UNIVERSITY OF CALIFORNIA SAN DIEGO

Effective Learning in Heterogeneous Distributed Environments

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Jiayun Zhang

Committee in charge:

Professor Rajesh K. Gupta, Chair
Professor Jingbo Shang, Co-Chair
Professor Arya Mazumdar
Professor Julian McAuley

2025

The Dissertation of Jiayun Zhang is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2025

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am deeply grateful to everyone who has supported me throughout my Ph.D. study.

First and foremost, I would like to thank my advisors, Prof. Rajesh Gupta and Prof. Jingbo Shang, for their guidance, patience, and trust. Rajesh gave me the freedom to explore my research interests while consistently providing insightful advice to keep me on track. His visionary input has profoundly influenced how I understand and value the broader implications of my research. Jingbo has been a wonderful co-advisor, helping me refine research questions and overcome obstacles, and providing constructive feedback at every stage. They have generously offered their help whenever I needed it, whether academically or personally. Their consistent encouragement, especially during difficult moments in the early years, has always inspired me to persevere and be open to new possibilities. I feel extremely fortunate to have been guided by and learned from such supportive mentors.

I also want to express my gratitude to my dissertation committee members, Prof. Julian McAuley and Prof. Arya Mazumdar, whose feedback has greatly strengthened this dissertation and enriched my research through thoughtful discussions. I am also thankful to Prof. Andrew B. Kahng and Prof. Sanjoy Dasgupta for joining my research exam committee.

Throughout my Ph.D., I have been fortunate to collaborate with many talented people, and I am deeply grateful to all of them: Dezhi Hong, Xinyang Zhang, Xiyuan Zhang, Shuheng Li, Haiyu Huang, Zihan Wang, Xiaohan Fu, Xiaofan Yu, Junshen Xu, Bugra Can, and Yi Fan. Collaborating with you has made my Ph.D. experience both productive and enjoyable.

This period of my life has also been one of remarkable personal growth. I truly appreciate all my friends, whose companionship has brought happiness and balance to this challenging journey. I cherish the laughter, shared experiences, and countless memories that have made these years special.

Finally, my deepest gratitude goes to my family. To my parents, thank you for your unconditional love and support throughout my life, and for always believing in me, no matter the choices I made. To my husband, Chenyang An, thank you for walking this journey with me every

step of the way since we were sixteen. Life is full of the unexpected, and through it all, you have been there—figuring things out with me, lifting me up when I struggled, and celebrating every milestone. I am incredibly lucky to have you by my side.

I would like to extend my thanks to my co-authors for kindly granting permission to include our publications and materials in my dissertation.

Chapter 2 incorporates material from the publication "Navigating Alignment for Non-identical Client Class Sets: A Label Name-Anchored Federated Learning Framework", by Jiayun Zhang, Xiyuan Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang, published in Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3 incorporates material from the publication "REACT: Residual-Adaptive Contextual Tuning for Fast Model Adaptation in Threat Detection", by Jiayun Zhang, Junshen Xu, Bugra Can, Yi Fan, published in Proceedings of the ACM Web Conference 2025. The dissertation author was the primary investigator and author of this paper.

Chapter 4 incorporates material from the publication "How Few Davids Improve One Goliath: Federated Learning in Resource-Skewed Edge Computing Environments", by Jiayun Zhang, Shuheng Li, Haiyu Huang, Zihan Wang, Xiaohan Fu, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang, published in Proceedings of the ACM Web Conference 2024. The dissertation author was the primary investigator and author of this paper.

Chapter 5 incorporates material from the publication "Orthogonal Calibration for Asynchronous Federated Learning", by Jiayun Zhang, Shuheng Li, Haiyu Huang, Xiaofan Yu, Rajesh K. Gupta, Jingbo Shang, published as Preprint arXiv:2502.15940, 2025. The dissertation author was the primary investigator and author of this paper.

Chapter 6 incorporates material from the publication "Minimally Supervised Contextual Inference from Human Mobility: An Iterative Collaborative Distillation Framework", by Jiayun Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang, published in Proceed-

ings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI), 2023. The dissertation author was the primary investigator and author of this paper.

VITA

2020 B.S. in Computer Science, Fudan University

2025 Ph.D. in Computer Science, University of California San Diego

PUBLICATIONS

Jiayun Zhang, Shuheng Li, Haiyu Huang, Xiaofan Yu, Rajesh K. Gupta, Jingbo Shang. "Orthogonal Calibration for Asynchronous Federated Learning." Preprint arXiv:2502.15940, 2025.

Shuheng Li, **Jiayun Zhang**, Xiaohan Fu, Xiyuan Zhang, Jingbo Shang, Rajesh K. Gupta. "Matching Skeleton-based Activity Representations with Heterogeneous Signals for HAR." In Proceedings of the 23st ACM Conference on Embedded Networked Sensor Systems (SenSys), 2025.

Jiayun Zhang, Junshen Xu, Bugra Can, Yi Fan. "REACT: Residual-Adaptive Contextual Tuning for Fast Model Adaptation in Threat Detection." In Proceedings of the ACM Web Conference 2025.

Jiayun Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang. "Contextual Inference From Sparse Shopping Transactions Based on Motif Patterns." IEEE Transactions on Knowledge and Data Engineering (TKDE), 2025.

Chenyang An, Zhibo Chen, Qihao Ye, Emily First, Letian Peng, **Jiayun Zhang**, Zihan Wang, Sorin Lerner, Jingbo Shang. "Learn from Failure: Fine-tuning LLMs with Trial-and-Error Data for Intuitionistic Propositional Logic Proving" In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL), 2024.

Jiayun Zhang, Shuheng Li, Haiyu Huang, Zihan Wang, Xiaohan Fu, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang. "How Few Davids Improve One Goliath: Federated Learning in Resource-Skewed Edge Computing Environments." In Proceedings of the ACM Web Conference 2024.

Xiyuan Zhang, Xiaohan Fu, Diyan Teng, Chengyu Dong, Keerthivasan Vijayakumar, **Jiayun Zhang**, Ranak Roy Chowdhury, Junsheng Han, Dezhi Hong, Rashmi Kulkarni, Jingbo Shang and Rajesh K. Gupta. "Physics-Informed Data Denoising for Real-Life Sensing Systems." In Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems (SenSys), 2023.

Xiyuan Zhang, Ranak Roy Chowdhury, **Jiayun Zhang**, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang. "Unleashing the Power of Shared Label Structures for Human Activity Recognition." In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM), 2023.

Jiayun Zhang, Xiyuan Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang. "Navigating Alignment for Non-identical Client Class Sets: A Label Name-Anchored Federated Learning Framework." In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023.

Jiayun Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang. "Minimally Supervised Contextual Inference from Human Mobility: An Iterative Collaborative Distillation Framework." In Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI), 2023.

Huiying Li, Shawn Shan, Emily Wenger, **Jiayun Zhang**, Haitao Zheng, Ben Y. Zhao. "Blacklight: Scalable Defense for Neural Networks against Query-Based Black-Box Attacks." In Proceedings of the 31st USENIX Security Symposium (USENIX Security), 2022.

Shawn Shan, Emily Wenger, **Jiayun Zhang**, Huiying Li, Haitao Zheng, Ben Y. Zhao. "Fawkes: Protecting Privacy against Unauthorized Deep Learning Models." In Proceedings of the 29th USENIX Security Symposium (USENIX Security), 2020.

Qingyuan Gong, **Jiayun Zhang**, Yang Chen, Qi Li, Yu Xiao, Xin Wang, Pan Hui. "Detecting Malicious Accounts in Online Developer Communities Using Deep Learning." In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM), 2019.

Qingyuan Gong, **Jiayun Zhang**, Xin Wang, Yang Chen. "Identifying Structural Hole Spanners in Online Social Networks Using Machine Learning." In Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos.

ABSTRACT OF THE DISSERTATION

Effective Learning in Heterogeneous Distributed Environments

by

Jiayun Zhang

Doctor of Philosophy in Computer Science

University of California San Diego, 2025

Professor Rajesh K. Gupta, Chair
Professor Jingbo Shang, Co-Chair

Machine learning applications have widely expanded from single, isolated systems to complex distributed environments, such as mobile computing, sensor networks, and healthcare systems. These environments often include thousands of nodes (e.g., edge devices) that collect data and perform local training or inference, exhibiting considerable heterogeneity in data and computing resources. The heterogeneity poses great challenges for training machine learning models across distributed systems.

This dissertation discusses practical scenarios for deploying machine learning in distributed environments. We introduce new methods for effective learning under heterogeneity,

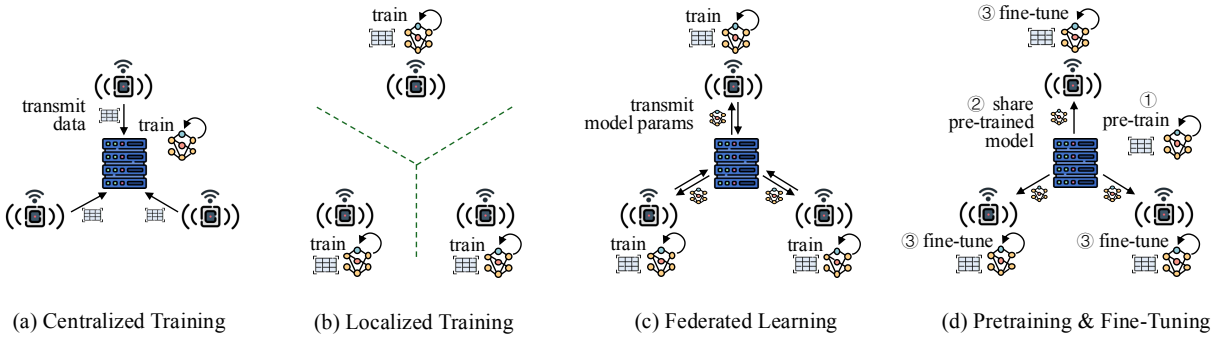
focusing on three dimensions: robustness, scalability, and quality. In the first part, we address data heterogeneity by leveraging contextual cues to align and adapt models, ensuring robustness to variations in distributed training data. In the second part, we manage system heterogeneity by developing knowledge aggregation methods that enable nodes with varying capacities to collaborate inclusively and efficiently. In the third part, we develop methods for deriving implicit contextual information from data, which is essential for finding correlations among distributed domains and enhancing model quality. The proposed approaches are designed to be model-agnostic, supporting various applications and system configurations. Extensive evaluations demonstrate that our methods achieve state-of-the-art performance across a wide range of real-world applications, including image and language processing, human sensing and mobile computing tasks, such as healthcare and activity recognition. By addressing different heterogeneity scenarios, our methods improve the global model trained on heterogeneous data sources by 6.14%, enhance node-specific adapted models by up to 14.85%, boost performance across nodes with diverse capacities, and accelerate training by $12\times$. The contributions in this dissertation enhance the practicality of Artificial Intelligence (AI) at the edge, facilitating the implementation of ubiquitous intelligent systems for seamless assistance.

Chapter 1

Introduction

Artificial Intelligence (AI) has demonstrated strong capabilities in complex tasks across various domains, assisting in numerous everyday and industrial applications. As a result, machine learning models are increasingly integrated into daily life and production processes, which are no longer confined to isolated systems but are deployed in ubiquitous, distributed environments. For example, large-scale sensor networks can continuously collect environmental data from multiple locations, which is then processed by machine learning algorithms deployed on edge devices for analytics and inference.

The real-world digital systems today have several notable characteristics. First, most systems are ***distributed*** in nature. These systems consist of interconnected nodes that operate in different locations, generating vast amounts of data and performing computations at varying scales of time and distances. For instance, a city-wide intelligent transportation system may connect thousands of vehicles and roadside infrastructure, collecting data about traffic flow, road conditions, and accidents from different geographical areas, each feeding information into machine learning pipelines to facilitate decision-making and enhance traffic safety. Second, these systems show considerable ***heterogeneity***. The heterogeneity arises from two key sources: data and compute. From the data perspective, devices often produce information with different feature and label distributions. From the computing perspective, resources vary widely, ranging from high-performance servers with powerful GPUs to resource-constrained edge devices or



(a) Centralized Training (b) Localized Training (c) Federated Learning (d) Pretraining & Fine-Tuning

Figure 1.1. Four common learning paradigms in distributed environments.

smartphones that can only train or store small models. Third, these systems involve interactions with users, introducing an additional layer of variability in inputs. Unlike traditional programs that execute deterministic steps, machine learning algorithms in distributed environments must account for changing usage patterns and dynamic environmental factors that affect both data generation and computational demands.

An effective machine learning system maintains reliable and high-quality performance despite the abovementioned variations. It is crucial for real-world deployments, as it ensures that models remain functional despite evolving conditions, such as new data distributions, changing compute resources, or unexpected user behaviors.

1.1 Learning Paradigms in Distributed Environments

There are multiple approaches for training machine learning models across distributed nodes. Each approach entails different trade-offs regarding data and model movement, scalability, and overall model performance. We discuss four main paradigms that are common in distributed environments. Figure 1.1 illustrates the key features.

