tau+e} - \eta_{\tau+e} \nabla \mathcal{R}_i(\theta_i^{\tau+e}), \tag{4.3}$$

where $\eta \in (0, 1)$ is the learning rate. The weight averaging policy on server is: $\bar{\theta}^\tau = \sum_{n_i \in \mathcal{N}} \rho_i \theta_i^\tau$.

Decentralized FL (DFL) [153] allows each client n_i to *only* communicate with its neighbors

defined as a collection $\mathcal{N}_i \subset \mathcal{N}$ of clients with connections between them. Hence, there is no need for a centralized server to aggregate the locally updated models as in CFL. Instead, at round t, each client n_i first aggregates the models received from its neighbors $n_j \in \mathcal{N}_i$, i.e., $\theta_i^t = \sum_{n_j \in \mathcal{N}_i \cup \{n_i\}} \theta_j^t$ and trains using local data D_i:

$$\theta_i^{t+1} = \theta_i^t - \eta_t \nabla \mathcal{R}_i(\theta_i^t). \tag{4.4}$$

In both CFL and DFL, the aggregated model on server and models in each client, respectively, will become the global model in the next round and continue to participate and communicate in training until this global model converges. **Data Augmentation:** Given non-i.i.d. and heterogeneous data, it is common for the class distribution to be highly imbalanced. To address this issue, CL-FML adopts the lightweight data augmentation method, MixUp [138], to locally augment data x with labels from $\mathcal{L}$. MixUp is a commonly used computationally efficient technique applied to classification tasks. **Note:** any data augmentation mechanism, such as CGAN[164] or simple techniques like cropping, flipping, rotation, and color transformations [165], could also be adopted, conditioned to a client's capacity, without compromising the efficacy of our method. Mixup uses linear interpolation between two input-label pairs, x_k and x_ℓ with labels y_k and y_ℓ, to synthesize the sample (x', y'): $x' = \lambda x_k + (1-\lambda)x_\ell$ and $y' = \lambda y_k + (1-\lambda)y_\ell$, with $\lambda \in (0, 1)$ controlling interpolation between samples.

4.5 The CL-FML Framework

4.5.1 Overview

The CL-FML framework extends the principles of decentralized FL (DFL) involving multiple meta-models and local generative models per cluster. The fundamental processes are illustrated in Figure 4.1. We consider a distributed environment with N clients $\mathcal{N}$. The clients are connected given a network topology represented by a directed graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$. The adjacency matrix $\mathcal{E} = [e_{i,j}] \in \mathbb{R}^{N \times N}$ defines the neighborhood $\mathcal{N}_i = \{n_j \in \mathcal{N} : e_{i,j} > 0\}$ of client $n_i \in \mathcal{N}$ as the subset of clients n_j that directly communicate with n_i. $e_{i,j} = 0$ indicates no communication between clients, i.e., $n_j \notin \mathcal{N}_i$. Also, $e_{i,j} = e_{j,i}$ may not always be valid for $n_i \neq n_j$. $\mathcal{E}$ is fixed. Each client n_i collects local labelled data $D_i = \{\mathbf{X}_i \times \mathbf{Y}_i \sim \mathcal{P}_i : \mathbf{X}_i \in \mathbb{R}^d; \mathbf{Y}_i \in \mathcal{L}\}$ from an unknown joint probability distribution $\mathcal{P}_i$. For any pair of clients (n_i, n_j) with $n_j \neq n_i$, the joint probability distributions can be either similar ($\mathcal{P}_j \approx \mathcal{P}_i$) or dissimilar ($\mathcal{P}_j \neq \mathcal{P}_i$). Clients do not rely on a third party to manage shared models, which precludes centralized model averaging schemes, like centralized FL. Any ML/DL model $f(x; \theta) \in \mathcal{F}$, e.g., image classifier, coming from a hypothesis class $\mathcal{F}$, contains all the model parameters $\theta \in \Theta$ trained locally from a d-dimensional parameter space Θ. We focus on classification, which can be easily extended to regression.

We consider unequal data distributions across the classes (labels) $y \in \mathcal{L}$. Each label in client n_i may have different distribution compared to other labels in $\mathcal{L}$. Label imbalance refers to a situation where the ratio between the number of samples belonging to a class to the total number of samples in n_i differs significantly for different classes. We use y and ℓ interchangeably to represent class and labels.

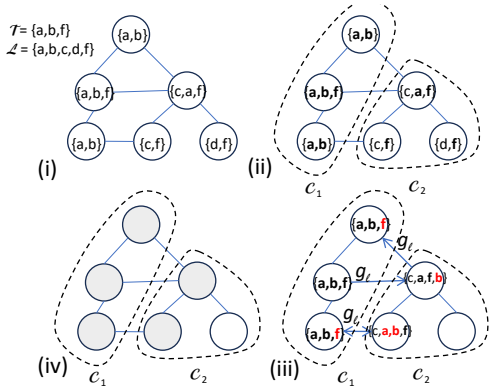


Figure 4.1: CL-FML instance (clockwise). **(i)** All clients' labels $\mathcal{L} = \{a, b, c, d, f\}$ and task's labels $\mathcal{T} = \{a, b, f\}$. **(ii)** Label-aware clustering into C_1, C_2 groups and decentralized cluster-based meta-learning. **(iii)** Decentralized training of g_ℓ for data augmentation. **(iv)** Decentralized fine-tuning for task-tailored meta-model among suitable clients (shaded).

4.5.2 Label-aware Client Clustering

We rely on the configurations introduced in [166, 167, 168, 169] for sharing only label distribution among clients. It is the minimum sufficient statistic to approximate a prior label distribution per cluster [168, 169]. Let $\mathcal{L} = \{\ell_1, \ldots, \ell_M\}$ be all the available labels across all clients in the network (see Figure 4.1(i)). Evidently, all labels are not known to all clients in advance. Clients have data with some labels from $\mathcal{L}$ and, in real cases, not all of them. To make the clients aware of the available labels, we introduce a label-aware distributed mechanism.

Ring-based Label Dissemination

Each client n_i disseminates only its local labels $\mathcal{L}_i \subset \mathcal{L}$ to neighbors. Eventually all clients are aware of $\mathcal{L}$. Let us assume a ring topology (any other topology could be adopted), where client n_i sends a message to its neighbour n_j and receives a message from another neighbour n_l. The label-aware mechanism passes messages of clients in the ring at rounds. At round t, client n_i expands its local label set $\mathcal{L}_i$ with the labels received from n_l, i.e., $\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \mathcal{L}_l$ and sends $\mathcal{L}_i$ to n_j. **Note:** Within up to $O(N)$ rounds, all the clients have received all the available labels of the network, i.e., $\mathcal{L}_i \equiv \mathcal{L}$. $O(N)$ is the *worst* case, where all initial label sets $\mathcal{L}_i$ are different from each other and singletons, i.e., $\mathcal{L}_i \cap \mathcal{L}_j \equiv \emptyset, \forall i, j$ and $|\mathcal{L}_i| = 1, \forall i$.

Label-aware Clustering

Based on the initial label set $\mathcal{L}_i$ and global label set $\mathcal{L}$, each client n_i represents its available labels with a probability distribution $\mathbf{p}_i = [p_1, \ldots, p_M] \in [0,1]^M$ using multi-hot encoding assigning to its available label a probability. Given $\mathcal{L}_i$ and $\mathcal{L}$, the multi-hot encoding vector $\mathbf{z} = [z_1, \ldots, z_M] \in \{0,1\}^M$ has $z_k = 1$ if the label $\ell_k \in \mathcal{L}_i$; $z_k = 0$, otherwise, $k = 1, \ldots, M$ and $\ell_k \in \mathcal{L}$:

$$p_k = \frac{z_k}{\sum_\kappa z_\kappa}, k = 1, \ldots, M \text{ with } \sum_{k=1}^{M} p_k = 1. \tag{4.5}$$

The entry p_k expresses the probability client n_i has data classified with label ℓ_k. In parallel to label-awareness mechanism, the nodes in the ring topology can efficiently elect a **leader** client n_l, e.g., using efficient and scalable ring-based leader election [170]. Leader n_l initiates a Minimum Spanning Tree (MST) to incrementally gather all probability label vectors $\{\mathbf{p}_i\}_{i=1}^N$, which will be used for clustering the clients into $K < N$ clusters based on these probability mass functions. The leader groups together the nodes' label distributions into K clusters. Each cluster is represented by the *cluster label distribution* $\mathbf{w}_k = [w_{k1}, \ldots, w_{kM}]$ associated with the labels $\ell_1, \ldots, \ell_M$, respectively. The cluster label distributions $\mathbf{w}_k$ are incrementally updated upon receiving a client's label distribution $\mathbf{p}_i$. The clustering objective is to minimize the expected quantization error $\mathcal{E}$:

$$\mathcal{E}(\{\mathbf{w}_k\}) = \mathbb{E}_{\mathbf{p}\in[0,1]^M}[\mathcal{H}(\mathbf{w}^*, \mathbf{p})|\mathbf{w}^* = \arg\min_k \mathcal{H}(\mathbf{w}_k, \mathbf{p})], \tag{4.6}$$

where cluster label distribution $\mathbf{w}^*$ is the closest to a client label distribution $\mathbf{p}$. We adopt the symmetric and bounded Hellinger distance [171], $\mathcal{H} \in [0,1]$, between probability label distributions. Hellinger distance is widely used for class imbalance problems [156, 148]. The Hellinger distance $\mathcal{H}(\mathbf{p}, \mathbf{q})$ between two discrete probability distributions $\mathbf{p} = \{p_m\}_{m\in[M]}, \mathbf{q} = \{q_m\}_{m\in[M]}$ is:

$$\mathcal{H}(\mathbf{p}, \mathbf{q}) = \frac{1}{\sqrt{2}}\Big(\sum_{m\in[M]} (\sqrt{p_m} - \sqrt{q_m})^2\Big)^{\frac{1}{2}}. \tag{4.7}$$

When $\mathcal{H}$ is 0, the two distributions are identical, while the maximum distance 1 is achieved when the two distributions are furthest apart. The cluster distributions $\mathbf{w}_k^t$ are updated when the label distributions is received from clients at step t. The update is carried by incrementally minimizing (4.6) using stochastic gradient descent to obtain the optimal cluster distributions:

$$\mathbf{w}_k^{(t)} \leftarrow \mathbf{w}_k^{(t-1)} - \eta\nabla\mathcal{E}(\{\mathbf{w}_k^{(t-1)}\}) \tag{4.8}$$

given an incoming label distribution $\mathbf{p}^{(t)}$, with learning rate $\eta \in (0, 1)$. The update rule for the j-th component of the cluster distribution $\mathbf{w}_k^{(t)}$ is then:

$$w_{kj}^{(t)} = \begin{cases} w_{kj}^{(t-1)} - \frac{\eta}{4}\left(1 - \sqrt{\frac{p_j^{(t)}}{w_{kj}^{(t-1)}}}\right) \frac{1}{\mathcal{H}(\mathbf{w}_k^{(t-1)}, \mathbf{p}^{(t)})} \\ \text{if } k = \arg\min_{\kappa \in [M]} \mathcal{H}(\mathbf{w}_\kappa^{(t-1)}, \mathbf{p}^{(t)}), \\ w_{kj}^{(t-1)} \text{otherwise.} \end{cases} \tag{4.9}$$

Once the cluster distributions $\{\mathbf{w}_k\}_{k \in [K]}$ have been concluded, the leader client assigns a client to a group of clients:

$$C_k = \{i \in \mathcal{N} : k = \arg\min_{\kappa \in [M]} \mathcal{H}(\mathbf{w}_\kappa, \mathbf{p}_i)\}, k \in [K]. \tag{4.10}$$

Based on the probability of labels in the cluster distribution vector $\mathbf{w}_k$, the associated group C_k is considered to have the list of the most probable labels $\mathcal{L}_k = \{\ell_\kappa \in \mathcal{L} : w_{k,\kappa} \geq \phi\}$, given a probability threshold $\phi \in (0, 1)$. That is, clients belonging to a group C_k have labels from $\mathcal{L}_k$, each one with at least probability ϕ. Note, these clients might also have additional labels, however, with potential probability less than ϕ. Therefore, the leader client uses MST to disseminate to each neighbour i (and the latter to their neighbors in turn) the group assignment and label message $\langle k, \mathcal{L}_k, \mathbf{w}_k \rangle$, i.e., client i belongs to group C_k where all the members have at least the labels from $\mathcal{L}_k$ associated with probability distribution $\mathbf{w}_k$. The N clients then split into K disjoint groups based on their most eminent label distributions (Figure 4.1(ii)).

To clarify, to construct the K disjoint client groups, CL-FML adopts a label-aware clustering approach based on the similarity of clients' local label distributions. During initialization, each client $n_i \in \mathcal{N}$ shares its observed label set L_i with its ring neighbor. These label sets are collected by the cluster leader, which computes similarity scores between each client's label set and a set of cluster prototypes $\{w_k\}_{k=1}^K$ that represent dominant label distributions across the network. The similarity is measured using the Jaccard index, as formalized in Equation (4.13), which quantifies the overlap between two label sets A and B as:

$$\text{sim}(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Based on these similarities, each client n_i is assigned to the most compatible cluster C_k whose prototype w_k is most similar to its label set L_i. This process results in a partitioning of the network into K disjoint clusters $\{C_k\}_{k=1}^K$, each consisting of clients with closely related label distributions. These label-coherent clusters form the basis for training specialized meta-models that are more aligned with the underlying data characteristics within each group.

4.5.3 Cluster-based Multiple Meta-model Learning

The primary objective is to train a tailored decentralized meta-model f_k for each cluster $C_k, k \in [K]$, capable of fast and flexible adaptation to on-demand tasks with varying label sets $\mathcal{A} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$. This is achieved via fine-tuning using selected samples from clients belonging to each cluster C_k. Consider a client $n_i \in C_k$ dividing its dataset D_i into two disjoint sets: *meta-training* set and *query* set. The meta-training set $D_i^M \subset D_i$, is used to train the cluster's meta-model f_k. f_k serves as the starting point to learn a generic representation of clients' data in C_k capable of adapting to future tasks' labels $\mathcal{T}_j \in \mathcal{A}$ assigned to C_k. The query set $D_i^Q \subset D_i$, with $D_i^M \cap D_i^Q \equiv \emptyset$, is selected to ensure equal representation of samples across all class labels $\mathcal{L}_k$ of cluster C_k. Therefore, D_i^Q refers to labeled-balanced samples eliminating class imbalances in the fine-tune stage of f_k. Within cluster C_k, each client $n_i \in C_k$ randomly initializes its local meta-model f_k's parameters $\theta_{k,i}$ and shares with its neighbors clients in the clusters. Each client n_i locally updates $\theta_{k,i}$ along with neighbors $\theta_{k,j}, n_j \in \mathcal{N}_i$ deriving a new local meta-model from its meta-training set D_i^M over local epochs E_M using mini-batch SGD. During round $t \in \{1, \ldots, T\}$, n_i aggregates its neighbors local meta-models as:

$$\tilde{\theta}_{k,i}^t = \sum_{n_j \in \mathcal{N}_i} \omega_j \theta_{k,j}^t \tag{4.11}$$

with $\omega_i = \frac{|D_i^M|}{\sum_{n_j \in \mathcal{N}_i} |D_j^M|}$ and then computes the gradient of the loss $\nabla \mathcal{R}(\tilde{\theta}_{k,i}^t)$ updating the local meta-model as:

$$\theta_{k,i}^{t+1} \leftarrow \tilde{\theta}_{k,i}^t - \eta \nabla \mathcal{R}(\tilde{\theta}_{k,i}^t), \tag{4.12}$$

with learning rate $\eta \in (0, 1)$. The cluster-based meta-model $\tilde{\theta}_k^T = \tilde{\theta}_{k,i}^T, \forall n_i$ is then passed to all clients in the cluster (Figure. 4.1(ii)). This meta-model locally maintained on each client serves as an initial model for fine-tuning the requested classifier associated with any future task from $\mathcal{A}$. Note, the meta-models $\{\tilde{\theta}_k^T\}_{k=1}^K$ may exhibit biases due to their training on imbalanced data across clients. Nevertheless, the objective obtaining these meta-models is to provide valuable starting points that will significantly accelerate the training process of the classifiers requested by tasks $\mathcal{A}$.

4.5.4 Task-tailored Distributed Meta-model Learning

Each client n_i is allocated to a cluster and is equipped with a meta-model $\tilde{\theta}_k^T$. Consider a new incoming task with label set $\mathcal{T}$ requesting the training of a classifier over distributed clients' data with labels $\mathcal{T} = \{\ell_\tau\} \subseteq \mathcal{L}$. The task should be assigned initially to a group C_k of clients that have the majority of the labels requested in set $\mathcal{T}$ based on the closest group cluster distribution, i.e., $k = \arg\min_{\kappa \in [K]} \mathcal{H}(\mathbf{w}_\kappa, \mathbf{q})$. $\mathbf{q} = \{q_m\}_{m \in [M]}$ is the probability label distribution of the task's requested labels $\mathcal{T}$, which can be trivially derived using multi-hot encoding discussed above.

We distinguish two cases:

Case I. If $\mathcal{T} \subseteq \mathcal{L}_k$, then, group C_k is the most suitable to directly handle this task involving its clients in the training.

Case II. If $\mathcal{T} \supset \mathcal{L}_k$, then the group C_k initiates a process for handling the labels in $\mathcal{T} \cap \mathcal{L}_k$, while we involve clients from other clusters $\{C_m\}_{m=1}^{K} \setminus \{C_k\}$ capable of handling the rest of the labels in $\tilde{\mathcal{T}} = \mathcal{T} \setminus \mathcal{L}_k$. We keep engaging clusters until all their labels are included in $\mathcal{T}$. We then rank these clusters based on their label contribution to task $\mathcal{T}$ and engage the minimum number $m \leq K$ of those cluster whose $\cup_{\kappa=1}^{m}\{\mathcal{L}_\kappa\} \subseteq \mathcal{T}$.

We adopt the well-known Tversky index [172] for set similarity to quantify the ranking, or similarity, $sim(\mathcal{T}, \mathcal{L}_k)$ between the labels requested by task $\mathcal{T}$ and k-th cluster $\mathcal{L}_k$. This index takes consideration not only the common labels $\mathcal{T} \cap \mathcal{L}_k$ *but* also the labels that cannot satisfy the task $\mathcal{T} \setminus \mathcal{L}_k$, hereinafter referred to as *missing labels*, and the extra labels from cluster that are not actually needed by the task $\mathcal{L}_k \setminus \mathcal{T}$, referred to as *verbose labels*. The similarity is then defined:

$$sim(\mathcal{T}, \mathcal{L}_k) = \frac{|\mathcal{T} \cap \mathcal{L}_k|}{|\mathcal{T} \cap \mathcal{L}_k| + a_1|\mathcal{T} \setminus \mathcal{L}_k| + a_2|\mathcal{L}_k \setminus \mathcal{T}|}. \tag{4.13}$$

By setting equal importance on missing and verbose labels $a_1 = a_2 = 0.5$, we obtain the Sørensen–Dice coefficient [173] as the label sets similarity with $sim(\mathcal{T}, \mathcal{L}_k) \in [0, 1]$. Hence, we engage the top-m clusters ranked by $sim(\mathcal{T}, \mathcal{L}_\kappa)$, $\kappa \in [m]$.

After selecting the most suitable cluster C^* (Case I) or most suitable clusters $C_+^* \equiv \cup_{\kappa=1}^{m}\{C_\kappa\}$ (Case II), the associated clients are engaged in the distributed training of the classifier w.r.t. label set $\mathcal{T}$ (see Figure 4.1). These clients have at least data labelled with the labels in $\mathcal{T}$. Initially these clients use their cluster-based meta-models f_κ from cluster $C_\kappa \in C_+^*$ to start off the training process. However, even though a substantial amount of relevant labeled-data may be available for C^* (or C_+^*), there might still be a need for augmentation of data in group $C_\kappa \in C_+^*$ with labels $\mathcal{T} \setminus \mathcal{L}_\kappa$, which are not present in κ-th cluster's client data (missing labels). This is required to facilitate the fine-tuning of the requested task-tailored meta model. This is less problematic than the case where clients are randomly selected from arbitrary clusters with an expected low or almost zero similarity w.r.t. relevant labels in $\mathcal{T}$.

For each suitable cluster $C_\kappa \in C_+^*$, the corresponding clients locally identify their missing labels required per task. Hence, these clients generate augmented data labelled by the missing labels using a MixUp meta-model g_ℓ from clients in cluster $C_\ell \in C_+^*, \ell \neq \kappa$, for which these labels are not missing (see Fig.4.1(iii)), i.e., MixUp g_ℓ generates labelled samples (x, y) conditioned on the labels y locally on a client $n_i \in C_\kappa$ such that $\{(x, y) : y \in \mathcal{T} \setminus \mathcal{L}_\kappa\}$. Clients within the cluster individually use MixUp models with a low number of overlapping labels to avoid redundant computation.

Based on selective MixUp models for data augmentation, a client $n_i \in C_\kappa$ can now construct

its query set $D_i^Q = \{(x, y) : y \in \mathcal{L}_\kappa \cup (\mathcal{T} \setminus \mathcal{L}_\kappa)\}$, i.e., including (i) the actual data labelled with the requested task labels and (ii) the augmented data labelled with the associated missing labels from the locally augmented data by MixUp g_ℓ. Subsequently, the task-tailored meta-model, notated as $\hat{\theta}^{T'}$ is fine-tuned based on the query sets of the clients in the suitable clusters C_+^* after T' fine-tuning rounds. **Note:** the augmentation remains available within clients for future use. The local update of the distributed task-tailored meta-model $\hat{\theta}_{\kappa,i}^t$ at fine-tuning round $t = 1, \ldots, T'$, at client n_i from suitable cluster $C_\kappa \in C_+^*$ uses batch SGD over the query set D_i^Q is given by:

$$\hat{\theta}_{\kappa,i}^{t+1} \leftarrow \bar{\theta}_{\kappa,i}^t - \eta \nabla \mathcal{R}(\bar{\theta}_{\kappa,i}^t), \text{ with } \bar{\theta}_{\kappa,i}^t = \sum_{n_j \in \mathcal{N}_i'} \tilde{\omega}_j \hat{\theta}_{\kappa,j}^t, \tag{4.14}$$

where the weight:

$$\tilde{\omega}_i = \frac{\omega_i sim(\mathcal{T}, \mathcal{L}_\kappa)}{\sum_{n_j \in \mathcal{N}_i'} \omega_i sim(\mathcal{T}, \mathcal{L}_j)} \tag{4.15}$$

incorporates the similarity ranking of the κ-th cluster in C_+^*. The neighborhood $\mathcal{N}_i'$ of the node n_i refers to the neighboring clients belonging to the suitable clusters (see Figure.4.1(iv)). Finally, clients $n_i \in C^*$ obtain the fine-tuned $\hat{\theta}_{\kappa,i}^{T'}$ for task's label set $\mathcal{T}$ in (4.14) at the end of fine-tuning epoch T' as requested. We provide the whole CL-FML process in Algorithm 1. From line 1 to 15, clients are grouped into K clusters training the cluster-based meta-models. These processes are executed only *once*. When a new task arrives with label set $\mathcal{T}$, CL-FML proceeds as shown from line 17 to 28.

4.6 Experimental Evaluation

4.6.1 Experiment Setup

We compare CL-FML against baselines across different ad-hoc tasks with label distribution shifts and imbalances over benchmark datasets. **Note: source code is available in our repository**[1].

Data Sets

We conducted experiments on MNIST, sampling and distributing data over $N = 50$ clients ensuring that each client possesses samples from one of two label groups. Each label group comprises two distinct digits (classes). We also experiment using Fashion-MNIST with $N = 100$ clients. 20% of clients host samples from one of two distinct groups with different sample portions per label, where each label group consists of more than five classes (each group randomly chosen from $\binom{10}{5} = 252$ label subsets). The remaining 80% clients host data such that at least 90% of the samples are labeled with up to five labels and the rest 10% with any other random classes.

[1] https://anonymous.4open.science/r/DSAA-07DE

Algorithm 2 The CL-FML Process

Input: N clients; K groups; task's label set $\mathcal{T}$; labels $\mathcal{L}$
Output: Cluster-based meta-models $\{\tilde{\theta}_k\}_{k=1}^{K}$; task-tailored meta-model $\hat{\theta}$

/*Label-aware clustering*/
for round $t = 1, \ldots, N$ **do**
    Client n_i sends its labels $\mathcal{L}_i$ to ring neighbor;
end for
for client $n_i \in \mathcal{N}$ **do**
    Leader receives $\mathbf{p}_i$ and updates $\{\mathbf{w}_k\}_{k=1}^{K}$ using (4.8)
end for
Leader assigns clients to clusters $\{\mathcal{C}_k\}_{k=1}^{K}$ using (4.10)
/*Cluster-based meta-models*/
Each client $n_i \in \mathcal{C}_k, \forall k$ initializes $\tilde{\theta}_{k,i}^{t=0}$.
for training round $t = 1, \ldots, T$ at client n_i **do**
    Share $\theta_{k,i}^{t}$ among $\mathcal{N}_i$ neighbors;
    Aggregate and update $\tilde{\theta}_{k,i}^{t}$ using (4.11) and (4.12)
end for
Each client $n_i \in \mathcal{C}_k$ returns $\tilde{\theta}_{k,i}^{T}, \forall k$.
/*A task arrives with label set $\mathcal{T}$*/
One client per $\mathcal{C}_k$ calculates $sim(\mathcal{T}, \mathcal{L}_k), \forall k$ using (4.13)
Obtain suitable clusters $\mathcal{C}_+^*$ w.r.t. task's labels $\mathcal{T}$;
/*Augmentation for missing labels*/
for suitable client n_i from $\mathcal{C}_\kappa \in \mathcal{C}_+^*$ **do**
    Share MixUp g_ℓ among $\mathcal{N}_i$ neighbors;
    Augment labelled samples with $y \in \mathcal{T} \setminus \mathcal{L}_\kappa$
end for
for fine-tuning round $t = 1, \ldots, T'$ at suitable n_i **do**
    Share $\hat{\theta}_{k,i}^{t}$ among $\mathcal{N}_i'$ neighbors;
    Aggregate and update $\hat{\theta}_{k,i}^{t}$ using (4.14)
end for
Return: Task-tailored meta-model $\hat{\theta}^{T\prime}$ for task's labels $\mathcal{T}$. =0

For a more realistic scenario, we used EMNIST with $N = 200$ clients and $\mathcal{L} = 62$ unbalanced classes. 25% of clients possess samples from one of four distinct groups, where each group comprises 30 labels or more (randomly chosen). The remaining clients have from 2 to 30 of these classes and extra samples from any other classes. Finally, we used CIFAR-100 with $N = 100$ clients and $\mathcal{L} = 100$ classes grouped into 20 super-classes. 25% of clients possess samples from one of 20 groups, where each group comprises 10 labels or more (randomly chosen). The remaining clients have from 2 to 20 of these classes with extra samples from any other classes. We experimented with $\{600, 600, 500, 500\}$ on-demand tasks with randomly chosen label subsets $\mathcal{T} \subset \mathcal{L}$ per task and similarities ranging from 0.25 to 0.8 for {MNIST, Fashion-MNIST, EMNIST, CIFAR-100}, respectively.

DL Models & Baselines

For meta-models, fine-tuning models for classification tasks, and MixUp models, we use Convolutional Neural Networks (CNN) each with a varying number of training and fine-tuning rounds. We conducted a comprehensive fair comparison with the most relevant baselines in a decentralized FL and FML setting over the same random network topology: the decentralized FL (**DFedAvg**) [153] (extending the centralized FL [10]), Cluster-based DFedAvg (**C-DFedAvg**) [174] and Group-based FML (**G-FML**) [155]. The FML methods, CL-FML and G-FML, require fine-tuning for their meta-models over relatively small amount of data. To ensure fairness, CL-FML and G-FML are compared over the same portion $1 - \alpha$ for fine-tuning obtaining their variants CL-FML(α) and G-FML(α); $\alpha \in \{0.5, 0.6, 0.7\}$ as in [155].

4.6.2 Main Results

Labelled-Data Augmentation Performance

We assess CL-FML's distributed data augmentation performance handling missing labels across all tasks. MixUp generated augmented data conditioned on missing labels from $\mathcal{L}$. Our experiments, following the setup in [164], remained consistent across all datasets with variations tailored to each dataset's characteristics (e.g., output layer). Since CL-FML introduces fine-tuning stages over MixUp-augmented samples, Table 4.1 shows the impact of these samples on classification accuracy vs. percentages of missing labels, i.e., the percentage loss on the overall accuracy due to augmentation. CL-FML sacrifices on average (2.3%, 2.9%, 2.6%, 2.7%) of the achieved performance with MixUps dealing with distribution shift challenge for MNIST, Fashion-MNIST, EMNIST, CIFAR-100, indicating the applicability of data augmentation in label shifting.

Table 4.1: Impact of Labelled-data Augmentation on Model Performance

	(%)pct. Missing Labels	Accuracy	Loss
MNIST	33%	96.81%	-1.7%
	50%	97.11%	-1.6%
	60%	95.09%	-3.0%
	67%	94.42%	-3.2%
Fashion-MNIST	25%	92.11%	-0.8%
	50%	86.92%	-3.1%
	60%	87.36%	-3.5%
	67%	80.58%	-3.3%
EMNIST	33%	95.27%	-1.3%
	50%	90.41%	-3.2%
	60%	90.23%	-2.8%
	67%	87.65%	-3.1%
CIFAR-100	33%	72.00%	-2.73%
	50%	70.01%	-2.85%
	60%	73.78%	-2.45%
	70%	68.97%	-2.57%

Decentralized Federated Meta-Learning Performance

We study the convergence speed and accuracy of CL-FML and G-FML over all datasets with the same fine-tuning data access percentages $\alpha \in \{0.7, 0.6, 0.5\}$ having number of clusters $K = (3, 10, 16, 20)$ and probability threshold $\phi = 0.75$ for MNIST, EMNIST[2], MEDMNIST[3], Fashion-MNIST (F-MNIST)[4], CIFAR-100, respectively (K and ϕ are optimized per dataset). The results, depicted in Figure 4.2 and 4.3 (sampled two clusters; we obtain similar results for other clusters), showcase CL-FML's meta-model performance (top-1 accuracy) due to label-aware clustering along with notable reductions in the required number of training rounds. It is evidenced that CL-FML achieves accuracy significantly greater than G-FML's global meta-model. As shown in Figure 4.2(a)-(b) (MNIST; clusters C1 and C2), having $\alpha = 0.5$ (50% of data for meta-model training and 50% for fine-tuning besides augmented data), CL-FML requires around 3 rounds, whereas G-FML's global meta-model requires more than 15 rounds to plateau the accuracy. This indicates that by grouping clients together based on label distributions and building cluster-driven meta-models, we reduce the number of training rounds due to *reusing* these models for fine-tuning based on the most suitable data accessed per tasks. Moreover, the obtained accuracy of CL-FML's meta-models is significantly higher than that of G-FML's global meta-model across all datasets. CL-FML's accuracy is on average 13.7% higher than G-FML (MNIST) for $\alpha = 0.6$, where this gap increases to 60.67% for $\alpha = 0.7$. In CIFAR-100, as shown in Figure 4.3, CL-FML achieves on average 35% more accuracy compared to G-FML across all α ratios. G-FML's accuracy drops with more class imbalanced data as overfits over the associated labels yielding reduced generalization capacity. Whilst, CL-FML is sufficient and robust with α.

Overall, a meta-model trained on fewer labels (e.g., cluster-driven labels) has the ability to generalize better over those classes. On the other hand, a meta-model trained on a larger number

[2] https://www.kaggle.com/datasets/crawford/emnist/

[3] https://www.kaggle.com/datasets/andrewmvd/medical-mnist

[4] https://www.kaggle.com/datasets/zalando-research/fashionmnist

of labels (e.g., global meta-model) may have a broader generalization but potentially shallower understanding of any subset (or even super-set) of those classes. This results in less accuracy. This is what occurred in our results. We obtain similar patterns with the other two datasets and clusters. These improvements are attributed to building meta-models with specific range of labels. The label-aware clustering, which brings together clients with *less* label variability, improves the meta-learning model performances. However, addressing non-i.i.d. data and label imbalances becomes more challenging with increased label variety. These factors slowdown in the learning speed and performance of G-FML's global meta-models.

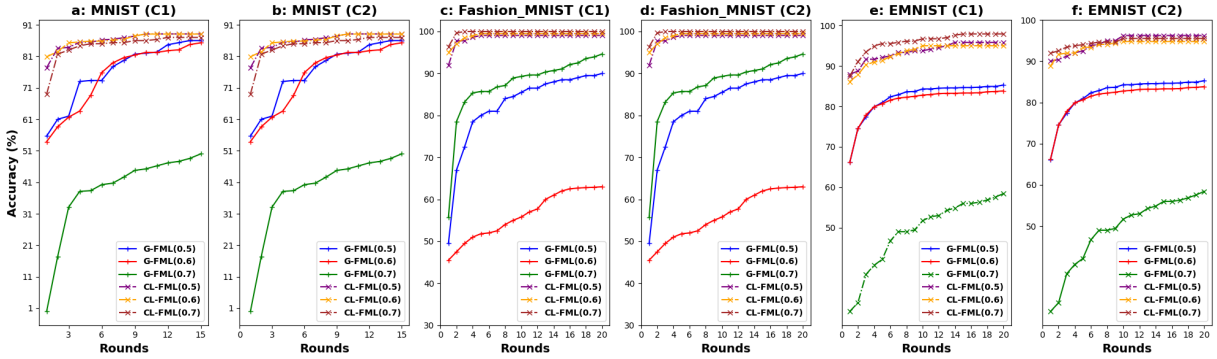


Figure 4.2: Multiple meta-models' top-1 accuracy (%) of CL-FML against global meta-model (G-FML) vs. convergence (samples of two groups).