Centralized Training. A traditional strategy aggregates all data from distributed nodes into a central data center, and trains a unified model on high-performance computing machines. In this way, the local device does not incur computational costs as most of the computations are done on one or several centralized servers. However, it can be infeasible in applications where

Table 1.1. Comparison of learning paradigms in distributed environments.

Paradigm	Data Transfer	Model Transfer	Scalability	Generalization
Centralized Training	High	None	Low	High
Localized Training	None	None	High	Low
Federated Learning	None	High	High	High
Pretraining & Fine-Tuning	Moderate	Moderate	High	High

transferring large volumes of data is costly or restricted by legal and organizational guidelines. Moreover, handling inference centrally can impose additional communication overhead and latency if local devices must transmit inference data and wait for results.

Localized Training. Localized training isolates the operations on each individual node. In this scenario, both training and inference happen on each device locally, without transferring data or models to a central server. While this minimizes the cost of information exchange, it often results in poor generalization since each node has limited or non-representative data, and there is no mechanism to benefit from knowledge accumulated by other nodes.

Federated Learning. Federated learning offers a balance between isolation and collaboration by keeping raw data on devices while sharing parameters of locally maintained and global models. Each node updates a local model on its private data, and these updates are periodically aggregated to form a more powerful global model, which is then redistributed for further local training [138]. This approach allows distributed nodes to learn from broader data distributions without direct data sharing. It is effective in leveraging the collective knowledge from a network of nodes.

Pretraining and Fine-Tuning. The process involves pretraining a model on a large aggregated or representative dataset (which could be publicly available or sourced from a smaller subset of devices), which is then distributed to individual nodes. Pretraining provides a strong baseline model that can be efficiently adapted to local domains, thereby addressing data scarcity on individual nodes and improving accuracy [244]. Local nodes then fine-tune the pretrained model with their smaller, task-specific datasets, improving performance in local contexts. This

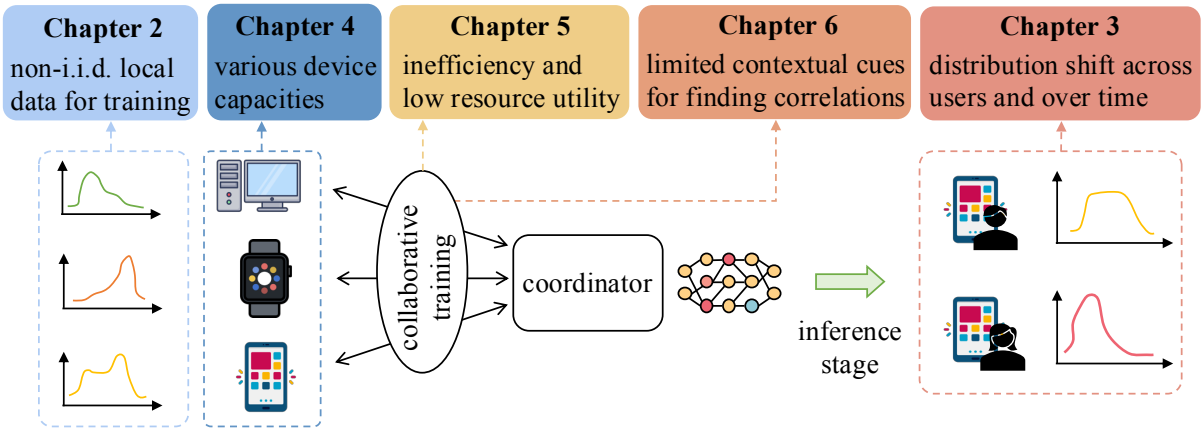


Figure 1.2. Major challenges in training and deploying machine learning models in distributed environments arise from heterogeneity in data and computing.

approach reduces the computational burden on resource-constrained devices by shifting much of the training complexity to the pretraining phase, and it also mitigates challenges of data scarcity at individual nodes as discussed in *localized training*.

As summarized in Table 1.1, federated learning and pretraining with subsequent fine-tuning are effective for handling data scarcity and heterogeneity in distributed environments. This dissertation focuses on these two paradigms to develop strategies that achieve robust performance across diverse devices and statistical environments. Furthermore, these two learning paradigms can be combined, with pretraining taking place within the federated learning process.

1.2 Challenges

Compared to centralized settings where data and computing resources are pooled in a single location, distributed environments introduce a range of additional complexities. As depicted in Figure 1.2, these complexities broadly revolve around data and computing.

Generalization Under Statistical Heterogeneity. In distributed environments, data generated by individual nodes or users can differ significantly in underlying statistical properties such as quantity, labels, features, labels, and modalities. Unlike centralized learning, which can mix all data to reduce bias, training models in distributed environments must deal with localized

data sources that may exhibit skewed or non-i.i.d. characteristics. This statistical heterogeneity makes the convergence harder and degrades model performance. Moreover, limited visibility into raw data—due to bandwidth constraints or organizational policies—further complicates collective learning. The key research question is: *How to train a robust machine learning model under statistical heterogeneity (without direct access to training data), but generalize well across diverse distributions?*

Evolving Environments and Data Distribution Shift. Over time, data distributions may shift due to changing user behaviors, environmental factors, or evolving operational conditions (e.g., sensor drift or dynamic network conditions). These shifts may be subtle or abrupt, making it challenging for a model trained on prior distribution to remain accurate without retraining. However, new data are often limited or arrive sporadically on distributed nodes, constraining how quickly a model can adapt to unforeseen patterns. Naive retraining would overfit to limited recent observations. The key research question is: *How can a model adapt to distribution shifts in heterogeneous environments when only a limited amount of new data is available?*

Heterogeneous Device Capacities. Distributed computing nodes exhibit varying device capabilities, including computational power, memory capacity, network bandwidth, etc. For example, a company wants to develop a machine learning system in collaboration with its end users. The company trains a large model using a vast dataset gathered from controlled environments, while its users can contribute by sharing local models trained on their personal data in the wild. The enterprise-grade computing machines can load and train large models, while the users' weak mobile devices can only train small models. Training large-scale models across this heterogeneous spectrum requires coordination and aggregation methods to ensure that each device actively contributes and benefits from others. The key research question is: *How can collaborative training frameworks work under system heterogeneity to achieve scalability and strong performance inclusively across diverse devices?*

Inefficiency and Low Resource Utility. Communication and synchronization overhead account for a substantial portion of training time in distributed learning frameworks, particularly

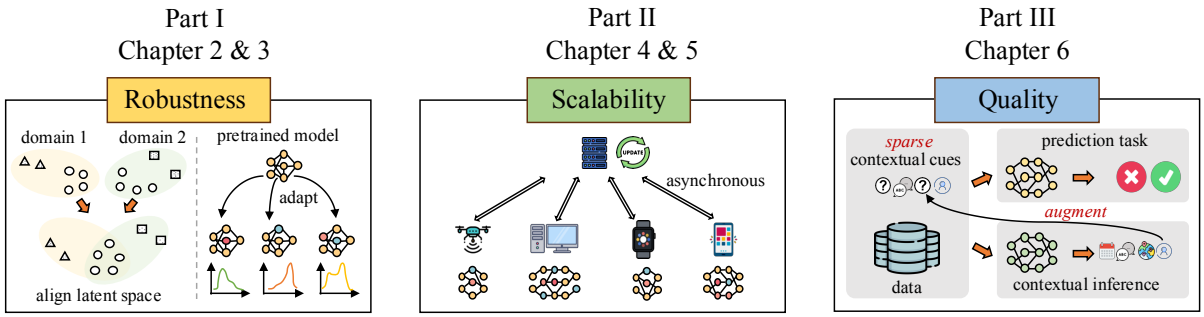


Figure 1.3. Dissertation outline. The first part addresses data heterogeneity to enhance robustness against variations in distributed training data. The second part manages system heterogeneity to enable collaboration among nodes with varying capacities and improve scalability. The third part describes ways of deriving contextual information from data, enhancing the quality of learning.

when devices exhibit uneven compute speeds or network conditions. Straggler nodes may delay global updates, and naive synchronous approaches can stall progress if even one device lags significantly. The key research question is: *How can the training process be accelerated in the presence of heterogeneous device resources while maintaining convergence quality?*

Limited Contextual Cues for Finding Correlations Across Nodes. Contextual information, such as demographic information, environmental conditions, or device usage patterns, can improve model performance and robustness as it provides additional information about underlying data patterns and correlations among domains [243]. For instance, in healthcare applications, knowledge of patient demographics and regional environmental factors can assist personalized diagnoses; usage patterns in distributed sensor networks offer insights into resource scheduling. Yet, in many distributed systems, such contextual cues are often sparse or inconsistently shared [235], limiting opportunities for effective knowledge transfer among disparate nodes. The key research question is: *How can contextual information be obtained to enhance the quality of learning in distributed environments?*

1.3 Dissertation Outline

We address the effectiveness in three aspects: robustness, scalability, and quality. This dissertation presents methods to address the core challenges of machine learning in heterogeneous

distributed environments. We propose five components, categorized into three parts, as depicted in Figure 1.3.

- **Part I (Chapters 2–3):** Focuses on *statistical heterogeneity* across devices and domains, involving learning paradigms of federated learning, pre-training, and fine-tuning. The proposed methods describe how to leverage expressive natural language class names [236] (Chapter 2) and other contextual cues such as temporal factors [233] (Chapter 3) to align and adapt models under varying data distributions in different stages, promoting model robustness.
- **Part II (Chapters 4–5):** Addresses *system heterogeneity*, including diverse device capacities and network constraints. We introduce a method [231] for representing different model architectures via graph modeling and enable collaboration among devices of varying capacities inclusively (Chapter 4). We further develop an asynchronous update scheme [232] to accelerate training (Chapter 5). These solutions extend the insights from Part I, enhancing scalability and efficiency in large-scale deployments.
- **Part III (Chapter 6):** Explores *contextual inference* as an auxiliary approach to link heterogeneous domains across distributed nodes. The techniques in Part I and Part II leverage various forms of contextual information (e.g., domain semantics, temporal factors); however, such information may be sparse or implicit in practice. Therefore, in this part, we present a contextual inference method [234] to uncover contextual information from data. The inferred contextual information can then be integrated into the previous components to ensure high-quality performance.

1.4 Contributions and Key Outcomes

We summarize below the contributions and key findings of each component of this dissertation:

- FEDALIGN (Chapter 2) presents an approach for handling non-identical client class sets in federated learning, where different clients may hold different or even non-overlapping classes. To

address this data heterogeneity, FEDALIGN aligns the latent spaces across clients by modeling the natural language class names and knowledge distillation. We evaluate FEDALIGN on real-world datasets of different applications (e.g., behavioral context recognition, text classification, medical code prediction), including both single-label and multi-label classification problems. FEDALIGN demonstrates state-of-the-art performance compared to various federated classification methods. On average, it improves accuracy by 6.14% over the best-performing baseline across all datasets.

- REACT (Chapter 3) proposes a model adaptation method that addresses the distribution shift problem from both pretraining and fine-tuning steps. REACT decomposes model weights into meta and adaptive components and updates the two components through meta-learning. By integrating a hypernetwork to generate adaptive weights based on data and contextual information, REACT enables knowledge sharing and adjusts model weights for new distributions with few fine-tuning efforts (i.e., a few gradient updates on a small set of new samples). We evaluate the effectiveness of REACT on two applications, network intrusion detection and malware detection, with scenarios where distribution shifts occur over time or across domains. REACT demonstrating substantial performance gains, with up to a 14.85% improvement in the area under the receiver operating characteristic curve (AUROC), compared to existing solutions.
- RECIPFL (Chapter 4) is the first work to investigate federated learning in resource-skewed environments, where a small number of powerful devices train large models while many resource-constrained devices train small models. Prior methods often fail to guarantee mutual benefits for all device types. RECIPFL employs a graph hypernetwork at the server, trains it to generate personalized model parameters for client models with different neural network architectures, and effectively transfers knowledge across these models. Our method supports arbitrary model scaling strategies, making it practical for real-world deployment. Our evaluations show RECIPFL improves accuracy by 4.5% for strong devices and 7.4% for weak devices, demonstrating that all types of devices can contribute and benefit from collaborative

training.

- ORTHOFL (Chapter 5) is an asynchronous update approach for accelerating federated learning under heterogeneous device capacities and network conditions. ORTHOFL maintains separate weights for global and local models to account for inconsistent global and local objectives in heterogeneous data environments. It introduces an orthogonal weight calibration mechanism, which projects global weight shifts onto subspaces orthogonal to the client's local updates, thereby mitigating interference and preserving meaningful contributions from asynchronous local and global progress. Based on our evaluation, ORTHOFL presents an average of 9.6% accuracy improvement over synchronous methods and a 12× speedup, and consistently outperforms existing asynchronous baselines under diverse delay patterns and heterogeneity scenarios.
- STCOLAB (Chapter 6) presents a collaborative distillation method for contextual inference under the constraint of minimal supervision, where only a few labeled samples per class are available. STCOLAB separates individual modules for different modalities (e.g., spatial and temporal). It sequentially trains two modules in multiple iterations and distills knowledge between them. In this way, it mutually calibrates these modules and combines complementary insights from each modality. Experiments on two real-world mobility datasets demonstrate that, with a small number of labeled samples per class (e.g., 10), STCOLAB accurately infers important demographic attributes, yielding a 4.74% increase in micro-F1 over the existing semi-supervised baseline.