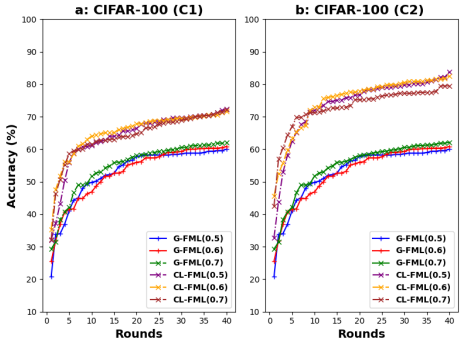


Figure 4.3: CL-FML against G-FML vs. convergence (samples of two groups).

Meta-model Fine-tuning Performance across Tasks

We further investigate the accuracy, required data access overhead, required amount of augmented data generated, and communication rounds during training and fine-tuning of all the methods over tasks with arbitrary label subsets. We experiment with a variety of tasks having similarities with labels $\mathcal{L}$ over different distributions of tasks per similarity. We notate $\mathcal{P}(\mathcal{T}|sim)$ the probability a task's label set $\mathcal{T}$ having similarity $sim(\mathcal{T},\mathcal{L})$ with available labels $\mathcal{L}$. Table 4.2 shows the training rounds for C-DFedAvg and DFedAvg, fine-tuning rounds for G-FML and CL-FML, along with required percentage of actual clients' data accessed, percentage of augmented data

Table 4.2: Comparison of Methods

MNIST								
Metric	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
Training Rounds	10	16	9	10	10	3	4	3
Actual Data Access (%)	100	100	30	40	50	30	40	50
Augmented Data Generation (%)	48.83	61.14	30.588	20.19	18.35	24.4	19.53	14.65
Accuracy (%) / F_1 score	96.80/0.96	94.39/0.93	95.19/0.94	93.12/0.93	94.84/0.94	96.63/0.96	96.45/0.96	96.36/0.97
Fashion-MNIST								
Metric	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
Training Rounds	15	20	10	10	9	6	5	6
Actual Data Access (%)	100	100	30	40	50	30	40	50
Augmented Data Generation (%)	38.46	47.82	14.34	19.12	23.91	11.53	15.38	19.23
Accuracy (%) / F_1 score	86.75/0.88	84.03/0.83	81.98/0.81	82.26/0.82	85.07/0.84	86.03/0.86	86.385/0.86	85.15/0.85
EMNIST								
Metric	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
Training Rounds	15	25	20	18	20	7	6	6
Real Data Access (%)	100	100	30	40	50	30	40	50
Augmented Data Generation (%)	33.185	64	14.34	19.12	23.91	11.53	15.38	19.23
Accuracy (%) / F_1 score	90.85/0.89	89.16/0.88	88.15/0.87	88.22/0.881	88.71/0.88	89.53/0.89	91.29/0.91	89.72/0.89
CIFAR-100								
Metric	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
Training Rounds	40	85	25	24	27	12	10	14
Real Data Access (%)	100	100	30	40	50	30	40	50
Augmented Data Generation (%)	36.78	68.2	24.33	27.73	29.83	18.67	22.34	24.57
Accuracy (%) / F_1 score	71.40/0.71	67.59/0.67	67.53/0.67	68.28/0.67	68.67/0.68	70.89/0.74	71.18/0.74	71.15/0.73

generated, classification accuracy and F_1 score [120] over all tasks. C-DFedAvg and DFedAvg require to train models *from scratch* for new tasks, since no meta-model is (re)used, while G-FML and CL-FML fine-tune their re-usable meta-models per task. CL-FML significantly reduces fine-tuning rounds and percentage of required actual data accessed for fine-tuning compared to all baselines, while achieving competitive performance.

In MNIST, CL-FML (α = 0.5) requires 56% fewer rounds compared to C-DFedAvg; such reduction is attributed to well-initialized meta-models yielding to 60% less augmented data to be generated. Notably, CL-FML utilizes augmented data exclusively for fine-tuning, resulting in nearly identical accuracy to C-DFedAvg with almost negligible sacrifice (0.34%) of accuracy. Similarly to DFedAvg, CL-FML demonstrates a substantial reduction (77%) in number of rounds. This reduction is attributed to the inherent challenges of random client selection in DFedAvg, which potentially results in choosing clients less suitable per task. Such mismatch leads to increased augmented data generation affecting convergence rate. Conversely, CL-FML selectively engages the most suitable clients per task due to label-aware clustering. Furthermore, CL-FLM achieves a remarkable decrease in communication rounds by (60%, 65%, 60%) compared to G-FML(α =0.5/0.6/0.7), respectively. This is reflected by the fast convergence rate obtained by CL-FML, which fine-tunes meta-models tailored to clients' clusters more *similar* to tasks' labels as opposed to G-FML. The latter does not explicitly focus on tasks' label distribution for building its meta-model. Regarding the amount of augmented data required, CL-FML achieves a significant reduction compared to C-DFedAvg (68.07% reduction), DFedAvg (68.07% reduction) and G-FML (15.28% reduction) for the reasons we mentioned above. One can observe similar average behavior of the methods in the other datasets. Notably, in Fashion-MNIST, CL-FML reduced the required rounds by 3 times compared to C-DFedAvg and DFedAvg, and 2 times

less compared to all variants of G-FML. Such improvements were achieved without sacrificing model accuracy as shown in Table 4.3. In terms of accuracy compared to C-DFedAvg, CL-FML incurred a minimal sacrifice of 0.63%, however, achieving communication efficiency due to less rounds.

In CIFAR-100 (see Table 4.2), CL-FML requires 4 to 7 times less rounds compared to C-DFedAvg and half rounds compared to G-FML on average, sacrificing only 0.44% (0.3%) of accuracy (Table 4.3) w.r.t. C-DFedAvg with $\alpha = 0.5(0.7)$, while being more accurate for DFedAvg and G-FML. Table 4.3 shows the break-down performance of each method based on tasks' similarity distribution and the associated probabilities $\mathcal{P}(\mathcal{T}|sim)$. CL-FML (mostly for $\alpha = 0.6$) outperforms the other methods across tasks similarities showcasing robust and efficient behaviour in tackling label shifting. C-DFedAvg and DFedAvg achieve 0.5% and 1.16% more accuracy for 67% similarity (MNIST) and 30% similarity (EMNIST), respectively, however, at the expense of almost 3 times most training rounds and 50% more augmented data.

Table 4.3: Impact of Overlapping/Similarity between Tasks & Clusters on Fine-tuned Models Performance.

MNIST									
$\mathcal{P}(\mathcal{T}\|sim)$	**Similarity**	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
20%	80%	98.96%	98.65%	99.29%	99.10	99.34%	98.96%	**99.55%**	99.24%
30%	67%	**97.11**%	96.074	95.48%	95.88%	96.07%	96.01%	96.61%	96.81%
30%	50%	96.91%	96.56	97.13%	96.73%	96.91%	**97.86%**	97.51%	97.11%
20%	30%	97.01%	96.01	95.56%	96.55%	96.48%	97.01%	**97.34%**	97.23%
Fashion-MNIST									
$\mathcal{P}(\mathcal{T}\|sim)$	**Similarity**	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
20%	80%	91.96%	91.14%	87.80%	88.89%	90.03%	89.40%	**92.11%**	90.79%
30%	70%	90.04%	85.15%	86.13%	87.96%	88.77%	87.52%	89.36%	**90.94%**
30%	50%	86.48%	77.71%	71.18%	72.22%	83.52%	86.80%	**86.92%**	86.17%
20%	25%	79.78%	75.56%	75.77%	79.65%	79.55%	**80.58%**	79.55%	78.94%
EMNIST									
$\mathcal{P}(\mathcal{T}\|sim)$	**Similarity**	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
20%	80%	93.27%	89.38%	92.21%	93.25%	92.83%	**93.81%**	92.15%	93.35%
30%	70%	90.18%	90.41%	91.35%	88.80%	88.50%	88.59%	**91.71%**	89.61%
30%	50%	89.25%	89.20%	85.70 %	86.75%	87.33%	86.34%	86.47%	**89.56%**
20%	30%	79.64%	**87.65%**	83.34%	83.84%	85.19%	86.38%	84.84%	83.84%
CIFAR-100									
$\mathcal{P}(\mathcal{T}\|sim)$	**Similarity**	**C-DFedAvg**	**DFedAvg**	**G-FML(0.7)**	**G-FML(0.6)**	**G-FML(0.5)**	**CL-FML(0.7)**	**CL-FML(0.6)**	**CL-FML(0.5)**
20%	80%	**72.84%**	65.60%	66.40%	67.60%	67.77%	71.20%	72.00%	72.82%
30%	70%	69.92%	67.60%	68.20%	67.86%	68.50%	69.54%	70.01%	**70.07%**
30%	50%	**74.84%**	71.13%	68.91 %	70.38%	70.67%	74.56%	73.78%	73.09%
20%	30%	68.43%	66.05%	66.63%	67.31%	67.77%	68.28%	**68.95**%	68.54%

Impact Label Imbalance on Meta-model Performance

Beyond evaluating the effectiveness of multiple meta-learning models of CL-FML in terms of accuracy and convergence speed for both the meta-models and (task-tailored) fine-tuned models, it is crucial to explore their potential in mitigating the significant impact of overall label imbalance. In cluster-based FL, where clients are clustered to address scalability and label shifting, variations in the number of labels per client pose a challenge. Traditional techniques like over-

sampling or under-sampling are not readily applicable in our context due to non-exchangeable and non-communicable data. To address such challenge, CL-FML employs multiple meta-models, leveraging the synergies of client clustering and re-balancing of classes within each cluster. The aim is to improve the overall meta-model performance compared to scenarios with a single global meta-model suffering from high or low label imbalance. CL-FML involves filtering out classes beyond the cluster distribution range for each client and incorporating augmented data generated by local MixUp models. The latter have been collaboratively fine-tuned among the most suitable clusters of clients per tasks.

Our findings presented in Table 4.4 reveal a substantial improvement in meta-model performance. When comparing Multiple Meta-models (**CL-FML/MM**) with One Global Meta-model with High Imbalance (**OG-MM-HI**) and One Global Meta-model with Low Imbalance (**OG-MM-LI**), we observe a performance boost of 54% and 4% respectively in MNIST. Similar trends are observed with the three other datasets with the highest improvement obtained in CIFAR-100. CL-FML employing multiple meta-models, which are fine-tuned over the most suitable clusters per tasks, effectively addresses the challenges posed by significant label imbalances across clients. By strategically combining label-aware clustering, re-balancing, and augmented data, CL-FML achieves notable enhancements in meta-models' performance across various datasets, which yields such meta-models reusable for future ad-hoc tasks. Overall, CL-FML evidenced

Table 4.4: Impact the Label Imbalance on Meta-model Accuracy

Dataset	**OG-MM-HI**	**OG-MM-LI**	**CL-FML/MM**
MNIST	63.21%	94.55%	98.45%
Fashion-MNIST	59.67%	95.77 %	98.02%
EMNIST	47.32%	75.37%	93.22 %
CIFAR-100	36.29%	70.41%	83.79%

efficiency and effectiveness in tackling on-demand tasks with arbitrary label distribution, mainly attributable to label-aware clustering that helps identifying the most suitable clients per task, and pre-trained distributed cluster-based meta-models that facilitate the fine-tuning of the final classifier over augmented data.

4.7 Conclusions

This chapter introduced the CL-FML framework, designed to address the challenges of classification tasks with label shifting and data heterogeneity in federated learning systems. CL-FML employs a decentralized federated meta-learning approach through a label-driven client clustering mechanism, grouping clients based on their label distributions. A key advantage of CL-FML is its ability to handle arbitrary classification tasks by utilizing multiple cluster-based meta-learning models. These models are specifically tailored to each client cluster, ensuring that the

models align with the distribution of labels within their respective clusters. This design allows the framework to generalize effectively across varying data distributions, enhancing classification accuracy and robustness, especially in scenarios involving label shifting and imbalanced data. Furthermore, CL-FML integrates advanced data augmentation techniques to overcome the limitations posed by scarce labeled data. By augmenting the available labeled data, the framework significantly improves classifier performance, even when only a small portion of the data is labeled. Extensive experimental evaluations against strong baselines have demonstrated the superiority of the CL-FML framework. The results highlight its effectiveness in achieving high classification accuracy, particularly in settings characterized by label distribution shifts and client data heterogeneity. CL-FML consistently outperforms traditional federated learning methods, showcasing the importance of its label-aware clustering and data augmentation strategies in handling the complexities of distributed machine learning.

4.8 Limitations & Directions of Enhancement

The CL-FML framework demonstrates significant improvements in model performance and convergence speed. However, it assumes that all client data is labeled, which is often challenging in practice. For instance, clients may lack motivation to label their data due to costs, time constraints, or insufficient expertise. These factors collectively contribute to the scarcity of labeled data. Additionally, the mechanism becomes more practical when clients can share their label distributions. Nevertheless, this is not a critical issue, as label distributions can be predicted using various methods, such as evaluating local model performance.

In our future work, we aim to address more dynamic scenarios where the majority of client data is unlabeled, and the server has no data. This is the direction we plan to explore in the next chapter.

Chapter 5

The Price of Labelling: A Two-Phase Federated Self-Learning Approach

5.1 Introduction

As discussed in previous chapters, FL is a privacy-preserving collaborative learning paradigm that eliminates the need to transfer client data. While most existing studies on FL primarily focus on supervised learning, assuming that all clients possess sufficient training data with ground-truth labels, this assumption may not always hold in practical scenarios. In many cases, a key challenge in FL is ensuring the *quality* of labels and effectively handling *unlabelled* data. These challenges arise due to various factors such as limited resources, labeling costs, human errors, or issues with data collection mechanisms [175]. The labeling challenges inherent in FL impede the development of robust models, especially in the presence of unlabelled data, which is not rare in realistic scenarios. This predicament often leads to models with reduced generalization capabilities [9].

Self-learning in FL environments (SFL) has been introduced to tackle some of these challenges [176]. SFL utilizes labelled data to train a global model, which self-learns using pseudo-labels of the unlabelled data or refines the noisy labelled data [177, 178]. However, SFL operates under the assumption that data are IID. In FL, data are often non-IID indicating a distribution shift between clients [165]. Moreover, even within labelled data, label imbalance can be encountered, where several clients have significant disparities between majority and minority classes. Such imbalances impede the convergence and degrade performance of the model [120, 9]. Model performance heavily relies on the quality and distribution of the training data. The high degree of heterogeneity among client data significantly decreases model performance, while achieving data homogeneity tends to improve it. This observation underscores the importance of addressing issues related to data distribution, such as skewness and non-IID characteristics.

Various SFL approaches in the literature can be classified into configuration-centric and data-centric. The former approaches rely on model importance weighting to address data

imbalance, e.g., FedNOVA [167], FedProx [138]. However, such approaches heavily depend on numerous communication rounds and may require sharing private information to enhance the overall performance of the global model. Data-centric approaches focus on improving the global model primarily by incorporating augmented data to increase the volume of labelled data [138, 152]. Our work falls under the data-centric approach significantly differing from previous data-centric approaches. Previous work has focused on 'perfect setups', which proves challenging to implement in real-world settings. Such setups hinge on several key unrealistic assumptions [152, 179]. **A1:** A subset of clients or the server itself has adequate labelled samples to effectively train supervised models, ensuring their generalization across unlabelled clients [152]. **A2:** The model can generate high-quality pseudo-labels by considering only labelled data during the FL process, as evidenced in e.g., teacher-student methods [178]. This means that clients with unlabelled data cannot be engaged in the FL process. These disparities between ideal and realistic scenarios prompt us to contemplate the following question: *What is the price of learning a global model using scarce and skewly distributed labelled data, while capitalizing on partially labelled and fully unlabelled data across clients?* Addressing the above problem becomes challenging in FL as data cannot be exchanged between clients due to privacy concerns.

This chapter introduces the two-Phase Federated self-Learning framework, coined **2PFL**, that addresses both extreme data scarcity and skewness in training classifiers over distributed labelled and unlabelled data. 2PFL demonstrates the ability to achieve high-performance models when trained with only 10% to 20% labelled data compared to the unlabelled data. 2PFL consists of two key components: data augmentation and progressive self-learning. We address data skewness through data augmentation, as an efficient mechanism for rectifying distribution skewness and enhancing data diversity [165, 138]. Moreover, 2PFL undergoes training using a two-phase self-learning. In the initial phase, 2PFL trains a model using clients' labelled data to pseudo-label unlabelled data. This is followed by training in the subsequent phase, incorporating both labelled and pseudo-labelled data.

The primary contributions of this chapter are outlined as follows: :

- The 2PFL framework is proposed as an all-inclusive approach that engages clients with labeled, partially labeled, and unlabeled heterogeneous data in distributed learning.
- Mechanisms are introduced to address data scarcity, enhance data augmentation, and implement two-phase pseudo-labeling of unlabeled data, guided by a confidence threshold schedule.
- Comprehensive experiments and a comparative assessment of 2PFL against baselines [180, 181, 10] and the ideal method are provided, demonstrating the effectiveness and efficiency of 2PFL in an all-inclusive FL ecosystem.

5.2 Related Work

SFL is a mechanism for training high-performing models with a limited amount of labeled data along with skewed and biased label distribution [182]. SFL in FL environments is proved powerful for collaboratively training models over distributed data with label deficiency. In supervised learning with fully labeled data, non-IID data result in biased models [183, 184]. Several studies [185, 186, 187] focused on re-balancing data distribution and reducing the underlying data heterogeneity. Eventually, this becomes more problematic in federated semi-supervised and self-supervised learning, as labelled data are limited and skewed [188]. This apparently leads to biased predictions of pseudo-labels for the unlabeled data during training [189]. Notably, the model is fed with wrong pseudo-labelled data yielding to decreasing model accuracy, slowing down convergence speed and reducing generalization capacity. The SFL approach in [190] focuses only on unlabelled data, which leads to a discrepancy between the objective functions for labelled and unlabelled data, resulting in gradient inconsistencies. To address this challenge and accelerate the training process while reducing the number of training rounds, a few SFL studies are based on pre-trained models. In [191, 192, 193], pre-trained models are utilized as initialization for FL rounds avoiding random initialization of model parameters. The principle objective of these approaches is to first train a *teacher* model using available labelled data and then use this model to make predictions for unlabelled samples. Subsequently, the *student* model is trained on both labelled and predicted (pseudo-labelled) samples. However, such approaches assume that the pre-trained model is well-fitted to the available IID data with no class imbalance issues. In addition, [191] utilized data augmentation (e.g., random flipping and cropping) to perform weak augmentation, thereby enhancing the size and quality of server's centralized data. In centralized learning, it is relatively straightforward to address data distribution issues adopting simple data augmentation techniques. In distributed learning, like in our case, we encounter non-IID data which is not rare in realistic scenarios. This poses challenges to global model fitting. 2PFL showcases that the global model struggles to fit well with scattered and skewed data.

To address these challenges, there are two research directions. The first direction deals with training a global model over centralized labelled data and using this model for distributed unlabelled data pseudo-labeling. This direction assumes that *all* the classes are available on the FL central server with potential class imbalances. To improve data distribution, [194, 175] approaches adopted data augmentation. Moreover, in cases where there are only limited labelled data and/or there are non-IID data on the central server, approaches like server [175, 195, 196] adopt strategies like oversampling, under-sampling, rotation, or cropping to mitigate these issues. The method in [197] used prototypes of labelled server's data for each class, where these prototypes are sent to clients for assigning pseudo-labels to unlabelled data. The second direction deals with the fact that both labelled and unlabelled data are distributed across clients. Several approaches in this direction assume possibility of exchanging data and/or statistical information

among clients, e.g., [198], while other studies are based on sharing locally augmented data across clients, e.g., in[184, 138]. Moreover, several studies in this direction like [199, 181] consider the case that some clients possess fully labelled data while the remaining clients have entirely unlabelled data. In this context, such approaches assume that the labelled and unlabelled data do not exhibit distribution shifts, which significantly simplifies the learning process. The approaches [180, 196, 200] falling into this direction assume that each client has partially labelled data such that labelled parts of the data have the same distribution as that of the unlabeled parts of the data. This assumption does not entirely reflect realistic cases either.

The fundamental differences of 2PFL with the relevant approaches are: 2PFL copes with the cases where there is no centralized server with available labelled data. Moreover, in 2PFL, clients can possess labelled non-IID data, partially labelled data with missing classes of unknown distribution and potential class imbalances, and clients with fully unlabeled data. In all these cases, 2PFL considers any arbitrary distributions from labelled to unlabelled data, thus, avoiding any assumptions of having same distributions and class imbalance ratios across the clients. 2PFL aims to reflect real-word scenarios in FL ecosystems w.r.t. clients' labeling nature of data. We introduce mechanisms to achieve data balancing during model training through data augmentation and a two-phase pseudo-labelling. Therefore, 2PFL improves the performance of the global model over non-IID data by incrementally pseudo-labelling and controlably augmenting data, which is significantly comparable with the ideal case, i.e., where there are only fully labelled IID data without label shifts across *all* the clients [181].

5.3 Overview & Fundamentals

Consider a set $\mathcal{N} = \{n_1, \ldots, n_N\}$ of distributed clients participating in a FL process by communicating with a central (parameter) server. Each client $n_i \in \mathcal{N}$ possesses a dataset $\mathcal{D}_i$ containing $C = \{0, \ldots, C-1\}$ classes (labels) of data, which can be labelled and/or unlabelled in any proportion. We classify the clients into three types based on their data:

- **Type I** clients (*labelled* clients) $n_i \in \mathcal{N}^L \subset \mathcal{N}$ are characterized by having *fully* labelled data denoted as $\mathcal{D}_i^L = \{(x_k, y_k)\}_{k=1}^{|\mathcal{D}_i^L|}$, where x_k is input and y_k is the corresponding label from C.

- **Type II** clients (*partially labelled* clients) $n_i \in \mathcal{N}^P \subset \mathcal{N}$ have dataset split into labelled and unlabelled samples, i.e., $\mathcal{D}_i^P = \{(x_k, y_k \vee \perp)\}_{k=1}^{|\mathcal{D}_i^P|}$, with $y_k \in C$ and $\perp$ representing unlabelled data.

- **Type III** clients (*unlabelled* clients) $n_i \in \mathcal{N}^U \subset \mathcal{N}$ have all samples unlabelled, i.e., $\mathcal{D}_i^U = \{(x_k, \perp)\}_{k=1}^{|\mathcal{D}_i^U|}$. We focus on cases where the labelled samples are much fewer than unlabelled ones, i.e., $|\mathcal{D}^L| \ll |\mathcal{D}^U|$, as stated in [196].

2PFL consists of the following phases (see Figure 5.1). **Phase 1** engages only clients of Types I and II in FL. The global model of Phase 1, $\theta_G^{(1)}$, is trained on labelled and augmented data from these clients for T_1 rounds. In turn, $\theta_G^{(1)}$ is used to pseudo-label clients' partially labelled data (Type II clients). **Phase 2** updates $\theta_G^{(1)}$ using high confidence pseudo-labelled samples engaging Type III clients in FL. The Phase 2 global model $\theta_G^{(2)}$ is then derived (after T_2 rounds) and used to re-label clients' data, while entering the **Phase 2+**. In this final phase, we progressively train and refine $\theta_G^{(2)}$ until all the high-confidence unlabelled samples across all clients are pseudo-labelled (after T_{2+} rounds). We then obtain the refined final model $\theta_G^{(2+)}$.

Remark 1: The amount of pseudo-labelled samples can be different based on the amount of mismatching between labelled and unlabelled samples. For example, if the mismatching rate is low, $\theta_G^{(1)}$ can pseudo-label most of the samples in the first phase. While, if the mismatching rate is high, it is expected $\theta_G^{(2)}$ and/or $\theta_G^{(2+)}$ to label many samples in Phases 2 and/or 2+, after been trained on pseudo-labelled samples (which are better generalized compared to unlabelled ones). In real-world settings, the local data distribution varies across clients [196]. Based on the non-IID data nature and types of clients, our mechanism determines the phase in which each client participates and selects augmentation techniques to generate a specific amount of data.

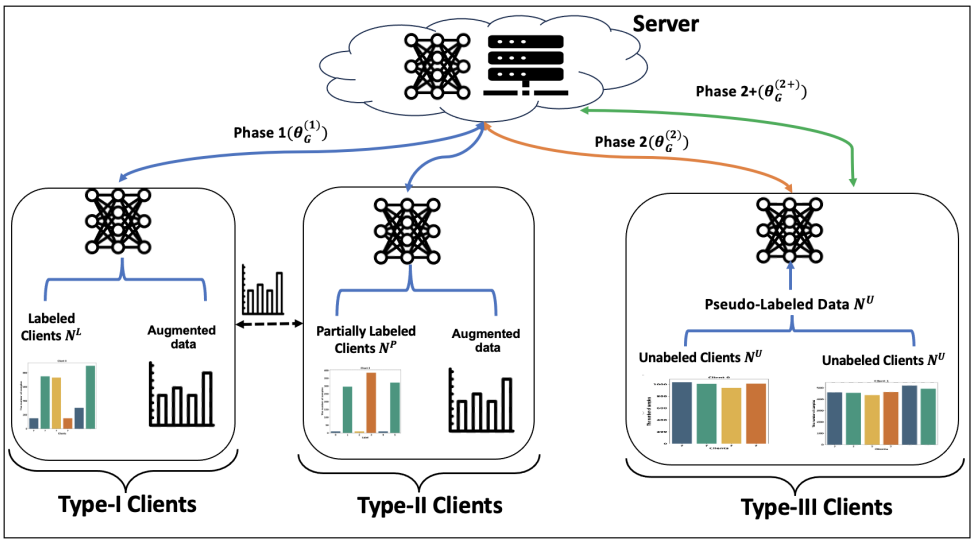


Figure 5.1: The Phases 1, 2 and 2+ of the 2PF framework progressively engaging labelled, partially-labelled and unlabelled clients in distributed self-learning.

In partially and fully labelled clients, the set of classes of labelled data $\mathcal{D}^L$ may not be the same as that in the unlabelled data $\mathcal{D}^U$. This indicates that the set of classes observed in each fully or partially labelled client may not be identical to the set of classes in the unlabelled data over clients. However, there might be an overlap between classes in labelled or partially labelled clients and classes in unlabelled clients. Moreover, there might be *unseen* classes within each labelled client's data (outside of C). Consequently, we neither train a model on unseen classes nor use it to pseudo-label samples from $\mathcal{D}^P$ or $\mathcal{D}^U$.

During each training round in Phases 2 and 2+, the global model generates pseudo-labels for

unlabelled samples over unlabelled clients. It is expected that several of these pseudo-labels may have low confidence (*falsely inferred pseudo-labels*), thus, leading to potential degradation of the model performance while being trained also with such samples. To mitigate this issue, our mechanism considers the pseudo-labels for training only when the global model assigns a very high probability to the inferred class w.r.t. a dynamic confidence threshold. Moreover, regarding data augmentation, mechanisms may require data sharing between clients. However, due to privacy concerns and limitations on communication resources, data sharing is not feasible, thus, clients are restricted to share only synthesized samples conditioned on specific labels [138].

5.4 The 2-Phase Federated Self-Learning Framework

5.4.1 Local Data Augmentation

Given the non-IID and heterogeneous data nature, it is common for the class distribution to be highly imbalanced. To address this issue, 2PFL adopts MixUp [152] to locally augment data over labelled and unlabelled clients. Note, any other data augmentation mechanism could also be adopted [201]. In labelled/partially labelled client $n_i \in \mathcal{N}^L \cup \mathcal{N}^P$, for any two inputs x_k and x_ℓ with labels y_k and y_ℓ, MixUp synthesizes the sample (x', y'):

$$x' = \lambda x_k + (1-\lambda)x_\ell \text{ and } y' = \lambda y_k + (1-\lambda)y_\ell \tag{5.1}$$

with $\lambda \in (0, 1)$, a blending parameter controlling interpolation between samples. In unlabelled client $n_i \in \mathcal{N}^U$, two randomly selected pseudo-labelled inputs x_k and x_ℓ with high-confidence pseudo-labels $\widehat{y}_k$ and $\widehat{y}_\ell$, respectively, generate the sample (x', y'):

$$x' = \lambda x_k + (1-\lambda)x_\ell \text{ and } y' = \lambda \widehat{y}_k + (1-\lambda)\widehat{y}_\ell. \tag{5.2}$$

5.4.2 2PFL Training Phases

2PFL exploits labelled, partially labelled and unlabelled data across all types of clients $(\mathcal{D}_i^L \cup \mathcal{D}_i^P \cup \mathcal{D}_i^U)_{n_i \in \mathcal{N}}$ to minimize the loss function $f(\theta_G) = f^L(\theta_G) + f^P(\theta_G) + f^U(\theta_G)$, where $f^L(\theta_G)$, $f^P(\theta_G)$, and $f^U(\theta_G)$ is the loss over labelled, partially labelled and unlabelled clients, respectively:

$$\begin{aligned}\min_{\theta_G} f(\theta_G) = \frac{1}{N^L}\sum_{\ell=1}^{N^L}\mathcal{L}^L(x_\ell^L, y_\ell^L; \theta_G) + \frac{1}{N^P}\sum_{p=1}^{N^P}\mathcal{L}^P(x_p^P, \hat{y}_p^P; \theta_G) \\ + \frac{1}{N^U}\sum_{u=1}^{N^U}\mathcal{L}^U(x_u^U, \hat{y}_u^U; \theta_G)\end{aligned} \tag{5.3}$$

and $\mathcal{L}$ is task-specific loss function on clients with labelled, partial labelled and unlabelled data. Therefore, we present the following phases of 2PFL.

Phase 1: Engagement of Labelled & Partially Labelled Clients: Phase 1 trains a global pseudo-labeling model $\theta_G^{(1)}$ from decentralized labelled and partially labelled clients $n_i \in \mathcal{N}^L \cup \mathcal{N}^P$. 2PFL firstly trains the model on labelled clients $\mathcal{N}^L$ using the *ground-truth* labels optimizing the loss [180]:

$$f(\theta_G^{(1)}) = \min \left[\frac{1}{N^L} \sum_{\ell=1}^{N^L} \mathcal{L}_{CE}(g(x_\ell; \theta_G^{(1)}), y_\ell) \right], \tag{5.4}$$

where $\mathcal{L}_{CE}$ is cross-entropy loss and $g(\cdot;\cdot)$ represents the classifier, e.g., convolutional neural network (CNN). In Phase 1, at the beginning of each round $t \leq T_1$, the global parameters $\theta_G^{(1)}$ are disseminated to each labelled client n_i locally updating over E local epochs:

$$\theta_i^{t,e+1} = \theta_i^{t,e} - \eta_t \nabla f_t(\theta_i^{t,e}), \quad e = 1, \ldots, E, \tag{5.5}$$

minimizing the cross-entropy loss. Upon completion of E local epochs, each labelled client $n_i \in \mathcal{N}^L$ sends its local model $\theta_i^{t,E}$ to the server for aggregation for round $t = 1, \ldots, T_1$:

$$\theta_{G,t}^{(1)} = \frac{1}{|\mathcal{N}^L|} \sum_{n_i \in \mathcal{N}^L} \theta_i^{t,E}. \tag{5.6}$$

At round $t > 1$, $\theta_{G,t}^{(1)}$ is distributed to each partially labelled client $n_i \in \mathcal{N}^P$ to be used for pseudo-labeling of partially labelled samples in the subsequent training rounds.

Each unlabelled client n_i uses $\theta_{G,t}$ to predict the label $\hat{y}_u$ for the unlabelled input x_u based on previous knowledge captured from previous rounds $\tau < t$. The labeling mechanism selects the class $c \in C$ with maximum predicted confidence from $\theta_{G,t}$, since the prediction will be used to pseudo-label x_u [196]. However, the classification accuracy of pseudo-labels can be low for some classes, especially, in cases of where existing unlabelled samples have not been trained sufficiently during the supervised phases over labelled clients $\mathcal{N}^L$. After several rounds $\tau = 1, \ldots, t$ of 'warm-up' model training, 2PFL from round $t > \tau$ and onwards produces pseudo labels for unlabelled samples, which will be as inputs to following rounds $t' > t$ (as fully labelled samples). That is $\theta_{G,t'}$ uses predictions from previous rounds $t < t'$ as target classes for past unlabelled samples as if they were true labels. The process goes through until all unlabelled samples are labelled. However, the predicted pseudo-labels can be noisy or may be possibly wrong [202]. By progressively training the global model involving pseudo-labels leads to 'forgetting' the knowledge acquired from labelled clients, a.k.a. the catastrophic forgetting problem in CNNs [200]. Therefore, to mitigate the potentially destabilizing effect of noisy labelled samples being used during progressive training, 2PFL considers only pseudo-labelled samples with high certainty (high confidence), while filtering out samples with low confidence. In this context, the pseudo-label for x_u is $\hat{y}_u = c$, with $c = \arg\max_{c' \in C} p_{\theta_{G,t}}(c'|x_u)$ i.e., the label

with maximum predicted confidence.