Collectively, these contributions promote the deployment of machine learning in practical distributed environments. The methodologies presented in this dissertation apply to diverse tasks, including image and text classification, activity recognition, anomaly detection, and medical care. By addressing both statistical and system heterogeneity, the proposed methods advance the effectiveness of machine learning solutions in modern digital systems.

Part I

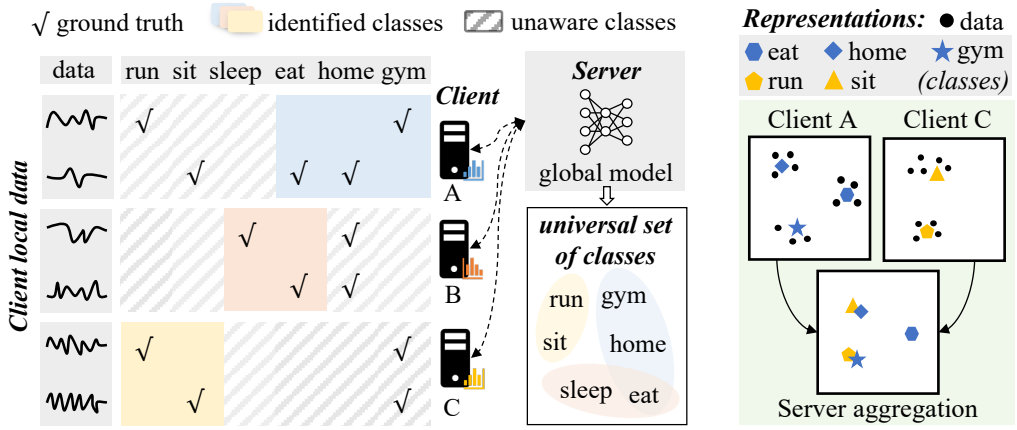
Addressing Statistical Heterogeneity

Chapter 2

Navigating Feature Alignment in Distributed Machine Learning

In this chapter, we address the challenge of statistical heterogeneity in distributed machine learning environments. We introduce an approach that leverages language semantics to align feature spaces across distributed nodes, mitigating inconsistencies that arise from non-independent and identically distributed (non-IID) data. While this heterogeneity problem is pervasive across many learning paradigms, we begin by focusing on federated learning as a practical case study.

Traditional federated classification methods, even those designed for non-IID clients, assume that each client annotates its local data with respect to the same universal class set. In this work, we focus on a more general yet practical setting, *non-identical client class sets*, where clients focus on their own (different or even non-overlapping) class sets and seek a global model that works for the union of these classes. If one views classification as finding the best match between representations produced by data/label encoder, such heterogeneity in client class sets poses a new significant challenge—local encoders at different clients may operate in different and even independent latent spaces, making it hard to aggregate at the server. We propose a novel method, FEDALIGN, to align the latent spaces across clients from both label and data perspectives. From a label perspective, we leverage the expressive natural language class names as a common ground for label encoders to anchor class representations and guide the data encoder learning across clients. From a data perspective, during local training, we regard the global class



(a) The problem setting of non-identical client class sets. **(b)** Unique challenge.

Figure 2.1. Illustrations of our problem setting and unique challenge of misaligned latent spaces across clients, using a behavioral context recognition system where users have different preferences in reporting (i.e., annotating) labels.

representations as anchors and leverage the data points that are close/far enough to the anchors of locally-unaware classes to align the data encoders across clients. Our theoretical analysis of the generalization performance and extensive experiments on four real-world datasets of different tasks confirm that FEDALIGN outperforms various state-of-the-art (non-IID) federated classification methods.

2.1 Introduction

Federated learning [138] allows multiple parties to collaboratively learn a global model effective for all participants while preserving the privacy of their local data. It brings benefits to various domains, such as recommendation systems [119, 128, 219], ubiquitous sensing [105, 189, 103] and mobile computing [216, 102, 108].

Existing federated classification methods [111, 109, 80, 198, 120, 199, 251, 132] typically assume that the local annotations at each client follow the same set of classes; however, this assumption does not hold true in many real-world applications. For example, a smartwatch company wants to build a human activity classifier for all activity types, as shown in Figure 2.1a. Although their smartwatch users as clients could experience almost all types of daily activities,

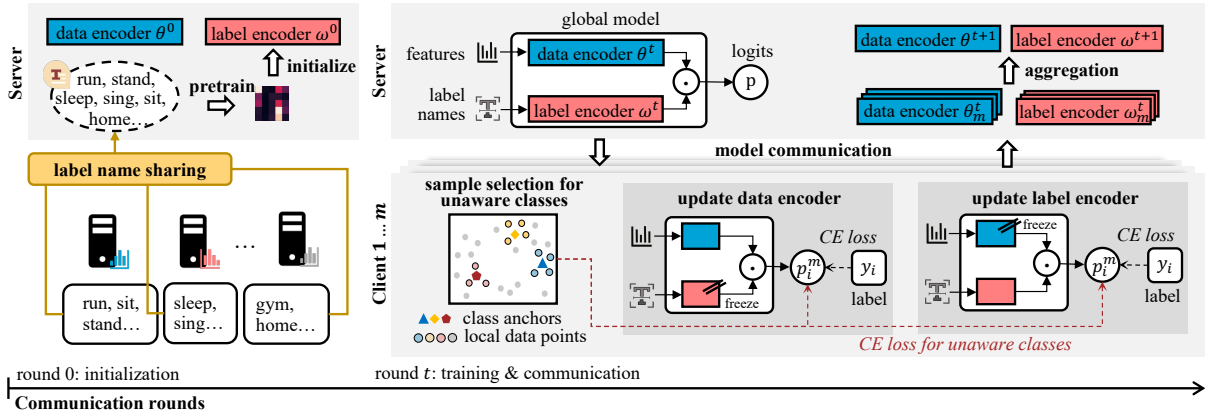


Figure 2.2. Overview of FEDALIGN. The label names are leveraged as a common ground for label encoders to anchor class representations. During local training, the two encoders perform alternating training to mutually regulate the latent spaces. The global class representations are regarded as class anchors. Pseudo-labels are assigned to partially-unlabeled local samples for unaware classes based on their distances to the anchors. An additional cross-entropy loss for unaware classes is added to the local learning objective to reduce the divergence between global and local distributions.

each user may only opt to report (i.e., annotate) a subset of activities. Another example is a federated medical diagnosis system, which attempts to infer all types of diseases of a patient for comprehensive health screening. Physicians and specialist groups with different expertise can participate in this federated learning system as clients. As one can see here, different specialists will only offer disease annotations within their domains, even if a patient may have several types of diseases at the same time. This makes the class sets at many clients non-identical and even non-overlapping.

We aim to lift this assumption and work on a general and rather practical federated learning setting, **non-identical client class sets**, where clients focus on their own (different or even non-overlapping) class sets and seek a global classification model that works for the union of these classes. We denote the classes that are not covered in the local annotations as *locally-unaware classes*. Note that each client can have local data whose true labels are among the locally-unaware classes. Also, the classification task here can be either single-label or multi-label. When it is multi-label, the local data might be only partially labeled due to the

locally-unaware classes. Therefore, this new setting is more general and challenging than the missing class scenario [113] which assumes the single-label scenario and no local data is from locally-unaware classes.

The non-identical client class sets pose a significant challenge of huge variance in local training across different clients. As shown in Figure 2.1b, one can view classification as a matching process between data representations and label representations in a latent space. Because of the non-identical client class sets, locally trained classifiers are more likely to operate in drastically different latent spaces. Moreover, when the class sets are non-overlapping, it is possible that the latent spaces at different clients are completely independent. This would result in inaccurate classification boundaries after aggregation at the server, making our setting more challenging than non-IID clients with identical client class sets.

We propose a novel federated learning method FEDALIGN, as shown in Figure 2.2, to align the latent spaces across clients from both label and data perspectives as follows:

(1) *Anchor the label representations using label names*. We observe that the natural-language class names (i.e., label names) often carry valuable information for understanding label semantics, and more importantly, they are typically safe to share with all parties. Therefore, we break the classification model into a data encoder and a label encoder as shown in Figure 2.2, and then leverage the label names as the common ground for label encoders. The server initializes the label encoder with pretrained text representations, such as word embedding. The label encoder will be then distributed to different clients and updated alternatingly with data encoders during local training and global aggregation, mutually regulating the latent space.

(2) *Connect the data representations via anchors of locally-unaware classes*. During local training, we regard the global class representations as anchors and utilize data points that are close/far enough to the anchors of locally-unaware classes to align the data encoders. Specifically, as shown in Figure 2.2, at each client, we annotate local data based on their distances to the anchors and add another cross-entropy loss between the pseudo-labels and

the model predictions. Such regularization encourages the data encoders to reside in the same latent space.

Our theoretical analysis shows that FEDALIGN can achieve a better generalization bound than traditional federated learning methods, suggesting a strong potential for performance improvement. Experiments on four real-world datasets, including the most challenging scenario of multi-label classification and non-overlapping client class sets, confirm that FEDALIGN outperforms various state-of-the-art (non-IID) federated classification methods. Our contributions are summarized as follows:

- We propose a more general yet practical federated classification setting, namely non-identical client class sets. We identify the new challenge caused by the heterogeneity in client class sets — local models at different clients may operate in different and even independent latent spaces.
- We propose a novel method FEDALIGN to align the latent spaces across clients for both label and data.
- Our generalization bound analysis and extensive experiments on four real-world datasets of different tasks confirm the superiority of FEDALIGN over various state-of-the-art (non-IID) federated classification methods both theoretically and empirically.

2.2 Preliminaries

Problem Formulation. We aim to generate a global classification model using federated learning with non-identical class sets, where each client only identifies part of the classes from its dataset. Denote the universal set of classes as $\mathbf{C}$, the set of classes that are identified on client m is $\mathbf{C}_m$, and the set of locally-unaware classes is $\overline{\mathbf{C}_m}$, where $\mathbf{C}_m \cup \overline{\mathbf{C}_m} = \mathbf{C}$. The goal is to learn a global model $g : \mathscr{X} \to \{0,1\}^{|\mathbf{C}|}$ that given $x \in \mathscr{X}$, all positive labels from $\mathbf{C}$ can be inferred.

The training set on client m is denoted as $\mathbf{D}_m = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i is the input data and $y_i = [y_{i,c}]_{c \in \mathbf{C}}$ is a vector showing the labels of each class. If $c \in \mathbf{C}_m$, $y_{i,c} \in \{0,1\}$. If $c \in \overline{\mathbf{C}_m}$, $y_{i,c}$ is unknown. It is possible that some data samples $x_i \in \mathbf{D}_m$ do not belong to any of the classes

in $\mathbf{C}_m$, i.e., $\forall c \in \mathbf{C}_m : y_{i,c} = 0$.

Backbone Classification Model. Let $\mathscr{Z} \subset \mathscr{R}^d$ be the latent feature space and $\mathscr{Y}$ be the output spaces. Generally, the classification model g can be decomposed into a data encoder $f : \mathscr{X} \to \mathscr{Z}$ parameterized by θ and a linear layer (i.e., classifier) $h : \mathscr{Z} \to \mathscr{Y}$ parameterized by ζ. The data encoder f generates representations for input data. Then, the classifier h transforms the representations into prediction logits. Given an input x_i, the predicted probability given by g is $g(x_i; \theta, \zeta) = \sigma(h(f(x_i; \theta); \zeta))$, where σ is the activation function. We discuss two types of classification tasks as follows.