In parallel to Phase 1, labelled clients can tackle the class imbalance problem by producing the required augmented samples discussed in Section 5.4.1 over their ground truth labels. Once the partially labelled clients have used $\theta_G^{(t)}$ to pseudo-label their unlabelled samples, they can augment their pseudo-labelled data as well to deal with potential class imbalance issues.

Phases 2 & 2+: Engagement of Unlabelled Clients & Fine-tuning: After Phase 1, the unlabelled clients along with the rest of clients are engaged in Phase 2 to enhance the robustness of the global $\theta_G^{(2)}$. This is achieved by progressively incorporating pseudo-labelled samples with high confidence obtained from previous rounds into the subsequent rounds with $t = T_1, \ldots, T_2$. This allows the global model to generate increasingly high quality pseudo-labels for unlabelled samples in unlabelled clients. Since the amount of labelled samples on the labelled and partially labelled clients is limited and less diverse, incorporating pseudo-labelled samples helps bridging this gap.

In Phase 2, 2PFL incrementally involves all the previously pseudo-labelled samples x_u with a high-confidence pseudo-labeling accuracy given a dynamic probability threshold $\phi \in (0.5, 0.9)$ such that $\hat{y}_u = c$ with $c = \arg\max_{c' \in C} p_{\theta_{G,t}^{(2)}}(c'|x_u)$ and $p_{\theta_{G,t}^{(2)}}(c'|x_u) \geq \phi$ in high hopes of being as close to ground truth as possible [175].

The objective of teacher-student FSL is to train a teacher model, which supervises the learning process of a student model that learns from labelled and unlabelled data jointly. First, a teacher model is built with the available labelled data and afterwards this is exploited to make predictions for unlabelled samples. Subsequently, the student model is trained on both labelled and predicted samples. In Phase 2, 2PFL relies on the confidence threshold ϕ to learn from unlabelled data residing on several clients, thus, boosting the performance of models trained in FL with varying percentages of labelled samples. To learn from unlabelled data, 2PFL generates a pseudo-label $\hat{y}_u$ for each available unlabeled data x_u on a client n_i:

$$\hat{y}_u = \arg\max_{c \in C} \frac{e^{z_c/L}}{\sum_{m \in C} e^{z_m/L}}, \tag{5.7}$$

where z_c's are the logits produced for unlabelled input x_u by the client n_i's model $g_i^{(2)}$ of Phase 2 with parameter $\theta_i^{(2)}$ before the softmax layer. 2PFL produces labels for the given 'soften' softmax values (L is a constant scalar). As the maximum of the softmax function remains unaltered, the predicted pseudo-label $\hat{y}_u$ is identical as if the original prediction (without scaling) for an unlabelled sample x_u was used; however, the prediction confidence is weakened. The dynamic threshold ϕ of confidence is proposed to follow a cosine learning schedule to discard low-confidence predictions when generating pseudo-labels. For the obtained pseudo-labels, 2PFL performs standard cross-entropy minimization while using $\hat{y}_u$ as targets as follows:

$$-\frac{1}{N^U} \sum_{u=1}^{N^U} \sum_{c \in C} \hat{y}_u \log(g_i(x_u; \theta_i^{(2)}), \hat{y}_u) = \mathcal{L}_{CE}(\hat{y}_u; p_{\theta_i^{(2)}}(x_u)). \tag{5.8}$$

The local model's loss function on client n_i on the t-th round in Phase 2 is:

$$\mathcal{L}_{CE}(y_\ell; p_{\theta_i^{(2)}}(y_\ell|x_\ell)) + \beta\mathcal{L}_{CE}(\hat{y}_u; p_{\theta_i^{(2)}}(\hat{y}_u|x_u)). \tag{5.9}$$

where $\beta \in (0, 1)$ weighs the effect of unlabeled samples on the training.

Remark 2: The involvement of highly-confident pseudo-labelled samples in 2PFL results in potential revising the classification probability of previously pseudo-labelled samples across training rounds. The less the global model training loss with training rounds becomes, the higher the possibility the classification probability of the best class of a pseudo-labelled sample turns, thus, more highly confident samples are incorporated in the training. Hence, 2PFL obtains a fine-tuned global model $\theta_{G,t}^{(2+)}$ at 'extra' rounds $t = T_2, \ldots, T_{2+}$. This is the Phase 2+ in our mechanism. It is important the global model to be trained with the pseudo-labelled samples as well, which enable the model to be fine-tuned. In this way, the server can provide a comparable or even better model (in terms of classification accuracy) than that of previous rounds, which is used by active clients in future rounds to pseudo-label.

5.5 Experimental Evaluation

5.5.1 Experimental Set-up

Datasets, Data Distribution & System Setting:

We evaluate 2PFL using image classification datasets MNIST, EMNIST[1], MEDMNIST[2], and Fashion-MNIST (F-MNIST)[3] with number of classes $|C| = (10, 47, 6, 10)$ respectively. We follow the approach in [181] to determine the distribution of data across clients by adopting the Dirichlet(γ) distribution to generate non-IID data across clients with $\gamma = 0.8$. In this context, the number of samples per class differs from one client to another. We use different number of clients $|\mathcal{N}| \in \{10, 20, 50\}$ over datasets and split the clients into Types I, II and III based on the ratio 2:3:5, respectively, i.e., 20% of clients have fully labelled samples, while 30% and 50% of clients have partially labelled and unlabelled samples, respectively. For Type II clients, we define the probability of missing class per sample $q = 0.3$, per class $c \in C$ (for Type III clients, $q = 1$ for each class). We set $\beta = 0.5$ as in [193] to control the effect of the unlabelled data in Phase 2/2+. The confidence threshold ϕ is initially set to 0.5 and gradually increases 0.9 during training w.r.t. a cosine schedule. We set $\lambda = 0.2$ for data augmentation as in [152]. For assessing the classification test accuracy, for each dataset, we use a *separate* subset of labelled samples (30%), which is not involved in *any* training phase.

[1] `https://www.kaggle.com/datasets/crawford/emnist/`
[2] `https://www.kaggle.com/datasets/andrewmvd/medical-mnist`
[3] `https://www.kaggle.com/datasets/zalando-research/fashionmnist`

Baselines:

To fairly validate **2PFL**, we compare against state-of-the-art FL methods and the *ideal* (ground-truth) theoretical benchmark in FL to understand the upper bound of performance. The **Ideal** FL benchmark considers the case where all clients have fully labelled samples without class imbalance and label shifted distributions. We simulated the 'Ideal' benchmark using FedAvg [10]. Our aim is to investigate how 2PFL compares with Ideal's performance in terms of accuracy and total training rounds under the context of having not only fully labelled clients, but also clients with partially labelled and unlabelled data with class imbalances. This chapter showcases the effectiveness of progressing pseudo-labelling and data augmentation including any type of clients under realistic scenarios compared to the ideal ones. We also compare against the baseline **FedAvg** [10], which is applicable only over clients with labelled data (Type I). FedAvg cannot engage Types II and III clients for training since it requires only labelled data to train the global model. In this context, we study the impact of lack of knowledge due to not involving Type II/III clients on the classifier predictive capacity. We showcase the advantage of progressive pseudo-labeling in an *all-inclusive* FL environment achieved by 2PFL. We further compare against **PL-FL** [180], which engages only Type II clients. We study the case where all clients have partially labelled samples with class imbalances over data. PF-FL will unveil the impact of the data augmentation on our self-learning context where several classes are missing across clients. Finally, we compare against **L&PL-FL** [181], which involves Type I and II clients with class imbalances as a more elaborate realistic scenario investigating pseudo-labeling influence. **Classifier model:** We use a CNN consisting of three convolutional layers followed by max-pooling, flattening, and two fully connected layers. The convolutional layers use ReLU consisting of 32, 64, and 128 filters. The final layer consists of classes neurons along with softmax activation.

5.5.2 Experimental Results

Impact of pseudo-labeling confidence on training phases.

We prioritize a high confidence threshold due to the challenges posed by non-IID and skewed data distributions. After training the model on labelled data (Phase1), we observe a decreased performance, especially in classes with significant skewness and distribution shifts across clients. Without addressing these issues, the model struggles to confidently pseudo-label unlabeled samples. Indicatively, using PL-FL on MNIST with maximum confidence threshold $\phi = 0.9$, only 4.5% of samples meet this criterion. Incorporating these pseudo-labeled samples into Phase2 yields a moderate performance improvement of 0.87%. In this case, we face two options: either conduct a significant high number of communication rounds, which is resource-intensive as it involves labelling a small percentage of data per round, or decrease the maximum confidence threshold ϕ to potentially include less confident samples for pseudo-labeling in earlier rounds.

Due to resource and communication constraints, we opt for the latter approach. We adjust the schedule of $\phi \in (0.5, 0.9)$ for all baselines to ensure that at least 10% of pseudo-labelled samples are included in subsequent rounds.

As shown in Table 5.1, by lowering ϕ leads to improved performance across all baselines in Phase2. However, when attempting to utilize the pseudo-labelled samples in Phase2+ engaging Type III clients in the process as well, we observe a decline in baselines' model performance apart from 2PFL. Even with the same confidence thresholds, baselines' models struggle to pseudo-label additional samples from Types II and III clients, often resulting in only 1 to 10 samples being pseudo-labelled, or none at all in most cases. Consequently, progressing with further training embracing all clients' types becomes less feasible.

On the other hand, 2PFL showcases its effectiveness and capacity to include all client types in all phases resulting in progressively labelling all unlabelled samples across the FL ecosystem with relatively high classification performance. This underscores the importance of employing dynamic threshold schedule to ensure the model learns from high-quality pseudo-labelled samples. To obtain such high-quality pseudo-labelled samples meeting the threshold criteria from the initial phase, 2PFL employs data augmentation to re-balance the data distribution across clients in the early stages of Phase1.

Table 5.1: Impact of high-confidence pseudo-labels on test accuracy across phases.

Dataset	Method	Phase1	Phase2	Phase2+
MNIST	**2PFL**	**96.93%**	**95.02%**	**97.31%**
	FedAvg	88.07%	88.67%	86.29%
	PL-FL	79.65%	85.10%	85.10%
	L&PL-FL	88.59%	90.01%	90.01%
F-MNIST	**2PFL**	**86.24%**	**88.05%**	**89.01%**
	FedAvg	81.15%	83.18%	82.16%
	PL-FL	76.70%	75.81%	75.77%
	L&PL-FL	71.43%	75.60%	72.43%
EMNIST	**2PFL**	**94.4%**	**94.8%**	**96.00%**
	FedAvg	72.47%	86.10%	84.35%
	PL-FL	53.30%	77.72%	83.45%
	L&PL-FL	84.38%	79.37%	78.20%
MEDMNIST	**2PFL**	**95.38%**	**98.53%**	**98.92%**
	FedAvg	54.69%	74.39%	71.41%
	PL-FL	49.76%	67.79%	59.54%
	L&PL-FL	86.45%	78.90%	74.88%

Comparison assessment with baselines.

Table 5.2 shows the effectiveness and efficiency of 2PFL compared against state-of-the-art baselines in terms of test accuracy, Labelled Data Ratio (LDR) per phase w.r.t. confidence threshold ϕ (i.e., percentage of samples that are labelled in each phase satisfying the ϕ confidence threshold), and number of training rounds. We show the total rounds T for Ideal, FedAvg, PL-FL, L&PL-FL and the rounds per phase (T_1, T_2, T_{2+}) for 2PFL with total rounds: $T_1 + T_2 + T_{2+}$. First, in terms of accuracy, 2PFL demonstrates the highest compared to FedAvg, PL-FL, and L&PL-FL across all datasets. Notably, 2PFL achieves comparable and even slightly higher

accuracy than Ideal model on F-MNIST and MedMNIST, especially due to augmentation and progressive highly-confident pseudo-labelling. For the other two datasets, we observe only a minimal difference of 0.62% and 0.41% accuracy. On average, we obtain less than 1% accuracy with less than 20% of the real data compared to Ideal. In a more detailed comparison against Ideal using MNIST, 2PFL achieves almost the same accuracy as Ideal (difference 0.622%), while, by the end of Phase2+, 2PFL obtains an additional 8.04% pseudo-labeled samples according to confidence threshold compared to Ideal. On average, 2PFL completed a total of 26 training rounds, six more rounds compared to Ideal. Similar patterns appeared in the rest of the datasets. Overall, this reflects our initial question about the 'price of labelling' indicating a significant improvement of labelling samples and achieving comparable performance with Ideal despite the skewed distribution of data. 2PFL is evidenced to achieve similar performance as Ideal in more realistic scenarios comparing with less realistic ones (assumed in baselines).

Comparing 2PFL against FedAvg over MNIST, 2PFL achieves a higher accuracy by 8.96% by pseudo-labelling 62% additional samples with high confidence. All of these improvements were attained with six extra rounds. Comparing with PL-FL, 2PFL achieves 18.14% higher accuracy by pseudo-labelling 61.75% more samples, with six fewer rounds. Whereas, comparing with L&PL-FL, 2PFL achieves 8.87% higher accuracy by pseudo-labeling 47.93% more samples, albeit requiring six additional rounds. 2PFL outperforms the baselines in terms of accuracy and rate of including pseudo-labelled samples in the training indicating its capacity to finally exploit *all* the available samples and knowledge across all client types. It is worth noting that 2PFL requires on average 8% more rounds compared to baselines across all datasets. This is expected because in each phase, 2PFL pseudo-labels a high percentage of unlabelled data, necessitating additional rounds for training in subsequent phases. Nonetheless, even with investing on a few more rounds, the price of pseudo-labelling becomes negligible compared to labelling samples from Type II & III clients. 2PFL attempts to include all clients avoiding leaving 'odd ones' out in federated training.

Table 5.2: Model test accuracy, LDR, and no. of training rounds across all methods.

		Baselines				**2PFL**		
Dataset	**Performance**	**Ideal**	**FedAvg**	**PL-FL**	**L&PL-FL**	**Phase1**	**Phase2**	**Phase2+**
MNIST	**Accuracy**	97.92%	88.59%	79.65%	88.67%	96.93%	95.02%	**97.31%**
	LDR,$\phi \in (0.5, 0.9)$	87.08%	35.25%	36.22%	49.31%	80.51%	82.78%	**94.70%**
	Rounds	20	20	32	20	10	11	5
F-MNIST	**Accuracy**	88.76%	79.89%	76.70%	71.43%	86.24%	88.05%	**89.01%**
	LDR,$\phi \in (0.5, 0.7)$	73.26%	20.11%	20.39%	49.31%	63.98%	70.77%	**88.80%**
	Rounds	20	20	20	20	10	7	5
EMNIST	**Accuracy**	96.40%	72.47%	53.30%	84.38%	94.4%	94.80%	**96.00%**
	LDR,$\phi \in (0.5, 0.9)$	66.3%	34.3%	39.37%	24.1%	63.525	67.07%	**76.55%**
	Rounds	20	18	15	20	10	10	8
MedMNIST	**Accuracy**	98.09%	54.69%	49.76%	86.45%	95.38%	98.53%	**98.92%**
	LDR,$\phi \in (0.5, 0.9)$	84.1%	26.53%	31.7%	20.22%	51.02%	60.57%	**82.91%**
	Rounds	30	20	20	20	10	5	7

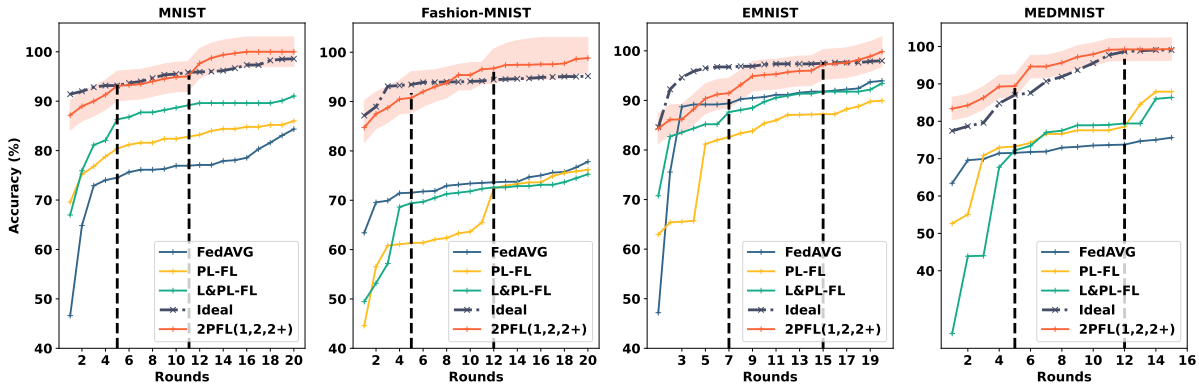


Figure 5.2: Accuracy vs. training rounds for all methods and datasets (the two vertical dotted lines correspond to T_1 and $T_1 + T_2$ round milestones of 2PFL's phases).

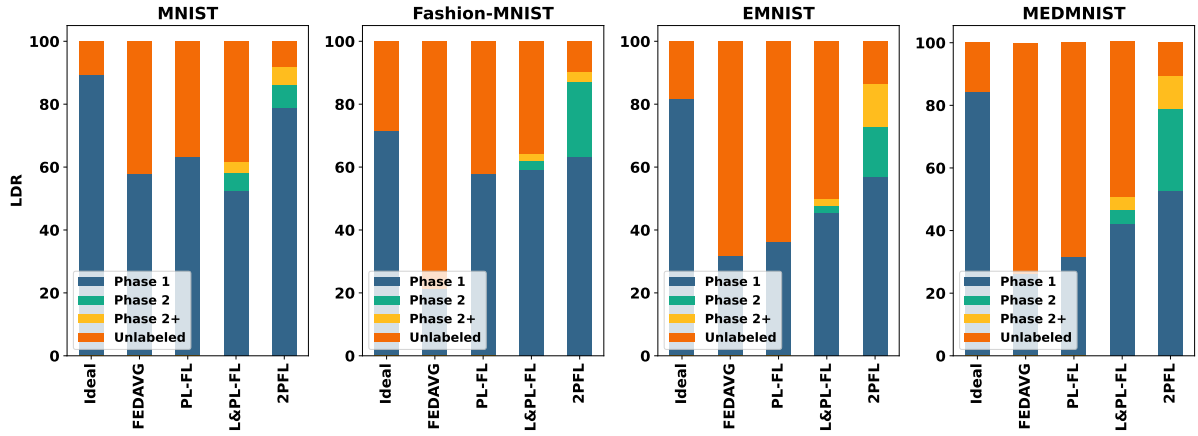


Figure 5.3: pseudo-labelling ratio of unlabelled samples across datasets and phases.

Impact of phases on model convergence & pseudo-labeling efficiency.

In terms of model convergence during training, the presence of non-IID data and label distribution shift can significantly hinder model convergence and compromise its ability to effectively adapt to various clients' types. Our baselines' comparisons reveal that, apart from Ideal, models exhibited slow convergence and lower accuracy compared to Ideal and 2PFL. Figure 5.2 shows that all baselines initially start with low accuracy and eventually plateau below or close to Ideal and 2PLF starting points in MNIST and Fashion-MNIST, and in EMNIST and MEDMNIST, respectively. This demonstrates that when the model is provided with low confidence pseudo-labelled samples, it results in reduced accuracy and slower convergence speed. Meanwhile, with 2PFL, the model converges relatively faster, with each phase positively impacting its speed. Notably, in Phase2+, 2PFL's model achieves higher accuracy than Ideal in MNIST, Fashion-MNIST, and EMNIST while matches the Ideal in MEDMNIST. This highlights the importance of pseudo-labeling and distribution re-balance via data augmentation influencing model performance.

Regarding pseudo-labeling efficiency measured by LDR, Figure 5.3 shows that in all baselines, models have pseudo-labelled the highest number of unlabelled samples from the first phase, as discussed in Section 5.5.2. On the contrary, 2PFL pseudo-labels the highest number of sam-

ples in Phase 1 while all the high-confidence (pseudo) labelled samples from Phase 1 are used to further improve the model performance in Phases 2 and 2+. 2PFL progressively pseudo-labels samples (some are also used for data augmentation tacking pseudo-class imbalance) achieving a final DLR close to Ideal. The indicates the impact of the dynamic confidence threshold on incrementally increasing the number of samples been labelled. Eventually, the increase rate DLR decreases from one phase to another as there a no more high-confidence samples considered for pseudo-labelling. As shown in Figure 5.3, after Phase 2+, the model does not proceed with pseudo-labeling as many samples as in Phases 1 and 2, thus, not bringing a further significant improvement.

5.6 Conclusions

Our 2PFL framework addresses the challenge of training FL models across different types of clients with limited and skewed labeled and unlabelled data. By leveraging data augmentation, 2PFL leads to improved model performance and accelerates convergence by progressive pseudo-labelling.s. Our comprehensive experiments and comparison with state-of-the-art methods highlight that 2PFL consistently outperforms baselines across various performance metrics and datasets. 2PFL exhibits superior convergence speed, accuracy, and data pseudo-labelling rate acquired in each phase. Therefore, *'the price for learning a global model with skewed and unlabelled data is minimal with 2PFL'*.

5.7 Limitations & Directions of Enhancement

The 2PFL framework has been proposed to train a global model (classifier) under conditions of extreme data scarcity and skewness across distributed labeled and unlabeled data. This framework relies on augmented data exchange to improve model performance. However, a limitation arises when augmented data cannot be exchanged due to resource constraints or communication overhead. In such cases, local models are trained only on local and augmented local data, which reduces model generalization and prediction confidence for unlabeled samples in other clients. Additionally, this limitation reduces the number of pseudo-labeled samples in each training step.

Another challenge with this framework is that it assumes at least 5% of the data is labeled. When labeled samples are below this threshold, achieving high-confidence predictions becomes difficult. Finally, during training, we assume predicted labels fall within the class range C, where C has values belong to $C = \{0, \ldots, C-1\}$.

In our future work, we plan to address more advanced scenarios, such as when there are only one or two labeled samples per class and when unlabeled clients might have label distributions beyond the range of C classes .

Chapter 6

Discussion and Conclusion

This thesis addresses the challenges of distributed data by studing the shift from the less privacy-restrictive environment of **EC** to the privacy-centric framework of **FL**. As data becomes increasingly decentralized and heterogeneous, and privacy regulations grow more stringent, this research introduces effective methods to enhance data representation, improve model performance and efficiency in distributed environments. The developed methodologies offer practical solutions to pressing concerns about data privacy and accessibility, enabling the field to effectively tackle the complexities of modern distributed machine learning systems.

6.1 Key Contributions

This thesis set out to answer several pivotal questions related to distributed learning, data offloading, and model training in environments where raw data cannot be shared across nodes. The contributions span multiple dimensions of federated and edge learning systems:

6.1.1 Data and Task Offloading in Edge Computing

In the first phase of this work, we tackled the challenge of optimizing task execution and data offloading in resource-constrained edge environments. The primary goal was to minimize data transmission while maximizing resource utilization on **edge servers**. By implementing task caching and fuzzy logic-based decision-making mechanisms, we developed strategies that effectively reduced latency and communication overhead while improving system efficiency. This approach was particularly impactful for time-sensitive tasks, such as real-time analytics, where data privacy and speed are critical.

6.1.2 Node and Data Selection in Distributed Learning

The next challenge addressed in the thesis was the intelligent selection of nodes and data in distributed predictive analytics. In traditional **FL**, irrelevant data is often fed to the model

during training due to the random selection of nodes, which can degrade model performance and consume the nodes' resources. The query-centric approach introduced in this thesis solves this issue by ranking nodes based on data relevance, ensuring that only the most suitable nodes participate in training. This not only enhances model accuracy but also reduces redundant communication in distributed systems, offering a significant step forward for FL systems that deal with non-IID data distributions.

6.1.3 Cluster-based & Label-aware Federated Meta-Learning

As the research progressed, addressing **label imbalance** and **non-iid data** distributions among clients emerged as a main challenge. Standard FL methods that rely on a single global model often falter when confronted with diverse data distributions, particularly in on-demand classification tasks. In response, we introduced the **Cluster-based & Label-aware Federated Meta-Learning (CL-FML)** framework, a method that clusters clients with similar label distributions and trains separate meta-models for each cluster. This approach dramatically improved performance by enabling focused learning in distributed environments, particularly for tasks where clients had diverse or missing label sets. Through CL-FML, we demonstrated that multi-model architectures are far more effective than traditional single-model methods, especially when handling arbitrary, unseen labels and tasks.

6.1.4 Self-Learning in Federated Learning with Minimal Labeled Data

One of the most pressing challenges in distributed learning environments is the **scarcity of labeled data**, as labeling is often expensive or infeasible in real-world applications. In the final technical chapter, we proposed the **Two-Phase Federated Self-Learning (2PFL)** framework to address this issue. 2PFL leverages both labeled and unlabeled data, using self-learning techniques to generate pseudo-labels from high-confidence predictions. This allowed to improve model performance even when working with limited labeled data—showing that models can achieve high accuracy with only 10-20% of labeled data. This framework is especially promising for future FL systems, where data labeling costs are likely to continue rising.

6.2 Addressing Future Challenges: Increasing Complexity in Data

As we look ahead, it is clear that **data challenges** will become even more pronounced. The growing emphasis on **data privacy** and **regulatory compliance**, alongside the need to ensure robust and efficient machine learning models, means that **access to data will become increasingly complicated**. As data collection becomes more restricted due to privacy concerns, the

heterogeneity of available data will also increase. This creates a situation where distributed systems must handle datasets that are not only sparse but also highly varied in structure, format, and labeling.

6.2.1 Data Privacy and Access Restrictions

With the rise of data protection regulations, such as **General Data Protection Regulation (GDPR)** and **California Consumer Privacy Act (CCPA)**, the rules around data sharing have become much stricter. In many fields, particularly healthcare and finance, **direct access to raw data** will continue to be tightly controlled. This will necessitate further innovation in **Federated Learning** approaches that allow for model training without accessing sensitive data. The frameworks proposed in this thesis, particularly CL-FML and 2PFL, serve as a foundation for handling such environments, as they are designed to operate with **minimal data sharing** while ensuring high model accuracy.

6.2.2 Increasing Data Heterogeneity

As data sources become more diverse, machine learning models will need to handle increasing levels of **data heterogeneity**. Whether it be differences in data formats, feature spaces, or label distributions, future models will need to adapt to a wide variety of data types. Our work on **label-aware clustering** and **meta-model learning** directly addresses this challenge by tailoring models to specific subsets of data, thus ensuring better performance in heterogeneous environments.

However, we recognize that future systems will need to go beyond just addressing heterogeneity at the label level. It will be critical to develop models that can handle **structural heterogeneity**, where data may come in different forms (e.g., text, images, time series) and varying levels of completeness. **Self-learning** techniques, such as the ones explored in 2PFL, are key to this future, as they allow models to **learn from unlabeled data** and **adapt dynamically** to new data distributions without the need for extensive human intervention.

6.3 Future Directions: Adapting to Data Scarcity and Complexity

In an era of **data scarcity** and **increasing complexity**, the future of **distributed machine learning** will hinge on our ability to make the most of what we have. As this thesis has shown, there is significant potential in leveraging **self-supervised learning**, **meta-learning**, and **multi-model architectures** to tackle the problems of **few labeled data** and **non-iid distributions**.

The key insight from this work is that future solutions must be **adaptive**, capable of **making the best use of limited data** while being prepared to handle **increasing heterogeneity**. Models

must be designed to learn not just from labeled data but also from unlabeled and semi-labeled data, as this will be a common scenario in many fields where labeling is time-consuming and expensive.

6.4 Final Remarks

In conclusion, this thesis has proposed a range of effective solutions to address key challenges that are commonly encountered in various federated learning scenarios, including **intelligent task management**, **node selection**, **cluster-based meta-learning**, and **self-learning**. As data challenges continue to evolve, it is imperative that we remain forward-thinking and ready to invest in **innovative solutions** that can build high-performance models with minimal data. By embracing the complexity of distributed systems and leveraging advances in **self-supervised learning**, we can ensure that future models are robust, efficient, and scalable in the face of growing data challenges.

The future of **distributed learning** will undoubtedly be filled with challenges, but with the right approaches—such as those explored in this thesis—we can build systems that are prepared to handle **any level of heterogeneity**, **minimal labeled data**, and **privacy constraints**. These advancements will be critical in fields such as **healthcare**, **finance**, **autonomous systems**, and beyond, where data privacy and diversity are paramount.

Table 6.1: Acronyms & Abbreviations.

Acronym	**Abbreviation**
EC	Edge Computing
CC	Cloud Computing
QoS	Quality of Services
FLI	Fuzzy Logic Inference System
DPA	Distributed Predictive Analytics
DML	Distributed Machine Learning
MSAP	Minimum Subset Average Problem
BNM	Best Node Model
MAD	Median Absolute Deviation
AM	Aggregate Model
WAM	Weighted Aggregate Model
FL	Federated Learning
RFL	Ranking-based Federated Learning
DS	Dataset
ML	Machine Learning
DL	Deep Learning
LR	Linear Regression
PR	Polynomial Regression
DNN	Deep Neural Network
MSE	Mean Squared Error
RIL	Ring-based Incremental Learning
GM	Global Model
RM	Random Model
GT	Game Theory
FS	Fair Selection
RL	Reinforcement Learning
MAB	Multi-armed Bandits
M-MAB	Max Utility-based MAB
TEF	Transformer Experts/Feedback MAB
SFL	Self-learning in FL environments

Bibliography

[1] Subrato Bharati et al. “Federated learning: Applications, challenges and future directions”. In: *International Journal of Hybrid Intelligent Systems* 18.1-2 (2022), pp. 19–35.

[2] Jiasi Chen and Xukan Ran. “Deep learning with edge computing: A review”. In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[3] Praveen Joshi et al. “Enabling all in-edge deep learning: a literature review”. In: *IEEE Access* 11 (2023), pp. 3431–3460.

[4] Haochen Hua et al. “Edge computing with artificial intelligence: A machine learning perspective”. In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35.

[5] Firdose Saeik et al. “Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions”. In: *Computer Networks* 195 (2021), p. 108177.

[6] Yanpei Liu et al. “Caching Placement Optimization Strategy Based on Comprehensive Utility in Edge Computing”. In: *Applied Sciences* 13.16 (2023), p. 9229.

[7] Siyuan Zhang, Guiquan Liu, et al. “Personalized Federated Learning with Attention Mechanisms”. In: *Academic Journal of Computing & Information Science* 6.11 (2023).

[8] You-Ru Lu, Xiaoqian Wang, and Dengfeng Sun. “Tackling Imbalanced Class in Federated Learning via Class Distribution Estimation”. In: (2022).

[9] Jing Zhang et al. “A Survey on Class Imbalance in Federated Learning”. In: *arXiv preprint arXiv:2303.11673* (2023).

[10] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[11] Pavlos Athanasios Apostolopoulos, Eirini Eleni Tsiropoulou, and Symeon Papavassiliou. “Risk-aware data offloading in multi-server multi-access edge computing environment”. In: *IEEE/ACM Transactions on Networking* 28.3 (2020), pp. 1405–1418.

[12] Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. "Data quality-aware task offloading in mobile edge computing: An optimal stopping theory approach". In: *Future Generation Computer Systems* 117 (2021), pp. 462–479.

[13] Dewang Ren et al. "GHCC: Grouping-based and hierarchical collaborative caching for mobile edge computing". In: *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE. 2018, pp. 1–6.

[14] Jaber Almutairi and Mohammad Aldossary. "A novel approach for IoT tasks offloading in edge-cloud environments". In: *Journal of Cloud Computing* 10.1 (2021), p. 28.

[15] Mohit Taneja and Alan Davy. "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm". In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2017, pp. 1222–1228.

[16] Yixue Hao et al. "Energy efficient task caching and offloading for mobile edge computing". In: *Ieee access* 6 (2018), pp. 11365–11373.

[17] Ihsan Ullah et al. "Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach". In: *Journal of Cloud Computing* 12.1 (2023), p. 112.

[18] Shuchen Zhou, Waqas Jadoon, and Iftikhar Ahmed Khan. "Computing offloading strategy in mobile edge computing environment: a comparison between adopted frameworks, challenges, and future directions". In: *Electronics* 12.11 (2023), p. 2452.

[19] Kostas Kolomvatsos and Christos Anagnostopoulos. "Proactive, uncertainty-driven queries management at the edge". In: *Future Generation Computer Systems* 118 (2021), pp. 75–93. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2020.12.028. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21000029.

[20] Mingye Li et al. "Efficient data offloading using markovian decision on state reward action in edge computing". In: *Journal of Grid Computing* 21.2 (2023), p. 25.

[21] VanDung Nguyen et al. "Joint offloading and IEEE 802.11 p-based contention control in vehicular edge computing". In: *IEEE Wireless Communications Letters* 9.7 (2020), pp. 1014–1018.