Single-Label Multi-Class Classification. In this setting, each sample is associated with only one positive class. In other words, the classes are mutually exclusive. We use softmax activation to get the predicted probability. The class with the maximum probability is predicted as the positive class. Let $g(x_i; \theta, \zeta)_c$ denote the predicted probability of x_i belonging to class c. During training, the cross-entropy loss is used as the loss function:

$$\ell(g(x_i; \theta, \zeta), y_i) = -\sum_{c \in \mathbf{C}_m} y_{i,c} \log g(x_i; \theta, \zeta)_c. \tag{2.1}$$

Multi-Label Classification. In this setting, each sample may be associated with a set of positive classes. For example, a person may have both diabetes and hypertension. The sigmoid activation is applied to get the predicted probability. Each element in the predicted probability represents the probability that the input data x_i is associated with a specific class. The final predictions are achieved by thresholding the probabilities at 0.5. If $g(x_i; \theta, \zeta)_c > 0.5$, x_i is predicted to be associated with class c. During training, the binary cross-entropy loss is used as the loss function:

$$\begin{aligned} \ell(g(x_i; \theta, \zeta), y_i) = -\sum_{c \in \mathbf{C}_m} [y_{i,c} \log g(x_i; \theta, \zeta)_c \\ + (1 - y_{i,c}) \log(1 - g(x_i; \theta, \zeta)_c)]. \end{aligned} \tag{2.2}$$

Federated Learning. Consider a federated learning system with M clients. The server

coordinates M clients to update the model in T communication rounds. The learning objective is to minimize the loss on every client, i.e., $\min_{\theta,\zeta} \frac{1}{M}\sum_{m\in[M]} \mathscr{L}_m(\theta,\zeta)$. At each round, the server sends the model parameters to a subset of clients and lets them optimize the model by minimizing the loss over their local datasets. The loss at client m is:

$$\mathscr{L}_m(\theta,\zeta) = \mathbb{E}_{(x_i,y_i)\sim \mathbf{D}_m} \ell(g(x_i;\theta,\zeta),y_i). \tag{2.3}$$

At the end of each round, the server aggregates the model parameters received from clients, usually by taking the average.

2.3 Methodology

Algorithm 1: Pseudo-code of FEDALIGN

Input : Communication rounds T, number of selected clients per round $|\mathbf{S}_t|$, local training epochs E.

Output : The final global model $g(x;\theta^T,\omega^T)$.

Server executes:

Collect label names from clients and pretrain text representations to initialize label encoder ω^0;

Randomly initialize data encoder θ^0;

for $t = 0,1,\ldots,T-1$ **do**

  Select a subset $\mathbf{S}_t$ of clients at random;

  for $m \in \mathbf{S}_t$ **do**

    $\theta_m^{(t+1)}, \omega_m^{(t+1)} \leftarrow$ **ClientUpdate**$(m, \theta^{(t)}, \omega^{(t)})$;

  $\theta^{(t+1)} \leftarrow \frac{\sum_{m\in\mathbf{S}_t} \theta_m^{(t+1)}}{|\mathbf{S}_t|}$; $\omega_c^{(t+1)} \leftarrow \frac{\sum_{m\in\mathbf{S}_t, c\in\mathbf{C}_m} \omega_{m,c}^{(t+1)}}{|\{m|m\in\mathbf{S}_t, c\in\mathbf{C}_m\}|}$;

return θ^T, ω^T;

ClientUpdate$(m, \theta^{(t)}, \omega^{(t)})$:

$\theta_m^{(t)}, \omega_m^{(t)} \leftarrow \theta^{(t)}, \omega^{(t)}$;

for $e = 1,2,\ldots,E$ **do**

  Calculate distances of data points and class anchors and form dataset $D_m'^{(t)}$;

  Alternatingly update $\theta_m^{(t+1)}$ and $\omega_m^{(t+1)}$;

return $\theta_m^{(t+1)}$, and $\omega_m^{(t+1)}$ to server ;

2.3.1 Overview

The pseudo-code of FEDALIGN can be found in Algorithm 1. Learning with FEDALIGN consists of the following steps:

1. **Label name sharing and label encoder initialization:** Before training, the server collects the natural language label names from the clients. The server initializes the label encoder's parameters ω^0 via pretrained text representations, such as word embedding.

2. **Client selection and model communication:** At t-th round, the server randomly selects a subset of clients $\mathbf{S}_t$ and sends the global model parameters to them.

3. **Local training:** Client $m \in \mathbf{S}_t$ independently trains its local model and returns the model parameters.

4. **Model aggregation:** The server aggregates the parameters of client models into global parameters.

Pretraining text representations and label encoder initialization in (1) are conducted only once at the beginning. Steps (2)-(4) repeat for T rounds until the global model converges. During local training in (3), each client $m \in \mathbf{S}_t$ conducts the following steps:

(a) **Select samples for unaware classes via class anchors:** Client m forms a dataset $\mathbf{D}_m'^{(t)}$ for locally-unaware classes $\overline{\mathbf{C}_m}$ by using the latest class representations as anchors and computing the distances to the data representations.

(b) **Alternating training of two encoders:** Client m freezes the label encoder and updates the data encoder. Then, it freezes the data encoder and updates the label encoder.

(c) **Model communication after local updates:** Client m sends the updated model parameters to the server.

2.3.2 Label Name-Anchored Matching

The vanilla model described in Section 2.2 learns feature spaces merely based on local training data with numerical label IDs. However, with non-identical client class sets, local models at different clients are likely to form different and even independent feature spaces, making the classification boundaries aggregated at the server inaccurate. To better align the feature spaces, we leverage the semantics of label names as a common reference to anchor class representations. The natural language label names carry valuable information for understanding label correlations. For example, in behavioral context recognition, the activity of "lying down" is likely to indicate the person is "sleeping", and the possible location of the activity is "at home". Such knowledge about label correlations not only exists in the datasets to investigate, but can also be mined through analyzing the semantics of label names.

Incorporating Label Encoder to Classification Model. We replace the classifier in a conventional classification model with a label encoder as shown in Figure 2.2. Let $\mathscr{W}$ be the set of natural language label names with respect to $\mathbf{C}$, and $\mathscr{Z}$ be the latent feature space. The new classification model $g = f \circ \gamma$ consists of two branches: a data encoder $f : \mathscr{X} \to \mathscr{Z}$ parameterized by θ and a label encoder $\gamma : \mathscr{W} \to \mathscr{Z}$ parameterized by ω. The $\circ$ is the operation to get dot product. The label encoder $\gamma(w_c; \omega)$ (parameterized by ω) takes the label names $w_c \in \mathscr{W}$ as inputs and maps them into representations. Prior knowledge about label semantics can be inserted into the label encoder by initializing it with pretrained label embeddings. Below, we introduce our implementation of pretraining by modeling label co-occurrence. We expect more advanced techniques like pretrained neural language models could further enrich the semantic label embedding, but we leave it as future work.

Semantic Label Embedding Pretraining. Inspired by existing works that learn semantic word embeddings based on word-word co-occurrence [20] and point-wise mutual information (PMI) [153, 101], we use an external text corpus related to the domain of the classification task to extract knowledge of label co-occurrence and pretrain label embeddings for initializing the

label encoder. Before collaborative training, each client shares the natural language names of its classes with the server. The server then searches for these label names in a large text corpus related to the domain of the classification task. It counts the occurrences of each label name w_i in the text segments (e.g., sentences or paragraphs). We notice that the label names can be phrases that contain multiple words, and the order of the words may change in different text segments while representing the same meaning. For example, "colon cancer" and "cancer of the colon" refer to the same concept. To handle such variations, we organize the label names into sets of words and use a sliding window of length L_w to scan the text segments. If the set of words in the label name is covered by the words within the sliding window, we consider the label name appears in the text segment. The length of the sliding window L_w varies per label name being searched. The co-occurrence of a pair of label names w_i and w_j is then calculated using the point-wise mutual information (PMI):

$$\text{PMI}(w_i, w_j) = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)},$$

where $p(w_i)$ and $p(w_j)$ are the individual distributions and $p(w_i, w_j)$ is the the joint distribution. The higher the $\text{PMI}(w_i, w_j)$, the stronger the association between the two label names w_i and w_j.

The server then learns semantic label embeddings based on the PMIs. The goal of label embedding learning is to learn a mapping function from label names to representations $\gamma : \mathscr{W} \to \mathscr{Z}$, which enforces labels with related semantic meanings to have similar representations. To achieve this, the server builds a label co-occurrence graph $G = \langle \mathbf{V}, \mathbf{E} \rangle$, where the nodes $\mathbf{V}$ represent the label names and the edges $\mathbf{E}$ represent the co-occurrence relationship between the nodes. The PMI values are zero-centered by subtracting the mean and are used as the edge weights between label names. Edges with negative weights are removed from the graph. For every source node $w \in \mathbf{V}$, we define $\mathbf{N}_s(w) \subset \mathbf{V}$ as its network neighborhood generated through simulating fixed-length random walks starting from w. The transition probability for random walk simulation is calculated by normalizing the edge weight. The objective function of label

embedding learning is defined as:

$$\max_{\gamma} \sum_{w \in \mathbf{V}} (-\log Z_w + \sum_{u \in \mathbf{N}_s(w)} \gamma(u) \cdot \gamma(w)),$$

where $Z_w = \sum_{v \in \mathbf{V}} \exp(\gamma(v) \cdot \gamma(w))$ and is approximated using negative sampling [140]. The mapping function γ is achieved by a single hidden layer feedforward neural network and the objective is optimized using stochastic gradient descent.

Representation Matching. Given an input x_i, the model uses the data encoder to generate its representation $f(x_i;\theta)$. Then, it takes the dot product of the data representation and every class representation. This way, it calculates the similarity between the input data and classes. An activation function σ is applied to the dot product to get the predicted probabilities of x_i:

$$g(x_i;\theta,\omega) = \sigma([f(x_i;\theta) \circ \gamma(w_c;\omega)]_{w_c \in \mathscr{W}}). \tag{2.4}$$

The choice of activation function is the same as defined in Section 2.2.

Alternating Encoder Training. With the new model design, we rewrite the learning objective in Equation 2.3 as:

$$\mathscr{L}_m(\theta,\omega) = \mathbb{E}_{(x_i,y_i)\sim \mathbf{D}_m} \ell[\sigma([f(x_i;\theta) \circ \gamma(w_c;\omega)]_{w_c \in \mathscr{W}}), y_i]. \tag{2.5}$$

The two encoders are two branches in the model. We want the representations obtained by one encoder to regulate the training of the other while preventing mutual interference. Therefore, at each local update step, we first fix the parameters in the label encoder and update the data encoder. Then, we fix the data encoder and update the label encoder. Let $\theta_{m,j}^{(t)}$ and $\omega_{m,j}^{(t)}$ be the parameters of the local data encoder and label encoder at j-th update step in t-th round and η be

the learning rate. The parameters are updated as:

$$\theta_{m,j+1}^{(t)} \leftarrow \theta_{m,j}^{(t)} - \eta \nabla_{\theta_{m,j}^{(t)}} \mathscr{L}_m(\theta_{m,j}^{(t)}, \omega_{m,j}^{(t)}), \tag{2.6}$$

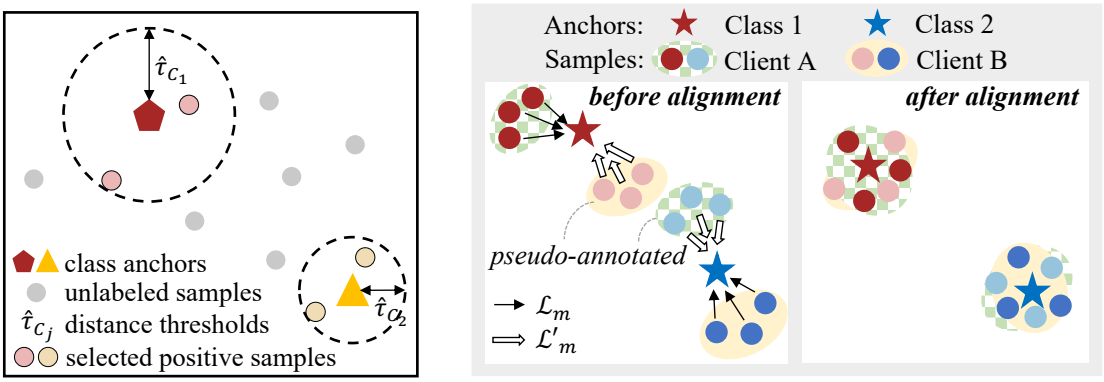
$$\omega_{m,j+1}^{(t)} \leftarrow \omega_{m,j}^{(t)} - \eta \nabla_{\omega_{m,j}^{(t)}} \mathscr{L}_m(\theta_{m,j+1}^{(t)}, \omega_{m,j}^{(t)}). \tag{2.7}$$

2.3.3 Anchor-Guided Alignment for Locally-Unaware Classes

Due to the lack of label information of certain classes to support supervision, the training at each client is biased toward the identified classes [132, 240]. To mitigate such drift, we further exploit the global class representations to assist the alignment for locally-unaware classes. Since we formulate the classification problem as a matching between representations of classes and local data at each client, the class representations produced by the global label encoder can reflect the global distribution. Therefore, we regard the global class representations as anchors and use them to identify features for unaware classes at each client. Specifically, at the beginning of each round of local training, the client measures the distances between class anchors and local data representations. The nearest and farthest samples from the anchors are annotated. An additional loss term is added to the local optimization objective to reduce the distribution mismatch. Compared with common practices of pseudo-labeling [137, 98] which assign labels based on model predictions, the annotations assigned by our anchor-guided method are independent of the biased classifier and are thus more reliable.

Deriving Class Anchors. When the client receives the parameters of the label encoder $\omega^{(t)}$ at t-th round, it uses the latest label encoder to derive the global class anchors: $\{\gamma(w_c; \omega^{(t)}) | w_c \in \mathscr{W}\}$.