[22] Xiaohuan Rao et al. "Edge caching and computation offloading for fog-enabled radio access network". In: *Wireless Personal Communications* 109 (2019), pp. 297–313.

[23] Zubair Sharif et al. "Priority-Based Resource Allocation Scheme for Resources Usage in Mobile Edge Computing Platform". In: *Journal of Hunan University Natural Sciences* 50.1 (2023).

[24] Shuyang Li, Xiaohui Hu, and Yongwen Du. "Deep Reinforcement Learning for Computation Offloading and Resource Allocation in Unmanned-Aerial-Vehicle Assisted Edge Computing". In: *Sensors* 21.19 (2021), p. 6499.

[25] Jinlai Xu et al. “Zenith: Utility-aware resource allocation for edge computing”. In: *2017 IEEE international conference on edge computing (EDGE)*. IEEE. 2017, pp. 47–54.

[26] Utsav Drolia et al. “Cachier: Edge-caching for recognition applications”. In: *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE. 2017, pp. 276–286.

[27] Jie Luo et al. “QoE-driven computation offloading for edge computing”. In: *Journal of Systems Architecture* 97 (2019), pp. 34–39.

[28] Zhaolong Ning et al. “Deep reinforcement learning for vehicular edge computing: An intelligent offloading system”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.6 (2019), pp. 1–24.

[29] Yuben Qu et al. “CoTask: Correlation-aware task offloading in edge computing”. In: *World Wide Web* 25.5 (2022), pp. 2185–2213.

[30] Ke Zhang et al. “Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading”. In: *IEEE Vehicular Technology Magazine* 12.2 (2017), pp. 36–44.

[31] Ying Liu et al. “Data caching optimization in the edge computing environment”. In: *IEEE transactions on services computing* 15.4 (2020), pp. 2074–2085.

[32] Jiaxin Zhang, Xing Zhang, and Wenbo Wang. “Cache-enabled software defined heterogeneous networks for green and flexible 5G networks”. In: *IEEE Access* 4 (2016), pp. 3591–3604.

[33] Shuo Wang et al. “Distributed edge caching scheme considering the tradeoff between the diversity and redundancy of cached content”. In: *2015 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE. 2015, pp. 1–5.

[34] Tuyen X Tran and Dario Pompili. “Adaptive bitrate video caching and processing in mobile-edge computing networks”. In: *IEEE Transactions on Mobile Computing* 18.9 (2018), pp. 1965–1978.

[35] Zhong Yang et al. “Cache-aided NOMA mobile edge computing: A reinforcement learning approach”. In: *IEEE Transactions on Wireless Communications* 19.10 (2020), pp. 6899–6915.

[36] Changsheng You et al. “Energy-efficient resource allocation for mobile-edge computation offloading”. In: *IEEE Transactions on Wireless Communications* 16.3 (2016), pp. 1397–1411.

[37] Pengfei Wang et al. “Task-driven data offloading for fog-enabled urban IoT services”. In: *IEEE Internet of Things Journal* 8.9 (2020), pp. 7562–7574.

[38] Haibo Ge et al. “Multi-server intelligent task caching strategy for edge computing”. In: *2022 4th International Conference on Natural Language Processing (ICNLP)*. IEEE. 2022, pp. 563–569.

[39] Lujie Tang et al. “A novel task caching and migration strategy in multi-access edge computing based on the genetic algorithm”. In: *Future Internet* 11.8 (2019), p. 181.

[40] Ibrahim A Elgendy et al. “Joint computation offloading and task caching for multi-user and multi-task MEC systems: reinforcement learning-based algorithms”. In: *Wireless Networks* 27.3 (2021), pp. 2023–2038.

[41] Lixing Chen et al. “Spatio–temporal edge service placement: A bandit learning approach”. In: *IEEE Transactions on Wireless Communications* 17.12 (2018), pp. 8388–8401.

[42] Lixing Chen and Jie Xu. “Budget-constrained edge service provisioning with demand estimation via bandit learning”. In: *IEEE Journal on Selected Areas in Communications* 37.10 (2019), pp. 2364–2376.

[43] Yiming Miao et al. “Intelligent task caching in edge cloud via bandit learning”. In: *IEEE transactions on network science and engineering* 8.1 (2020), pp. 625–637.

[44] Stephen T Welstead. *Neural network and fuzzy logic applications in C/C++*. John Wiley & Sons, Inc., 1994.

[45] Mohammad G Khoshkholgh et al. “Randomized caching in cooperative UAV-enabled fog-RAN”. In: *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2019, pp. 1–6.

[46] VanDung Nguyen et al. “Flexible computation offloading in a fuzzy-based mobile edge orchestrator for IoT applications”. In: *Journal of Cloud Computing* 9.1 (2020), pp. 1–18.

[47] Zhenjiang Zhang et al. “A new task offloading algorithm in edge computing”. In: *EURASIP Journal on Wireless Communications and Networking* 2021.1 (2021), p. 17.

[48] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. “Fuzzy workload orchestration for edge computing”. In: *IEEE Transactions on Network and Service Management* 16.2 (2019), pp. 769–782.

[49] U Mohan Rao, YR Sood, and RK Jarial. “Subtractive clustering fuzzy expert system for engineering applications”. In: *Procedia Computer Science* 48 (2015), pp. 77–83.

[50] Navuday Sharma et al. “On-demand ultra-dense cloud drone networks: Opportunities, challenges and benefits”. In: *IEEE Communications Magazine* 56.8 (2018), pp. 85–91.

[51] Zhixiong Chen, Nan Xiao, and Dongsheng Han. “Multilevel task offloading and resource optimization of edge computing networks considering UAV relay and green energy”. In: *Applied sciences* 10.7 (2020), p. 2592.

[52] Paul Voigt and Axel Von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer International Publishing, 2017.

[53] California Legislative Information. *California Consumer Privacy Act (CCPA)*. https://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_id=201720180AB375. 2018.

[54] U.S. Department of Health & Human Services. *Health Insurance Portability and Accountability Act of 1996 (HIPAA)*. https://www.hhs.gov/hipaa/for-professionals/privacy/index.html. 1996.

[55] Dongdong Ye et al. "Federated learning in vehicular edge computing: A selective model aggregation approach". In: *IEEE Access* 8 (2020), pp. 23920–23935.

[56] Dean Abbott. *Applied Predictive Analytics: Principles and Techniques for the Professional Data Analyst*. 1st ed. Wiley Publishing, 2014. ISBN: 1118727967.

[57] Christos Anagnostopoulos. "Edge-centric inferential modeling analytics". In: *Journal of Network and Computer Applications* 164 (2020), p. 102696. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102696. URL: https://www.sciencedirect.com/science/article/pii/S1084804520301703.

[58] Siwei Feng and Han Yu. "Multi-participant multi-class vertical federated learning". In: *arXiv preprint arXiv:2001.11154* (2020).

[59] Zirui Zhu and Lifeng Sun. "Federated Trace: A Node Selection Method for More Efficient Federated Learning". In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2021, pp. 1234–1238.

[60] Chien-Sheng Yang, Ramtin Pedarsani, and A Salman Avestimehr. "Edge computing in the dark: Leveraging contextual-combinatorial bandit and coded computing". In: *IEEE/ACM Transactions on Networking* 29.3 (2021), pp. 1022–1031.

[61] Yongheng Deng et al. "Auction: Automated and quality-aware client selection framework for efficient federated learning". In: *IEEE Transactions on Parallel and Distributed Systems* 33.8 (2021), pp. 1996–2009.

[62] Fernando E Casado et al. "Concept drift detection and adaptation for federated and continual learning". In: *Multimedia Tools and Applications* (2022), pp. 1–23.

[63] Mannsoo Hong, Seok-Kyu Kang, and Jee-Hyong Lee. "Weighted Averaging Federated Learning Based on Example Forgetting Events in Label Imbalanced Non-IID". In: *Applied Sciences* 12.12 (2022), p. 5806.

[64] Kostas Kolomvatsos and Christos Anagnostopoulos. "A Proactive Statistical Model Supporting Services and Tasks Management in Pervasive Applications". In: *IEEE Transactions on Network and Service Management* 19.3 (2022), pp. 3020–3031.

[65] Sumit Rai, Arti Kumari, and Dilip K Prasad. "Client Selection in Federated Learning under Imperfections in Environment". In: *AI* 3.1 (2022), pp. 124–145.

[66] Christopher Tran and Elena Zheleva. "Improving Data-driven Heterogeneous Treatment Effect Estimation Under Structure Uncertainty". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 1787–1797.

[67] Paolo Bellavista et al. "Differentiated Service/Data Migration for Edge Services Leveraging Container Characteristics". In: *IEEE Access* 7 (2019), pp. 139746–139758.

[68] Georgios Boulougaris and Kostas Kolomvatsos. "A QoS-aware, Proactive Tasks Offloading Model for Pervasive Applications". In: *2022 9th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2022, pp. 24–31.

[69] Kostas Kolomvatsos et al. "Proactive Time-Optimized Data Synopsis Management at the Edge". In: *IEEE Transactions on Knowledge and Data Engineering* 34.7 (2022), pp. 3478–3490. DOI: 10.1109/TKDE.2020.3021377.

[70] Anna Karanika et al. "A Demand-driven, Proactive Tasks Management Model at the Edge". In: *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2020, pp. 1–8.

[71] Madalena Soula et al. "Intelligent tasks allocation at the edge based on machine learning and bio-inspired algorithms". In: *Evolving Systems* 13.2 (Apr. 2022), pp. 221–242.

[72] Shengchao Chen et al. "Prompt federated learning for weather forecasting: Toward foundation models on meteorological data". In: *arXiv preprint arXiv:2301.09152* (2023).

[73] Reza Shokri and Vitaly Shmatikov. "Privacy-preserving deep learning". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1310–1321.

[74] Parimala Boobalan et al. "Fusion of federated learning and industrial Internet of Things: A survey". In: *Computer Networks* 212 (2022), p. 109048.

[75] Jianyu Wang, Ming Zhao, and Li Yang. "Federated Learning for Industrial IoT: A Case Study on Predictive Maintenance". In: *IEEE Transactions on Industrial Informatics* 16.5 (2020), pp. 3230–3240.

[76] Yanna Jiang et al. "Blockchained federated learning for internet of things: A comprehensive survey". In: *ACM Computing Surveys* 56.10 (2024), pp. 1–37.

[77] Xu Cheng, Chendan Li, and Xiufeng Liu. "A review of federated learning in energy systems". In: *2022 IEEE/IAS industrial and commercial power system Asia (I&CPS Asia)* (2022), pp. 2089–2095.

[78] Thanh Toan Nguyen et al. "Manipulating recommender systems: A survey of poisoning attacks and countermeasures". In: *ACM Computing Surveys* (2024).

[79] Woonghee Lee. "Reward-based participant selection for improving federated reinforcement learning". In: *ICT Express* (2022).

[80] Rituparna Saha et al. "Data-Centric Client Selection for Federated Learning Over Distributed Edge Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 34.2 (2022), pp. 675–686.

[81] Ahmad Hammoud et al. "Data-driven federated autonomous driving". In: *International Conference on Mobile Web and Intelligent Information Systems*. Springer. 2022, pp. 79–90.

[82] Jack Goetz et al. "Active federated learning". In: *arXiv preprint arXiv:1909.12641* (2019).

[83] Shameem A. Puthiya Parambath, Christos Anagnostopoulos, and Roderick Murray-Smith. "Sequential query prediction based on multi-armed bandits with ensemble of transformer experts and immediate feedback". In: *Data Mining and Knowledge Discovery* (Aug. 2024). URL: https://doi.org/10.1007/s10618-024-01057-4.

[84] Jaewook Lee et al. "Data distribution-aware online client selection algorithm for federated learning in heterogeneous networks". In: *IEEE Transactions on Vehicular Technology* 72.1 (2022), pp. 1127–1136.

[85] Zhongchang Zhou et al. "A Decentralized Federated Learning Based on Node Selection and Knowledge Distillation". In: *Mathematics* 11.14 (2023), p. 3162.

[86] Christos Anagnostopoulos and Kostas Kolomvatsos. "An intelligent, time-optimized monitoring scheme for edge nodes". In: *Journal of Network and Computer Applications* 148 (2019), p. 102458. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.102458. URL: https://www.sciencedirect.com/science/article/pii/S1084804519303182.

[87] Weiwei Lin et al. "Contribution-based Federated Learning client selection". In: *International Journal of Intelligent Systems* (2022).

[88] Tiansheng Huang et al. "Stochastic client selection for federated learning with volatile clients". In: *IEEE Internet of Things Journal* (2022).

[89] Xiong Wang, Jiancheng Ye, and John CS Lui. "Decentralized task offloading in edge computing: A multi-user multi-armed bandit approach". In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE. 2022, pp. 1199–1208.

[90] Tze Leung Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22.

[91] Saeed Ghoorchian and Setareh Maghsudi. "Multi-armed bandit for energy-efficient and delay-sensitive edge computing in dynamic networks with uncertainty". In: *IEEE Transactions on Cognitive Communications and Networking* 7.1 (2020), pp. 279–293.

[92] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. “Finite-time analysis of the multi-armed bandit problem”. In: *Machine learning* 47 (2002), pp. 235–256.

[93] Bochun Wu, Tianyi Chen, and Xin Wang. “A combinatorial bandit approach to UAV-aided edge computing”. In: *2020 54th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2020, pp. 304–308.

[94] Wen He et al. “Bandit learning-based service placement and resource allocation for mobile edge computing”. In: *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE. 2020, pp. 1–6.

[95] Jianji Ren et al. “Collaborative edge computing and caching with deep reinforcement learning decision agents”. In: *IEEE Access* 8 (2020), pp. 120604–120612.

[96] Shuai Yu et al. “When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network”. In: *IEEE Internet of Things Journal* 8.4 (2020), pp. 2238–2251.

[97] Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P. Pezaros. “Data quality-aware task offloading in Mobile Edge Computing: An Optimal Stopping Theory approach”. In: *Future Gener. Comput. Syst.* 117 (2021), pp. 462–479.

[98] Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P. Pezaros. “Optimized Contextual Data Offloading in Mobile Edge Computing”. In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2021, pp. 473–479.

[99] Kyriaki Panagidi et al. “To Transmit or Not to Transmit: Controlling Communications in the Mobile IoT Domain”. In: *ACM Trans. Internet Techn.* 20.3 (2020), 22:1–22:23.

[100] Tiffany Tuor et al. “Data Selection for Federated Learning with Relevant and Irrelevant Data at Clients”. In: *ArXiv* abs/2001.08300 (2020).

[101] Lokesh Nagalapatti, Ruhi Sharma Mittal, and Ramasuri Narayanam. “Is your data relevant?: Dynamic selection of relevant data for federated learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 7. 2022, pp. 7859–7867.

[102] Tahani Aladwani et al. “Query-driven Edge Node Selection in Distributed Learning Environments”. In: *39th IEEE International Conference on Data Engineering, ICDE 2023 - Workshops, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2023, pp. 146–153. DOI: 10.1109/ICDEW58674.2023.00029.

[103] Fotis Savva et al. “Large-Scale Data Exploration Using Explanatory Regression Functions”. In: *ACM Trans. Knowl. Discov. Data* 14.6 (Sept. 2020). ISSN: 1556-4681. DOI: 10.1145/3410448. URL: https://doi.org/10.1145/3410448.

[104] Sepanta Zeighami et al. “A neural database for differentially private spatial range queries”. In: *Proc. VLDB Endow.* 15.5 (Jan. 2022), pp. 1066–1078. ISSN: 2150-8097.

[105] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets". In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD '00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 427–438. ISBN: 1581132174.

[106] Fabrizio Angiulli, Salvatore Basta, and Claudio Pizzuti. "Distance-based Detection and Prediction of Outliers". In: *IEEE Transactions on Knowledge and Data Engineering* 18.2 (2006), pp. 145–160. DOI: 10.1109/TKDE.2006.29.

[107] Rassoul Hajizadeh, Ali Aghagolzadeh, and Mehdi Ezoji. "Mutual neighborhood and modified majority voting based KNN classifier for multi-categories classification". In: *Pattern Anal. Appl.* 25.4 (Nov. 2022), pp. 773–793. ISSN: 1433-7541.

[108] Christos Anagnostopoulos and Peter Triantafillou. "Query-Driven Learning for Predictive Analytics of Data Subspace Cardinality". In: *ACM Trans. Knowl. Discov. Data* 11.4 (June 2017). ISSN: 1556-4681.

[109] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. "SuRF: Identification of Interesting Data Regions with Surrogate Models". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1321–1332.

[110] Dinh Phamtoan and Tai Vovan. "Automatic fuzzy genetic algorithm in clustering for images based on the extracted intervals". In: *Multimedia Tools Appl.* 80.28–29 (Nov. 2021), pp. 35193–35215. ISSN: 1380-7501.