Selecting Samples for Locally-Unaware Classes. Client m uses the received data encoder to generate representations of its local data: $\{f(x_i; \theta^{(t)}) | x_i \in \mathscr{X}_m\}$. Then, the client



(a) Selecting positive samples based on global class anchors

(b) Effect of matching and alignment by using the global class representations as anchors

Figure 2.3. (a) illustrates how positive samples are annotated for locally-unaware classes based on distances to class anchors. (b) shows the effect of matching and alignment.

calculates the cosine distance from every class anchor to the local data in latent space:

$$d_{i,c}^{(t)} = 1 - \frac{\gamma(w_c;\omega^{(t)}) \circ f(x_i;\theta^{(t)})}{\|\gamma(w_c;\omega^{(t)})\|_2 \cdot \|f(x_i;\theta^{(t)})\|_2}. \tag{2.8}$$

Then, the client annotates samples for the locally-unaware classes $\overline{\mathbf{C}_m}$ based on the distances. Samples with the closest distances to the class anchor $\gamma(w_c;\omega^{(t)})$ are annotated as positive samples of class c. Similarly, samples that are farthest from $\gamma(w_c;\omega^{(t)})$ are annotated as negative samples of c. The number of samples to be annotated depends on the percentile of distances. We define two thresholds, $\hat{\tau}_c^{(t)}$ and $\check{\tau}_c^{(t)}$, as the q_1-th and q_2-th percentile of the distances over all samples for annotating positive and negative samples respectively. The client annotates the samples whose distances are less than $\hat{\tau}_c^{(t)}$ as positive samples (i.e., $\tilde{y}_{i,c}^{(t)} = 1$) and those with distances greater than $\check{\tau}_c^{(t)}$ as negative samples (i.e., $\tilde{y}_{i,c}^{(t)} = 0$). Figure 2.3a shows an example of selecting positive samples for two classes. The dataset for alignment after the t-th round is as follows:

$$\mathbf{D}'^{(t)}_m \leftarrow \{(x_i, \tilde{y}_i^{(t)}) | d_{i,c}^{(t)} < \hat{\tau}_c^{(t)} \text{ or } d_{i,c}^{(t)} > \check{\tau}_c^{(t)}, c \in \overline{\mathbf{C}_m}\}. \tag{2.9}$$

For single-label classification, we add another constraint that a sample whose true label is not in $\mathbf{C}_m$ is annotated as a positive sample of class $c \in \overline{\mathbf{C}_m}$ only if c is the closest to it among all classes.

Alignment. The annotations for unaware classes are then used to guide the alignment at client m. We add an additional loss term to the local learning objective. The loss over $\mathbf{D'}_m^{(t)}$ is as follows:

$$\mathscr{L'}_m^{(t)}(\theta, \omega) = \mathbb{E}_{(x_i, \tilde{y}_i) \sim \mathbf{D'}_m^{(t)}} \ell' [\sigma([f(x_i; \theta) \circ \gamma(w_c; \omega)]_{w_c \in \mathscr{W}}, \tilde{y}_i], \tag{2.10}$$

where ℓ' represents the loss function with the same choice as defined in Equation 2.1 and 2.2. A slight difference is that ℓ' here is summed over $\overline{\mathbf{C}_m}$. Finally, the local learning objective is to jointly minimize Equation 2.5 and 2.10, i.e., $\min_{\theta,\omega}[\mathscr{L}_m(\theta, \omega) + \mathscr{L'}_m^{(t)}(\theta, \omega)]$. Figure 2.3b illustrates the effect of these two losses.

2.4 Analysis

In this section, we perform an analysis of the generalization performance of the aggregated model in federated learning.

Denote $\mathscr{D}$ as the global distribution on input space $\mathscr{X}$, and $\tilde{\mathscr{D}}$ as the induced global distribution over feature space $\mathscr{Z}$. Similarly, for the m-th client, denote $\mathscr{D}_m$ as the local distribution and $\tilde{\mathscr{D}}_m$ be the induced image of $\mathscr{D}_m$ over $\mathscr{Z}$. We review a typical theoretical upper bound for the generalization of global hypothesis [152, 251, 120]:

Theorem 2.4.1 (Generalization Bound of Federated Learning)**.** *Assume there are M clients in a federated learning system. Let* $\mathbf{H}$ *be the hypothesis class with VC-dimension d. The global hypothesis is the aggregation of* h_m*, i.e.,* $h = \frac{1}{M}\sum_{m\in[M]} h_m$*. Let* $\mathscr{L}(h)$ *denote the expected risk of h. With probability at least* $1-\delta$*, for* $\forall h \in \mathbf{H}$*:*

$$\begin{aligned} \mathscr{L}(h) \leq & \frac{1}{M}\sum\nolimits_{m\in[M]} \hat{\mathscr{L}}_m(h_m) + \frac{1}{M}\sum\nolimits_{m\in[M]} [d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}}) + \lambda_m] \\ & + \sqrt{\frac{4}{N}(d\,log\,\frac{2eN}{d} + log\,\frac{4M}{\delta})}, \end{aligned} \tag{2.11}$$

where $\hat{\mathscr{L}}_m(h_m)$ is the empirical risk on the m-th client given N observed samples, $d_{\mathbf{H}\Delta\mathbf{H}}(\cdot,\cdot)$ is the $\mathscr{A}$-distance that measures the divergence between two distributions based on the symmetric difference with respect to $\mathbf{H}$, λ_m is the risk of the optimal hypothesis over $\mathbf{H}$ with respect to $\mathscr{D}$ and $\mathscr{D}_m$, e is the base of the natural logarithm.

Theorem 2.4.1 applies to the traditional algorithm FedAvg [138], we observe two factors that affect the quality of the global hypothesis: the divergence between the local and global distributions $d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}})$ and the sample size N. Then, we discuss the generalization bound when FEDALIGN introduces empirical distributions for locally-unaware classes to align latent spaces.

Corollary 2.4.1.1 (Generalization Bound of Federated Learning with Mix-up Distributions)**.** *Let $\mathscr{D}'_m$ denote the distribution added for aligning the m-th client. Define the mix-up distribution $\mathscr{D}^*_m$ to be a mixture of the original local distribution $\mathscr{D}_m$ and $\mathscr{D}'_m$:*

$$\mathscr{D}^*_m = \alpha\mathscr{D}_m + (1-\alpha)\mathscr{D}'_m, \tag{2.12}$$

where $\alpha \in [0,1]$ is the weight of the original distribution, which is decided by the number of empirical samples added. Let $\mathbf{H}$ be the hypothesis class with VC-dimension d. The global hypothesis is the aggregation of h_m, i.e., $h = \frac{1}{M}\sum_{m\in[M]} h_m$. With probability at least $1-\delta$, for $\forall h \in \mathbf{H}$:

$$\begin{aligned}\mathscr{L}(h) \leq & \frac{1}{M}\sum\nolimits_{m\in[M]} \hat{\mathscr{L}}_m(h_m) \\ & + \frac{1}{M}\sum\nolimits_{m\in[M]}[\alpha d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}}) + (1-\alpha) d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}'_m, \tilde{\mathscr{D}}) + \lambda_m] \\ & + \sqrt{\frac{4}{N^*}(d\,log\,\frac{2eN^*}{d} + log\,\frac{4M}{\delta})},\end{aligned} \tag{2.13}$$

where $\hat{\mathscr{L}}_m(h_m)$ is the empirical risk on the m-th client given N^* ($N^* > N$) observed samples, e is the base of the natural logarithm.

Table 2.1. Dataset statistics. The imbalance factor refers to the ratio of the smallest class size to the largest class size.

Dataset	$\|C\|$	# of clients	Avg. N_m	Avg. $\|C_m\|$	Imbalance	Remarks
ES-5	51	5	5,446	10.2	0.0013	Multi-label, non-overlapping client class sets
ES-15		15	1,769	3.4		
ES-25		25	1,073	2.04		
MIMIC-III-10	50	10	807	5	0.1157	
PAMAP2-9	18	9	1,287	5	0.2049	Single-label
R8-8	8	8	617	3	0.0130	

By combining the local dataset with pseudo-annotated samples, FEDALIGN increases the sample size i.e., $N^* > N$, thus the last term of the bound becomes smaller. Second, given that the selected samples are in proximity to the anchors which are derived by the ensemble of the empirical distributions across all clients, the distribution derived via class anchors would exhibit lower divergence from the global distribution compared to the original local distribution i.e., $d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}'_m, \tilde{\mathscr{D}}) < d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}})$. The proof and more details are given in Appendix. Therefore, FEDALIGN can achieve a better generalization bound than traditional methods [138], suggesting a strong potential for performance improvement.

2.5 Experiments

2.5.1 Datasets

We conduct experiments on 6 datasets covering 4 different application scenarios and both single-label and multi-label classification problems. Table 2.1 offers an overview and the details are as follows.

1. **Behavioral Context Recognition.** The task is to infer the context of human activity. **ExtraSensory** [190] is a benchmark dataset for this task. The classes can be partitioned into 5 categories (e.g. location, activity, etc.). Based on ExtraSensory, we construct 3 datasets with non-overlapping client class sets. **ES-5**: We set 5 clients and every client only has annotations

from a different category (i.e., one category to one client). Training samples are then assigned to clients according to their associated classes. Since ExtraSensory is a multi-label dataset, we assign samples based on the most infrequent class among multiple labels to ensure each locally-identified class will have at least one positive sample. To make this dataset more realistic, we always assign all data of a subject to the same client. **ES-15 and ES-25**: We increase the number of clients to 15 and 25 to further challenge the compared methods. We start with the 5 class groups as ES-5 and iteratively split the groups until the number of class groups matches the number of clients. During every split, we select the group with the most classes and randomly divide it into two sub-groups. Every class group is visible and only visible to one client. One can then apply a similar process as ES-5 to assign training samples to clients.

2. **Medical Code Prediction.** Medical codes describe whether a patient has a specific medical condition or is at risk of development. The task is to annotate medical codes from clinical notes. We start with the MIMIC-III database [75] and follow the preprocessing method in [144] to form the benchmark MIMIC-III 50-label dataset. The classes span 10 categories in the ICD-9 taxonomy[1]. We construct **MIMIC-III-10** by partitioning the dataset into 10 clients following the same strategy as in ES-5.
3. **Human Activity Recognition.** The task aims at identifying the movement or action of a person based on sensor data. We start with the PAMAP2 [161] dataset, which collects data of physical activities from 9 subjects. We construct the **PAMAP2-9** dataset by regarding each subject as a client. For each client, we randomly select 5 classes to be its locally-identified classes.
4. **Text Classification.** We use the Reuters-21578 R8 dataset [21], which consists of news articles classified into 8 categories. We construct **R8-8** by randomly partitioning the data into 8 subsets and assigning one subset to each client. For each client, we randomly select 3

[1] the International Statistical Classification of Diseases and Related Health Problems (ICD): ftp://ftp.cdc.gov/pub/Health_Statistics/NCHS/Publications/ICD-9/ucod.txt

classes to be the identified classes.

2.5.2 Text Corpus for Semantic Label Embedding Pretraining

Domain-specific raw corpora are often readily available. For example, novel books that depict daily scenes can be used for understanding human activities. Academic journals are good study resources for understanding concepts in different professional domains. We use the following corpora for applications in our experiments:

- **BookCorpus** [250] is a large collection of free novel books collected from the internet. It is a popular text corpus in the field of natural language processing and has contributed to the training of many influential language models such as BERT [32] and GPT [19].
- **PubMed Open-Access (OA) subset** [148] is a text archive of journal articles and preprints in biomedical and life sciences. It has been widely used for biomedical text mining [99].
- **CommonCrawl (CC) News dataset** [57] is an English-language news corpus collecting news articles published between 2017 to 2019 from news sites worldwide.

We use PubMed-OA as the text corpus for medical code prediction, BookCorpus for behavioral context recognition and human activity recognition, and CC News for text classification. We consider a sentence as a text segment in BookCorpus, an article as a text segment in CC News, and a paragraph as a text segment in PubMed-OA.

2.5.3 Compared Methods

We compare FEDALIGN with classical [138] and state-of-the-art federated learning methods for non-IID data [111, 109, 80] as follows.

- **FedAvg** [138] is a classical federated learning algorithm where the server averages the updated local model parameters in each round to obtain the global model.
- **FedProx** [111] enforces a L_2 regularization term in local optimization which limits the distance between global and local models.

- **MOON** [109] adds a contrastive loss term to maximize the consistency of representations learned by the global and local models and minimize the consistency between representations learned by the local models of consecutive rounds.

- **Scaffold** [80] maintains control variates to estimate the update directions of global and local models. The drift in local training is approximated by the difference between the update directions. This difference is then added to the local updates to mitigate drift.

We also compare two state-of-the-art methods designed for federated classification with missing classes. Specifically,

- **FedRS** [113] is designed for the missing class scenario where each client owns data for part of the classes (i.e., *locally-identified classes* in our terminology). It restricts the weight update of missing classes by adding scaling factors to the softmax operation.
- **FedPU** [121] addresses scenarios where clients annotate only a small portion of their data, and unlabeled data exists for both locally identified and unaware classes. It leverages the labeled data at every client to estimate the misclassification loss of unaware classes from other clients and incorporates this estimated loss into local optimization objectives.

2.5.4 Experimental Setup

Base Neural Network Model. For a fair comparison, we use the same model setting for all compared methods. The data encoder is based on the Transformer architecture [192] with one encoder layer. There are 4 attention heads, and the dimension of the feed-forward network is 64. The label encoder is a single hidden layer neural network. The dimension d of representations is 256. Since the size of the label encoder is equivalent to the classifier layer in the conventional classification model, there is no extra overhead during model communication in FEDALIGN. Additionally, when considering future work involving the use of advanced neural language models as the label encoder, we can train only the adapter module [63], i.e., adding a small number of new parameters to the pretrained language model. The adapters are transferred

Table 2.2. Main experimental results (% averaged over 5 runs). ES-5, ES-15, ES-25 and MIMIC-III-10 are multi-label datasets where class sets across clients have no overlap. PAMAP2-9 and R8-8 are single-label datasets where client class sets overlap.

Method	ES-5		ES-15		ES-25		MIMIC-III-10		PAMAP2-9		R8-8	
	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc
FedAvg [138]	28.77	80.79	22.71	62.44	19.52	50.42	35.04	67.07	68.89	71.45	78.51	92.76
FedProx ($\mu = 0.001$) [111]	29.26	80.67	22.42	61.91	19.48	51.16	30.57	61.76	69.70	73.63	79.05	92.61
FedProx ($\mu = 0.0001$) [111]	28.53	79.14	22.52	61.95	19.05	52.33	33.73	64.31	69.38	71.78	75.98	91.78
MOON ($\mu = 0.001$) [109]	29.12	81.00	22.84	62.53	19.52	50.24	34.62	66.34	71.70	74.25	79.26	93.07
Scaffold [80]	28.14	77.13	23.15	61.69	19.73	48.81	33.58	62.84	73.57	75.60	82.83	94.43
FedPU [121]	27.95	79.82	22.27	59.22	16.97	34.59	33.23	62.83	85.50	87.66	83.06	94.17
FedRS ($\alpha = 0.5$) [113]	28.01	78.72	22.50	62.09	19.44	51.41	34.82	66.80	68.70	71.42	80.10	92.74
FedRS ($\alpha = 0.9$) [113]	28.25	79.25	22.55	62.17	19.40	50.87	35.44	67.45	71.81	74.44	76.68	91.81
FEDALIGN	**30.19**	**84.05**	**23.36**	**73.61**	**20.80**	**67.12**	**37.97**	**74.37**	**87.21**	**88.14**	**83.76**	**94.92**

and aggregated, while the other layers remain fixed at all parties.

Evaluation Metrics. Due to label imbalance, we adopt both accuracy and F1-score to evaluate the performance. They are often used as benchmark metrics for the datasets and tasks in our experiments [190, 46, 161, 155]. We calculate the metrics for each class and report the macro-average. All experiments are repeated 5 times with a fixed set of random seeds for all compared methods.

Train/Test Split. We set aside a portion of the dataset for testing the global model. For MIMIC-III and R8, we use the data split provided by the dataset. For the other datasets, we use 20% of the data for testing and distribute the rest of the data to clients for training.

Federated Learning Setting. For ES-5, ES-15, ES-25, PAMAP2-9 and R8-8, we run $T = 50$ rounds. For MIMIC-III-10, we run $T = 100$ rounds as it takes longer to converge. The number of selected clients per round is $|S_t| = 5$ and the local epochs $E = 5$. Note that we conduct sensitivity analysis in Section 2.5.7 and show the conclusion of the results is robust to the value of $|S_t|$ and E.

Hyperparameters. For the compared methods, we try different values for the hyper-parameters μ in FedProx and MOON, and α in FedRS, that are often adopted in previous work [111, 109, 113]. The values are displayed alongside the method name in Table 2.2.

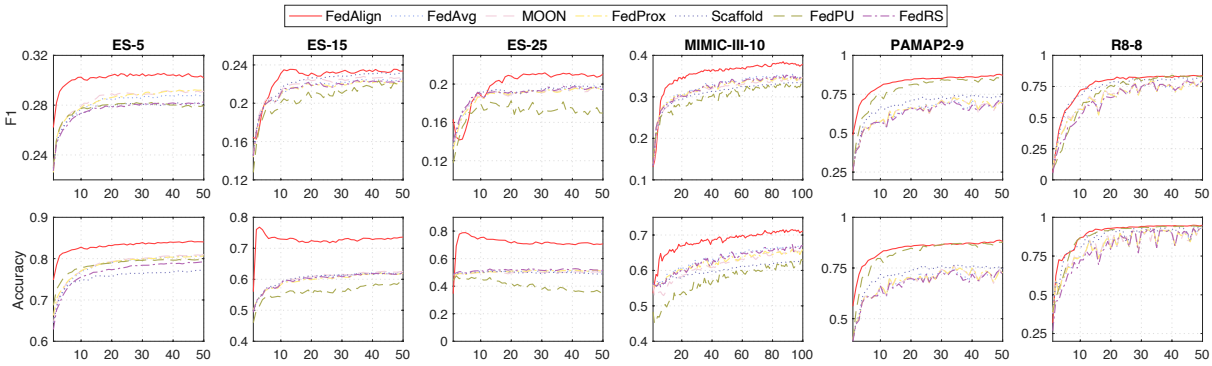


Figure 2.4. Performance w.r.t. communication rounds on six datasets. The results are averaged over 5 runs.

2.5.5 Main Results and Analysis

Multi-Label, Non-overlapping Client Class Sets. Table 2.2 shows the results. As one can clearly see, FEDALIGN always yields better performance than the baseline methods. Remarkably, with non-identical client class sets, the three state-of-the-art algorithms designed to deal with non-IID data (i.e., FedProx, MOON, and Scaffold) do not guarantee improvement over FedAvg (e.g., Scaffold loses to FedAvg on ES-5). In addition, although FedRS and FedPU are designed for missing class scenarios, their mechanisms are specifically tailored for single-label classification. In the context of multi-label classification, the label of one class does not indicate the labels of other classes, and the weight update of a class is solely influenced by its own features. Therefore, the scaling factors in FedRS and the misclassification loss estimation in FedPU become ineffective.

Single-Label, Non-identical but Overlapping Client Class Sets. FEDALIGN outperforms the baselines on both applications. The non-IID problems that FedRS and FedPU aim to tackle (i.e., missing class scenario, and positive and unlabeled data) are slightly different from ours. Although they show improvements over FedAvg and methods designed for the typical non-IID setting (i.e., FedProx, MOON, and Scaffold), FEDALIGN shows better performance compared with FedRS and FedPU in the problem of non-identical client class sets.

Performance w.r.t. Communication Rounds. Figure 2.4 shows the test performance

Table 2.3. F1-score (% averaged over 5 runs) of ablation studies.

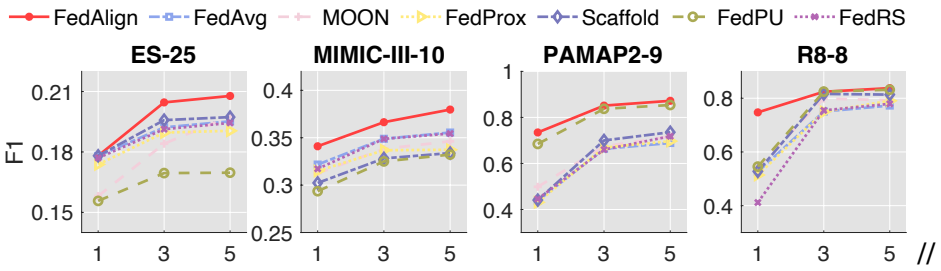
Method	ES-25	MIMIC-III-10	PAMAP2-9	R8-8
FedAvg	19.52	35.04	68.89	78.51
FEDALIGN w/o AL	19.57	37.87	70.87	79.67
FEDALIGN w/o SE	19.62	34.91	83.39	82.37
FEDALIGN w/o AT	20.28	37.09	86.01	83.22
FEDALIGN	**20.80**	**37.97**	**87.21**	**83.76**

with respect to communication rounds. FEDALIGN shows its advantage from the early stages of training. This indicates the pretrained text representations provide good initialization for the label encoder to guide the alignment of latent spaces. We do notice a decrease in the F1-score of FEDALIGN on ES-25 during initial rounds. This can be attributed to the noise in pseudo annotations for locally-unaware classes due to the undertrained encoders. However, as the training progresses, the quality of the pseudo annotations improves, leading to enhanced performance.

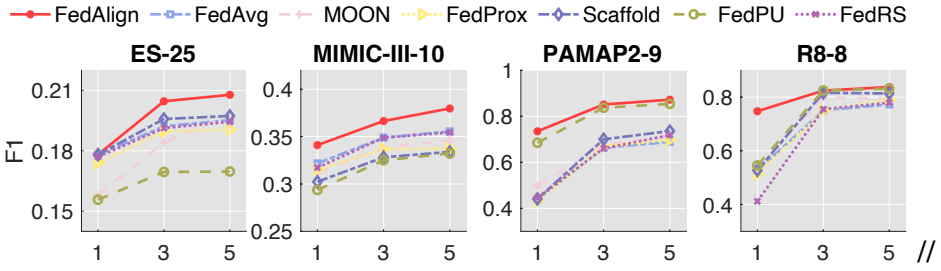
2.5.6 Ablation Studies

We conduct ablation studies to evaluate the contribution of each design in FEDALIGN. First, we evaluate the performance of the method without alignment for locally-unaware classes (denoted as **FEDALIGN w/o AL**). The classification model consists of a data encoder and a label encoder and the algorithm conducts alternating training of the two modules. Second, we evaluate the performance of the method without the semantic label name sharing (denoted as **FEDALIGN w/o SE**). In this case, the dataset for alignment is formed by annotating the samples according to prediction confidence given by the latest global model. For locally-unaware classes, samples with high prediction confidence are pseudo-annotated, and the confidence thresholds are decided by the same percentile values as in FEDALIGN. Third, we evaluate the performance of the method without alternating training (denoted as **FEDALIGN w/o AT**) which updates label and data encoders simultaneously.

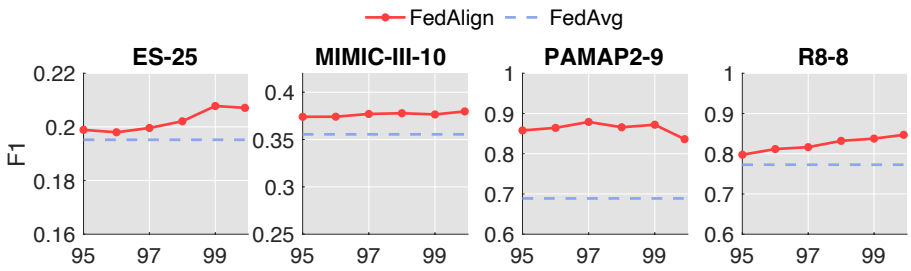
Since the model aggregation method in FEDALIGN is based on FedAvg (i.e., averaging



(a) Performance w.r.t. Participating Clients Per Round



(b) Performance w.r.t. Local Training Epochs



(c) Performance w.r.t. Distance Threshold

Figure 2.5. Sensitivity analyses of FEDALIGN.

the model parameters), we also compare FedAvg as the baseline method. Table 2.3 shows the F1-scores. We notice the performance decreases when removing any of the designs. This suggests the designs in FEDALIGN all contribute to improvement, and combining them can produce the best performance.