[111] Sony Peng et al. "Centralized machine learning versus federated averaging: A comparison using mnist dataset". In: *KSII Transactions on Internet and Information Systems (TIIS)* 16.2 (2022), pp. 742–756.

[112] Yue Wu et al. "Personalized Federated Learning under Mixture of Distributions". In: *arXiv preprint arXiv:2305.01068* (2023).

[113] Fotis Savva et al. "Large-scale data exploration using explanatory regression functions". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14.6 (2020), pp. 1–33.

[114] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. "Outlier Detection: How to Threshold Outlier Scores?" In: *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing*. AIIPCC '19. Sanya, China: Association for Computing Machinery, 2019. ISBN: 9781450376334. DOI: 10.1145/3371425.3371427. URL: https://doi.org/10.1145/3371425.3371427.

[115] H. Brendan McMahan et al. "Federated Learning of Deep Networks using Model Averaging". In: *CoRR* abs/1602.05629 (2016).

[116] Tan Li and Linqi Song. "Privacy-Preserving Communication-Efficient Federated Multi-Armed Bandits". In: *IEEE Journal on Selected Areas in Communications* 40.3 (2022), pp. 773–787. DOI: 10.1109/JSAC.2022.3142374.

[117] Vicenç Torra. "A systematic construction of non-i.i.d. data sets from a single data set: non-identically distributed data". In: *Knowledge and Information Systems* 65.3 (Mar. 2023), pp. 991–1003.

[118] Martin Arjovsky et al. *Invariant Risk Minimization*. 2020. arXiv: 1907.02893 [stat.ML]. URL: https://arxiv.org/abs/1907.02893.

[119] Benjamin Aubin et al. *Linear unit-tests for invariance discovery*. 2021. arXiv: 2102.10867 [cs.LG]. URL: https://arxiv.org/abs/2102.10867.

[120] Shameem Puthiya Parambath, Nicolas Usunier, and Yves Grandvalet. "Optimizing F-Measures by Cost-Sensitive Classification." In: *NIPS*. 2014, pp. 2123–2131.

[121] Yunlu Yan and Lei Zhu. *A Simple Data Augmentation for Feature Distribution Skewed Federated Learning*. 2023. arXiv: 2306.09363 [cs.LG]. URL: https://arxiv.org/abs/2306.09363.

[122] Tahani Aladwani et al. "The Price of Labelling: A Two-Phase Federated Self-learning Approach". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2024, pp. 126–142.

[123] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. "Aggregate Query Prediction under Dynamic Workloads". In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 671–676.

[124] Christos Anagnostopoulos and Peter Triantafillou. "Efficient Scalable Accurate Regression Queries in In-DBMS Analytics". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 2017, pp. 559–570.

[125] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. "Adaptive learning of aggregate analytics under dynamic workloads". In: *Future Generation Computer Systems* 109 (2020), pp. 317–330. ISSN: 0167-739X.

[126] Kostas Kolomvatsos and Christos Anagnostopoulos. "A probabilistic model for assigning queries at the edge". In: *Computing* 102.4 (Apr. 2020), pp. 865–892. ISSN: 1436-5057.

[127] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. "Overview of Data Exploration Techniques". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD '15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 277–281. ISBN: 9781450327589. DOI: 10.1145/2723372.2731084.

[128] Binbin Gu, Saeed Kargar, and Faisal Nawab. "Efficient Dynamic Clustering: Capturing Patterns from Historical Cluster Evolution". In: *International Conference on Extending Database Technology*. 2022. URL: https://api.semanticscholar.org/CorpusID:247218619.

[129] Qianyu Long, Kostas Kolomvatsos, and Christos Anagnostopoulos. "Knowledge reuse in edge computing environments". In: *Journal of Network and Computer Applications* 206 (2022), p. 103466. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103466. URL: https://www.sciencedirect.com/science/article/pii/S108480452200114X.

[130] Qianyu Long et al. "FedDIP: Federated Learning with Extreme Dynamic Pruning and Incremental Regularization". In: *2023 IEEE International Conference on Data Mining (ICDM)*. 2023, pp. 1187–1192. DOI: 10.1109/ICDM58522.2023.00146.

[131] Qianyu Long, Christos Anagnostopoulos, and Kostas Kolomvatsos. "Enhancing Knowledge Reusability: A Distributed Multitask Machine Learning Approach". In: *IEEE Transactions on Emerging Topics in Computing* (2024), pp. 1–14. DOI: 10.1109/TETC.2024.3390811.

[132] Nastasiya F. Grinberg, Oghenejokpeme I. Orhobor, and Ross D. King. "An evaluation of machine-learning for predicting phenotype: studies in yeast, rice, and wheat". In: *Machine Learning* 109.2 (Feb. 2020), pp. 251–277.

[133] Shameem Puthiya Parambath et al. "Max-Utility based arm selection strategy for sequential query recommendations". In: *Asian Conference on Machine Learning*. PMLR. 2021, pp. 564–579.

[134] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. "Bandits and Experts in Metric Spaces". In: *J. ACM* 66.4 (May 2019). ISSN: 0004-5411. URL: https://doi.org/10.1145/3299873.

[135] Lihong Li et al. "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 661–670. ISBN: 9781605587998. URL: https://doi.org/10.1145/1772690.1772758.

[136] Saleh ALFahad et al. "Task offloading in mobile edge computing using cost-based discounted optimal stopping". In: *Open Computer Science* 14.1 (2024), p. 20230115.

[137] Qiyuan Wang et al. "Maintenance of model resilience in distributed edge learning environments". In: *2023 19th International Conference on Intelligent Environments (IE)*. IEEE. 2023, pp. 1–8.

[138] Tehrim Yoon et al. “Fedmix: Approximation of mixup under mean augmented federated learning”. In: *arXiv preprint arXiv:2107.00233* (2021).

[139] TV Nguyen et a. “A novel decentralized federated learning approach to train on globally distributed, poor quality, and protected private medical data”. In: *Scientific Reports* 12.1 (2022), p. 8888.

[140] Hongda Wu and Ping Wang. “Probabilistic Node Selection for Federated Learning with Heterogeneous Data in Mobile Edge”. In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2022, pp. 2453–2458.

[141] Yongxin Guo, Xiaoying Tang, and Tao Lin. “FedRC: Tackling Diverse Distribution Shifts Challenge in Federated Learning by Robust Clustering”. In: *arXiv preprint arXiv:2301.12379* (Jan. 2023).

[142] Haoyu Ren, Darko Anicic, and Thomas A Runkler. “TinyReptile: TinyML with Federated Meta-Learning”. In: *arXiv preprint arXiv:2304.05201* (2023).

[143] Hae Beom Lee et al. “Learning to balance: Bayesian meta-learning for imbalanced and out-of-distribution tasks”. In: *arXiv preprint arXiv:1905.12917* (2019).

[144] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. “Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach”. In: *Adv Neural Inf Process Syst* 33 (2020), pp. 3557–3568.

[145] Sen Lin, Guang Yang, and Junshan Zhang. “A collaborative learning framework via federated meta-learning”. In: *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2020, pp. 289–299.

[146] Jie Yan et al. “Federated clustering with GAN-based data synthesis”. In: *arXiv preprint arXiv:2210.16524* (2022).

[147] Mei Cao et al. “C2s: Class-aware client selection for effective aggregation in federated learning”. In: *High-Confidence Computing* 2.3 (2022), p. 100068.

[148] Amirhossein Reisizadeh et al. “Robust federated learning: The case of affine distribution shifts”. In: *Adv Neural Inf Process Syst* 33 (2020), pp. 21554–21565.

[149] Jian Xu and Shao-Lun Huang. “A Joint Training-Calibration Framework for Test-Time Personalization with Label Shift in Federated Learning”. In: *32nd ACM CIKM*. 2023, pp. 4370–4374.

[150] Lei Yang et al. “Personalized federated learning on non-IID data via group-based meta-learning”. In: *TKDD* 17.4 (2023), pp. 1–20.

[151] Hong Lang et al. “Augmented Concrete Crack Segmentation: Learning Complete Representation to Defend Background Interference in Concrete Pavements”. In: *IEEE TIM* 73 (2024), pp. 1–13.

[152] Chenyou Fan, Junjie Hu, and Jianwei Huang. "Private semi-supervised federated learning". In: *International Joint Conference on Artificial Intelligence*. 2022.

[153] Sulaiman A. Alghunaim and Kun Yuan. "A Unified and Refined Convergence Analysis for Non-Convex Decentralized Learning". In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 3264–3279.

[154] Xiaomin Ouyang et al. "Clusterfl: a similarity-aware federated learning system for human activity recognition". In: *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 2021, pp. 54–66.

[155] Lei Yang et al. "Personalized Federated Learning on Non-IID Data via Group-Based Meta-Learning". In: *ACM Trans. Knowl. Discov. Data* 17.4 (Mar. 2023). ISSN: 1556-4681.

[156] Yuwei Sun, Ng Chong, and Hideya Ochiai. "Feature distribution matching for federated domain generalization". In: *Asian Conference on Machine Learning*. PMLR. 2023, pp. 942–957.

[157] Christopher Briggs, Zhong Fan, and Peter Andras. "Federated learning with hierarchical clustering of local updates to improve training on non-IID data". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–9.

[158] Avishek Ghosh et al. "An efficient framework for clustered federated learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19586–19597.

[159] Jianfei Zhang and Shuaishuai Lv. "Fedlabcluster: A clustered federated learning algorithm based on data sample label". In: *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*. IEEE. 2021, pp. 423–428.

[160] Sen Lin, Guang Yang, and Junshan Zhang. "Real-time edge intelligence in the making: A collaborative learning framework via federated meta-learning". In: *arXiv preprint arXiv:2001.03229* (2020).

[161] Weiming Zhuang et al. "MAS: Towards Resource-Efficient Federated Multiple-Task Learning". In: *IEEE/CVF CVPR*. 2023, pp. 23414–23424.

[162] Jiangang Shu et al. "Clustered federated multitask learning on non-IID data with enhanced privacy". In: *IEEE Internet of Things* 10.4 (2022), pp. 3453–3467.

[163] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints". In: *IEEE transactions on neural networks and learning systems* 32.8 (2020), pp. 3710–3722.

[164] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[165] Yunlu Yan and Lei Zhu. "A Simple Data Augmentation for Feature Distribution Skewed Federated Learning". In: *arXiv preprint arXiv:2306.09363* (2023).

[166] Jun Luo and Shandong Wu. "Fedsld: Federated learning with shared label distribution for medical image classification". In: *19th IEEE ISBI*. 2022, pp. 1–5.

[167] Afroditi Papadaki et al. "Minimax demographic group fairness in federated learning". In: *2022 ACM FAccT*. 2022, pp. 142–159.

[168] Ahmad Khalil et al. "Label-Aware Aggregation for Improved Federated Learning". In: *8th FMEC*. IEEE. 2023, pp. 216–223.

[169] Yu-Tong Cao et al. "Knowledge-aware federated active learning with non-iid data". In: *IEEE/CVF*. 2023, pp. 22279–22289.

[170] Murali Krishna Ramanathan et al. "Randomized Leader Election". In: *Distrib. Comput.* 19.5–6 (Apr. 2007), pp. 403–418.

[171] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. "A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions". In: *ACM Comput. Surv.* 52.4 (Aug. 2019). ISSN: 0360-0300.

[172] Amos Tversky. "Features of similarity". In: *Psychological Review* 84.4 (1977), pp. 327–352. ISSN: 19391471.

[173] Aaron Carass et al. "Evaluating White Matter Lesion Segmentations with Refined Sørensen-Dice Analysis". In: *Scientific Reports* 10, 8242 (May 2020), p. 8242.

[174] Xiaomin Ouyang et al. "ClusterFL: A Clustering-Based Federated Learning System for Human Activity Recognition". In: *ACM Trans. Sen. Netw.* 19.1 (Dec. 2022). ISSN: 1550-4859.

[175] Enmao Diao, Jie Ding, and Vahid Tarokh. "SemiFL: Semi-supervised federated learning for unlabeled clients with alternate training". In: *NeurIPS* 35 (2022), pp. 17871–17884.

[176] Bowen Zhang et al. "Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling". In: *NeurIPS* 34 (2021), pp. 18408–18419.

[177] Liang Qiu et al. "Federated Semi-Supervised Learning for Medical Image Segmentation via Pseudo-Label Denoising". In: *IEEE Journal of Biomedical and Health Informatics* (2023).

[178] Zhengyi Zhong et al. "Semi-HFL: semi-supervised federated learning for heterogeneous devices". In: *Complex & Intelligent Systems* 9.2 (2023), pp. 1995–2017.

[179] Hongyi Zhang et al. "mixup: Beyond empirical risk minimization". In: *arXiv preprint arXiv:1710.09412* (2017).

[180] Liwei Che et al. "Fedtrinet: A pseudo labeling method with three players for federated semi-supervised learning". In: *IEEE Intl Conf Big Data*. 2021, pp. 715–724.

[181] Xiaoxiao Liang et al. “Rscfed: Random sampling consensus federated semi-supervised learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10154–10163.

[182] Sunder Ali Khowaja et al. “Selffed: Self-supervised federated learning for data heterogeneity and label scarcity in iomt”. In: *arXiv preprint arXiv:2307.01514* (2023).

[183] Aisha Mohamed et al. “Popularity agnostic evaluation of knowledge graph embeddings”. In: *UAI*. PMLR. 2020, pp. 1059–1068.

[184] Moming Duan et al. “Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications”. In: *ICCD*. IEEE. 2019, pp. 246–254.

[185] Yutong Dai et al. “Tackling data heterogeneity in federated learning with class prototypes”. In: *AAAI*. Vol. 37. 6. 2023, pp. 7314–7322.

[186] Aldin Vehabovic et al. “Ransomware Detection Using Federated Learning with Imbalanced Datasets”. In: *IEEE 20th HONET*. IEEE. 2023, pp. 255–260.

[187] Han Wang et al. “Non-IID data re-balancing at IoT edge with peer-to-peer federated learning for anomaly detection”. In: *14th ACM WiSec*. 2021, pp. 153–163.

[188] Hyuck Lee, Seungjae Shin, and Heeyoung Kim. “Abc: Auxiliary balanced classifier for class-imbalanced semi-supervised learning”. In: *NeurIPS* 34 (2021), pp. 7082–7094.

[189] Suraj Kothawade et al. “Basil: Balanced active semi-supervised learning for class imbalanced datasets”. In: *arXiv preprint arXiv:2203.05651* (2022).

[190] Chaoyang He et al. “Ssfl: Tackling label deficiency in federated learning via personalized self-supervision”. In: *preprint arXiv:2110.02470* (2021).

[191] Jieming Bian, Zhu Fu, and Jie Xu. “FedSEAL: semi-supervised federated learning with self-ensemble learning and negative learning”. In: *preprint arXiv:2110.07829* (2021).

[192] Milind Rao et al. “Federated self-learning with weak supervision for speech recognition”. In: *ICASSP*. IEEE. 2023, pp. 1–5.

[193] Vasileios Tsouvalas, Aaqib Saeed, and Tanir Ozcelebi. “Federated self-training for semi-supervised audio recognition”. In: *ACM TECS* 21.6 (2022), pp. 1–26.

[194] Zhe Zhang et al. “Semi-supervised federated learning with non-iid data: Algorithm and system design”. In: *HPCC*. 2021, pp. 157–164.

[195] Taehyeon Kim et al. “Navigating Data Heterogeneity in Federated Learning: A Semi-Supervised Approach for Object Detection”. In: *NeurIPS* 36 (2024).

[196] Haowen Lin et al. “Semifed: Semi-supervised federated learning with consistency and pseudo-labeling”. In: *arXiv preprint arXiv:2108.09412* (2021).

[197] Mingzhao Yang et al. “Exploring One-shot Semi-supervised Federated Learning with A Pre-trained Diffusion Model”. In: *arXiv preprint arXiv:2305.04063* (2023).

[198] Te-Chuan Chiu et al. “Semisupervised distributed learning with non-IID data for AIoT service platform”. In: *IEEE Internet of Things Journal* 7.10 (2020), pp. 9266–9277.

[199] Ming Li, Qingli Li, and Yan Wang. “Class Balanced Adaptive Pseudo Labeling for Federated Semi-Supervised Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16292–16301.

[200] Chao Zhang et al. “Non-IID always Bad? Semi-Supervised Heterogeneous Federated Learning with Local Knowledge Enhancement”. In: *32nd ACM International Conference on Information and Knowledge Management*. 2023, pp. 3257–3267.

[201] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. “Learning Structured Output Representation using Deep Conditional Generative Models”. In: *NeurIPS*. Vol. 28. 2015.

[202] Zonglin Di et al. “Federated Learning with Openset Noisy Labels”. In: (2022).