2.5.7 Sensitivity Analysis

Participating Clients Per Round. The number of participating clients in each round (i.e., $|S_t|$) has an effect on the speed of convergence [112]. We vary $|S_t|$ from 1 to 5 and compare FEDALIGN with all baseline methods. The comparisons in F1-score are shown in Figure 2.5a. We observe that FEDALIGN can always outperform the baseline methods under different values

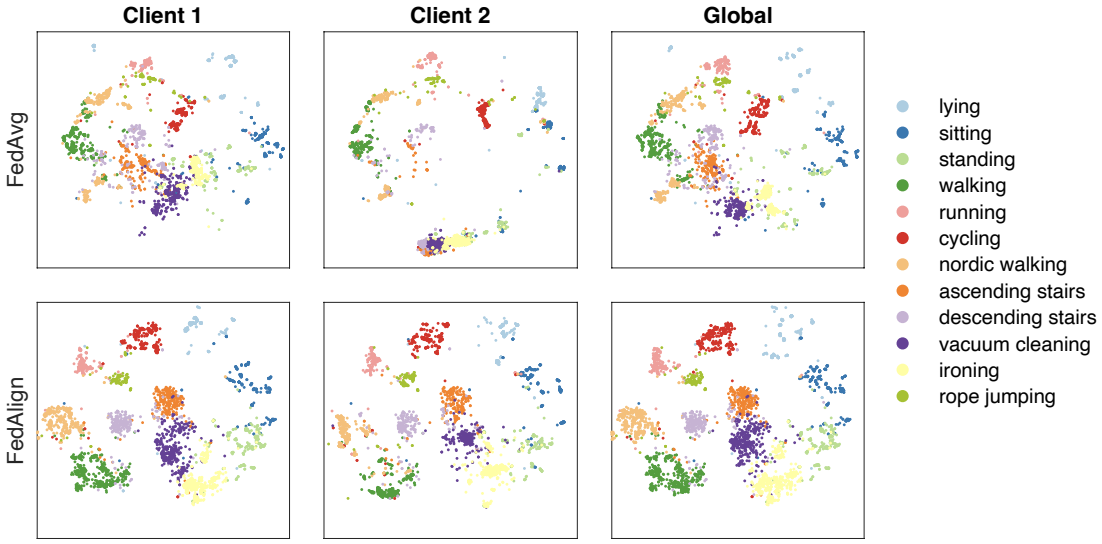


Figure 2.6. Data representations generated by two local models and the global model on the testing set of PAMAP2-9.

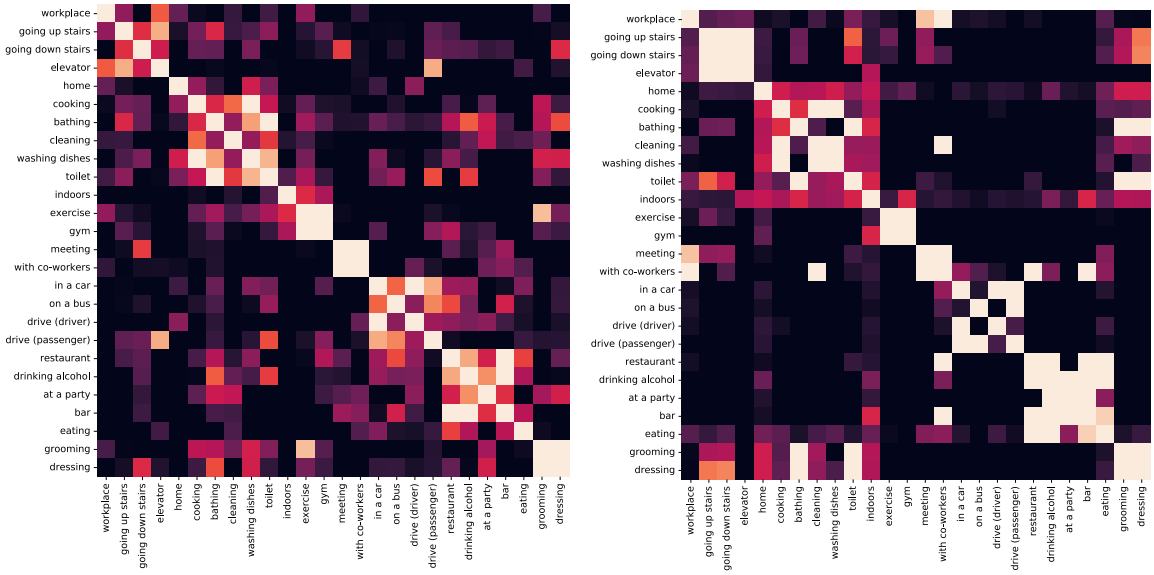
of $|S_t|$.

Local Training Epochs. We vary the local training epochs from 1 to 5 and compare the performance of FEDALIGN with all baseline methods. The comparisons are shown in Figure 2.5b. We see that FEDALIGN has consistently better performance than the baselines.

Distance Threshold for Selecting Samples for Unaware Classes. In Section 2.3.3, we set the threshold for assigning labels to samples for locally-unaware classes based on distance percentiles. To test the robustness of FEDALIGN to this hyperparameter, we vary the threshold for annotating positive samples by using different percentiles (95 to 99.9). Figure 2.5c shows the result. We see that FEDALIGN only needs a very small amount of pseudo annotations to have significant improvements over FedAvg. Notably, samples closer to the class anchors exhibit a higher likelihood of being accurately annotated, providing better guidance for alignment.

2.5.8 Case Studies

Visualization of Feature Latent Spaces. We visualize the learned data representations in PAMAP2-9. We generate the data representations on the testing set by the global model and the



(a) Similarity of Class Representations

(b) PMI of Labels in Centralized Dataset

Figure 2.7. (a) shows cosine similarities among class representations of ES-25 learned via FEDALIGN. (b) demonstrates the PMI of labels in the centralized dataset as a reference of ground truth. Brighter colors indicate higher similarity/PMI.

local models of two participating clients after 50 communication rounds. The locally-identified classes at the two clients are {walking, running, cycling, ironing, rope jumping} and {walking, lying, sitting, standing, vacuum cleaning} respectively. There are one overlapping class and four client-exclusive classes per client. We use t-SNE [191] to project the representations to 2-dimensional embeddings and compare the learned representations by FedAvg and FEDALIGN. In order to see if the representations generated by different client models are aligned by classes, for each algorithm, we gather the data representations generated by the client models and the global model together to perform the t-SNE transformation. The visualization is shown in Figure 2.6. We position them in the same coordinates. When training via FedAvg, we observe that the data representations of the same class generated by the two local models are likely to fall into different locations in the latent space. This suggests that the latent spaces of the two clients are misaligned, leading to less discriminability among data representations from different classes in the global latent space after model aggregation. On the contrary, when training via

FEDALIGN, the data representations of the same class generated by the two local models have similar locations in latent space. In addition, the data representations learned by FEDALIGN have clearer separations than those learned by FedAvg.

Similarity Among Class Representations. We then analyze the similarities among the class representations of ES-25 learned via FEDALIGN. Recall that ES-25 is the multi-label classification task where the class sets at different clients are non-overlapping. We use the label encoder from the global model trained after 50 rounds to generate class representations. For a clear view of group similarities, we apply Spectral Clustering [221] to cluster the class representations and sort the label names based on the assigned clusters. We visualize the cosine similarities of a subset of the classes as shown in Figure 2.7a, where brighter colors indicate higher similarity. The observed similarity patterns in the class representations conform with our knowledge about what contexts of daily activities often happen together or not. For example, the representations of the classes, "toilet" and "bathing", "meeting" and "with co-workers", "gym" and "exercise" have higher similarity, while they have less similarity with other classes. To provide a reference for ground truth, we calculate the PMI of labels based on their co-occurrence in the centralized dataset to indicate how strong the association is between every two classes. We show the results in Figure 2.7b. The brighter the color, the higher the PMI (i.e., the two classes have a stronger association). The order of the classes is the same as in Figure 2.7a. We observe the two figures display similar patterns of associations among classes. Although the class sets of different clients are non-overlapping, the label encoder trained via FEDALIGN successfully captures associations among classes across clients.

2.6 Related Work

Federated Learning with Non-IID Data. One of the fundamental challenges in federated learning is the presence of non-IID data [78]. The reasons and solutions to this challenge are being actively explored. Common solutions involve adding local regularization [111, 109, 80],

improving server aggregation [198, 120, 199], and leverage public dataset [120] or synthesized features [132, 251] to calibrate models. These methods tackle more relaxed non-IID problems that assume clients have the same set of classes. As shown in our experiments, these baselines show marginal improvements over FedAvg when the clients have unaware classes. Some recent works [113, 121] consider the problem of clients having access to only a subset of the entire class set. For example, FedRS [113] addresses the case where each client only owns data from certain classes. FedPU [121] focuses on the scenario where clients label a small portion of their datasets, and there exists unlabeled data from both positive (i.e., *locally-identified* in our terminology) and negative (i.e., *locally-unaware*) classes. The problem settings differ from ours. Moreover, these methods are specifically tailored for single-label classification, where the presence of one class indicates the absence or presence of other classes. When applied to our problem, they demonstrate less improvement compared to FEDALIGN.

Label Semantics Modeling. In tasks where some of the label patterns cannot be directly observed from the training dataset, such as zero-shot learning [97], it is hard for the model to generalize to unseen classes. To deal with the problem, several methods are proposed to leverage prior knowledge such as knowledge graphs [194] or model semantic label embedding from textual information about classes [100, 136, 212, 158]. For example, Ba et al. [100] derived embedding features for classes from natural language descriptions and learned a mapping to transform text features of classes to visual image feature space. Radford et al [158] used contrastive pretraining to jointly train an image encoder and a text encoder and predict the correct pairings of image and text caption, which helps to produce high-quality image representations. Matsuki et al [136] and Wu et al [212] incorporate word embeddings for zero-shot learning in human activity recognition. These methods show the potential of using semantic relationships between labels to enable predictions for classes not observed in the training set, which motivates our design of semantic label name sharing.

2.7 Summary

We studied the problem of federated classification with non-identical class sets. We propose FEDALIGN and demonstrate its use in federated learning for various applications. FEDALIGN incorporates a label encoder in the backbone classification model. Semantic label learning is conducted by leveraging a domain-related corpus and shared label names. The pretrained semantic label embeddings contain the knowledge of label correlations and are used to guide the training of the data encoder. Moreover, the anchor-guided alignment enriches features for unaware classes at each client based on global class anchors and reduces the discrepancy between local distributions and global distribution. These two designs are key to mitigating client variance in FEDALIGN, which addresses the challenge of non-identical class sets. We show that FEDALIGN improves the baseline algorithms for federated learning with non-IID data and achieves new state-of-the-art. It is worth mentioning that FEDALIGN can work when clients can only share the label IDs by assuming label names are unknown and randomly initializing the label encoder. Of course, advanced techniques like neural language models can be applied to generate and enrich the label representations [242], and we leave it as future work.

Chapter 2 incorporates material from the publication "Navigating Alignment for Non-identical Client Class Sets: A Label Name-Anchored Federated Learning Framework", by Jiayun Zhang, Xiyuan Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang, published in Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023. The dissertation author was the primary investigator and the lead author of this paper.

Chapter 3

Model Adaptation Under Distribution Shift

Building upon the foundation of federated learning introduced in the previous chapter, we now address a follow-up challenge: distribution shift during inference. Discrepancies between the training and test data distributions often emerge during deployment, compromising model performance. In this chapter, we propose a solution that leverages meta-learning to prepare the model for adaptation and employs fine-tuning with limited new data and contextual cues to handle distribution shifts effectively. This method is discussed within the broader context of the pretraining and fine-tuning paradigm. As a key application, we explore anomaly detection, a domain particularly susceptible to distribution shifts, to demonstrate the effectiveness of our approach in maintaining model robustness under changing conditions.

Web and mobile systems show constant distribution shifts due to the evolvement of services, users, and threats, severely degrading the performance of threat detection models trained on prior distributions. Fast model adaptation with minimal new data is essential for maintaining reliable security measures. A key challenge in this context is the lack of ground truth, which undermines the ability of existing solutions to align classes across shifted distributions. Moreover, the limited new data often fails to represent the underlying distribution, providing sparse and potentially noisy information for adaptation. In this work, we propose REACT, a novel method that adapts the model using a few unlabeled data and contextual insights. We leverage the

inherent data imbalance in threat detection and meta-train weights on diverse unlabeled subsets to generalize common patterns across distributions, eliminating the reliance on labels for alignment. REACT decomposes a neural network into two complementary components: meta weights as a shared foundation of general knowledge, and residual adaptive weights as adjustments for specific shifts. To compensate for the limited availability of new data, REACT trains a hypernetwork to predict adaptive weights based on data and contextual information, enabling knowledge sharing across distributions. The meta weights and the hypernetwork are updated alternately, maximizing both generalization and adaptability. Extensive experiments across multiple datasets and models demonstrate that REACT improves AUROC by 14.85% over models without adaptation, outperforming the state-of-the-art.

3.1 Introduction

Threat detection is an essential component in web and mobile systems that identifies malicious activities across networks, endpoints, and software, defending against security risks. Cyber environments undergo continual distribution shifts due to various factors, including users joining and leaving the network, user behavior changes, and software updates. These shifts severely degrade the performance of threat detection models trained on prior distributions. For example, during special events on the Web, such as major sales promotions, there is often a surge in users visiting the site and subscribing to services, and many of them may cancel the subscription and reduce their activity after the event, causing abrupt shifts in network traffic. Threat detection models trained on typical traffic are less effective in these scenarios. Adapting model weights after minimal exposure to new data is crucial for timely and effectively identifying threats in dynamic and adversarial environments.

A critical challenge in managing distribution shifts in threat detection is the lack of ground truth, as this requires user reports or detailed inspections by domain experts. Traditional methods [13, 41, 129, 172, 178, 184, 210], which rely on labels from either source or target

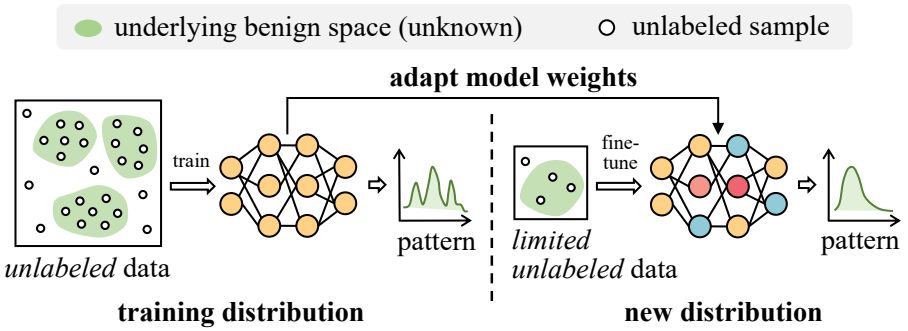


Figure 3.1. Illustration of our problem setting of model adaptation.

domains to align classes across shifted distributions, fail to address this scenario. Moreover, the limited new data often does not fully represent the underlying distribution, providing sparse and potentially noisy information for adaptation. Existing methods [35, 92, 104, 116] fine-tune models exclusively on these limited data. They may exhibit large variations in performance and are sensitive to the quality of the observed data, often prone to overfitting [67, 197, 234, 248]. The problem setting is illustrated in Figure 3.1.

Due to label scarcity, threat detection often exhibits extreme data imbalance, with a few suspicious activities (e.g., unauthorized access attempts, anomalous traffic, malware) hidden among a vast majority of benign behaviors. This imbalance presents an opportunity to address distribution shifts. Instead of learning the exact benign and suspicious behaviors and matching them across domains, models could learn to generalize majority patterns for various distributions. Samples that deviate from such patterns are regarded as potential threats [2, 59]. Therefore, we apply meta-learning on diverse unlabeled subsets dominated by benign samples. These subsets are sampled according to underlying shifts, e.g., time-based sampling for temporal shifts. After being meta-trained on various scenarios, the model can quickly adapt to new distributions by adjusting the learned pattern using a few unlabeled samples.

To compensate for the limited new data, we utilize contextual information to model correlations among distributions. For example, in network intrusion detection, a newly deployed service like a microservices-based API can find similarities with common web servers (e.g.,

Apache and Tomcat) based on their intrinsic characteristics of services and deployment environments (e.g., service configuration, user role). By modeling these contexts, we can recognize relationships across distributions and transfer knowledge from mature systems to newly deployed ones.

Building on these insights, we introduce REACT (Residual-Adaptive Contextual Tuning), a novel method adapting models with a few unlabeled new data and contextual insights. Given a neural network, REACT decomposes its weights into the sum of two components: meta weights, which are meta-trained to form a solid foundation of general knowledge and shared globally, and adaptive weights, which are the residual components fine-tuned to specific distributions. We leverage a hypernetwork [55] to generate adaptive weights based on data and contextual information. Intuitively, the hypernetwork maps its inputs onto a low-dimensional manifold within the parameter space [26, 170]. This mapping positions adaptive weights for similar contexts and data patterns close to each other, enabling knowledge transfer across different distributions. During training, REACT optimizes the meta weights and the hypernetwork alternately through meta-learning on subsets sampled according to underlying shifts. At inference, the adaptive weights are fine-tuned from the prediction given by the trained hypernetwork, while the meta weights are fixed, preserving the generalizability of the model [27, 107, 135, 188].

We theoretically analyze the convergence of REACT on linear models, showing the parameters converge at a linear rate characterized by the eigenvalues of the sample matrices and other hyperparameters in REACT. Our approach is model-agnostic, broadly applicable to various neural networks and loss functions. We evaluate REACT on three datasets with different backbone models. Compared to models without adaptation, REACT improves the AUROC by 14.85% with few fine-tuning efforts (e.g., update 1 to 10 gradient steps on 10 to 100 samples). Sensitivity analyses show that REACT is robust to variations in the number of samples, the number of fine-tuning steps, and contamination in training data. We further showcase the capability of REACT for parameter-efficient fine-tuning, achieving 5.75% higher AUROC with 94.3% fewer parameters updated compared to full fine-tuning, highlighting its

efficiency. Our contributions are as follows:

- We study the problem of fast model adaptation under distribution shift in threat detection, focusing on a practical yet challenging scenario where labels are unavailable and only limited data from new distributions are observed.
- We introduce REACT, a novel adaptation method using a few unlabeled data and contextual insights. REACT decomposes model weights into meta and adaptive components and updates them through meta-learning alternately. It employs a hypernetwork to generate adaptive weights based on data and contexts, enabling knowledge transfer across distributions.
- We establish the convergence rate of REACT through theoretical analysis. Moreover, we conduct extensive evaluations on multiple model architectures and datasets, demonstrating that REACT consistently outperforms various state-of-the-art methods.

3.2 Related Work

Threat Detection. Threat detection [14, 126, 182, 183, 224, 48] aims to identify security risks in systems and networks, such as insider threat [126, 224], intrusion attack [34], malware [106], spammer [182]. Typically, the ground truths for benign and malicious activities are not available, as they require user reports or inspections by domain experts. As a result, threat detection follows unsupervised or semi-supervised approaches in anomaly detection based on different assumptions of data distribution [2, 59]. These methods assume the majority of data belong to a "normal" class, while anomalies deviate from this norm, e.g., lying in low-density regions [252], being far from normal data clusters [163], or showing high reconstruction errors from the latent space of normal data [2]. These methods are designed for static environments and are not robust to distribution shifts. Changes in data distributions can significantly degrade model performance.

Distribution Shifts in General Machine Learning. Distribution shift means the distributions of the training and testing data are different, leading to poor model generalization to unseen

data [50]. To address the challenge, adaptation methods have been proposed [41, 129, 210]. We focus on works designed for unsupervised or semi-supervised scenarios due to the data specificity in threat detection. Unsupervised domain adaptation [13] is a closely related topic, which adapts models to target domains that have no labeled data. Methods include invariant representation learning [178], prototype-oriented conditional transport [184], contrastive pre-training [172]. However, these methods assume the availability of labels from source data to guide adaptation, falling short in threat detection where labels in source domains are also unavailable.

Model Adaptation in Threat Detection. Distribution shifts are observed in threat detection, such as malware detection [77], network intrusion detection [24, 217], and log anomaly detection [73]. Traditional supervised approaches [11, 77, 151] require extensive labeling, making them impractical for real-world deployment. Recent efforts have recognized this limitation and have been exploring adaptation approaches without relying on labels. Unsupervised domain adaptation methods, like learning domain-invariant representations [22] have been extended. However, they typically require simultaneous training on source and target domains, making them less suitable for emerging domains. Test-time adaptation, such as batch normalization updates [104], energy-based models [205], and trend estimation [81], updates models during inference without gradient descent. Though efficient, they are limited to minor shifts [52] or sequential shifts that display continuous patterns [81]. To address more severe and random shift, meta-learning [43, 186] is a promising approach that trains a meta model on a variety of learning tasks, enabling adaptation to new distributions with a small amount of data. Prior works have applied meta-learning to graph neural networks [35] and autoencoders [92] for few-shot detection, and introduced prototype-oriented optimal transport for adapting models to new multivariate time-series [116]. However, these methods fine-tune models solely on limited data from new distributions, leading to variations in adaptation performance. In contrast, our method considers contextual information about shifts to understand correlations among distributions and transfer knowledge, improving adaptability.

3.3 Preliminaries

3.3.1 Problem Definition

Distribution shifts in threat detection involve changes in the probability distribution of data over time or across domains (e.g., users, services). These shifts can affect the marginal feature distribution $\mathscr{P}(x)$, the conditional distribution $\mathscr{P}(y|x)$, or both, where x is the data and y is the category. Consider a threat detection model $f(\cdot;\theta)$ trained on a dataset $\mathbf{D} = \{x_i\}_{i=1}^{N}$ drawn from a distribution $\mathscr{P}$. The dataset $\mathbf{D}$ is unlabeled and is dominated by samples from the benign class. The model parameters θ are optimized by minimizing a loss function $mathcalL$. Common choices for $mathcalL$ include reconstruction loss in autoencoders or contrastive loss in self-supervised learning. The objectives: $\theta^* = \text{argmin}_\theta \, \mathbb{E}_{x\sim\mathscr{P}} \mathscr{L}(f(x;\theta))$. Our goal is to develop an adaptation method that updates model parameters to θ' using a few examples from the new distribution $\mathscr{P}'$. The observed dataset $\mathbf{D}'$ from distribution $\mathscr{P}'$ is unlabeled, and its size is small $|\mathbf{D}'| = k \ll |\mathbf{D}|$.

3.3.2 Meta-Learning

Meta-learning trains models that can quickly adapt to new tasks using only a few examples. A task $\mathscr{T}_i$ is defined as an independent learning problem with a dataset following a specific distribution $\mathscr{P}_i$ and a learning objective which is to find the optimal parameters θ_i^* that minimize the expected loss $\theta_i^* = \text{argmin}_\theta \, \mathbb{E}_{x\sim\mathscr{P}_i} \mathscr{L}(f(x;\theta))$. The dataset $\mathbf{D}_i$ for task $\mathscr{T}_i$ is divided into a support set $\mathbf{D}^i_{\text{support}}$ and a query set $\mathbf{D}^i_{\text{query}}$. $\mathbf{D}^i_{\text{support}}$ is used to fine-tune the model to learn task-specific parameters for $\mathscr{T}_i$, while $\mathbf{D}^i_{\text{query}}$ evaluates how well the model generalizes the learned task-specific knowledge.

Model-Agnostic Meta-Learning (MAML) [43] is one of the most representative algorithms, which optimizes the initial model parameters θ so that after fine-tuning, the model performs well across various tasks, minimizing the average loss. For each task $\mathscr{T}_i$, the parameters are fine-tuned using the support set $\mathbf{D}^i_{\text{support}}$: $\theta_i^* = \text{argmin}_\theta \sum_{x\in\mathbf{D}^i_{\text{support}}} \mathscr{L}(f(x;\theta))$. The weight

initialization θ is optimized using the query sets:

$$\theta^* = \operatorname*{argmin}_{\theta} \sum_{\mathscr{T}_i} \sum_{x \in \mathbf{D}^i_{\text{query}}} \mathscr{L}(f(x; \theta_i^*)).$$

During inference, the model is fine-tuned on a few samples from the new distribution and then applied to all testing samples.

3.3.3 Hypernetwork

A hypernetwork [56] is a neural network that predicts the weights of another neural network (i.e., target network). By training a single hypernetwork to predict weights across multiple tasks rather than optimizing each one independently, hypernetworks offer a parameter-efficient solution for model adaptation. It has shown effective in improving learning efficiency through parameter sharing [3, 17, 134, 229]. Let h represent the hypernetwork with parameters ϕ, and let f denote the target network. Given a representation $\mathbf{V}_i$ for describing task $\mathscr{T}_i$, the hypernetwork generates the model weights $\theta = h(\mathbf{V}_i; \phi)$, which are loaded into f for the downstream task. Given multiple tasks $\mathscr{T}_i$ and the corresponding task representations $\mathbf{V}_i$, the learning objective is to optimize the hypernetwork's parameters ϕ to minimize the loss $\mathscr{L}$ across these tasks:

$$\phi^* = \operatorname*{argmin}_{\phi} \sum_{\mathscr{T}_i} \sum_{x \in \mathbf{D}^i} \mathscr{L}[f(x; h(\mathbf{V}_i; \phi))].$$

3.4 Methodology

We approach the problem from both meta-training and fine-tuning perspectives. Through meta-training, the model establishes a strong, generalizable foundation that can be applied to most scenarios. Then, through fine-tuning, the model weights are slightly adjusted for specific shifts. We propose a method that decomposes model weights into two components and alternately optimizes them to address both perspectives. Algorithm 2 provides the pseudo-code.

Algorithm 2: Training Procedure of REACT

Input: Task distribution $\mathscr{P}(\mathscr{T})$, target network f, hypernetwork h
Output: Meta weights θ_{meta}, hypernetwork weights ϕ

Initialize model weights θ_{meta} and ϕ;
while *not converged* **do**
    /* Update meta weights */
    Sample a set of tasks $\{T_i\}_{i=1}^{M} \sim \mathscr{P}(\mathscr{T})$;
    for *each task* T_i **do**
        Form support set D_i^{support} and query set D_i^{query} and extract contextual information c_i;
        Generate adaptive weights: $\theta_i^{\text{adapt}} = h(D_i^{\text{support}}, c_i; \phi)$;
        Fine-tune θ_i^{adapt} on D_i^{support} following Eq. 3.1;
    Update θ_{meta} following Eq. 3.2;
    /* Update hypernetwork */
    Sample a set of tasks $\{T_j\}_{j=1}^{M} \sim \mathscr{P}(\mathscr{T})$;
    for *each task* T_j **do**
        Form support set D_j^{support} and query set D_j^{query} and extract contextual information c_j;
        Update ϕ following Eq. 3.2;

3.4.1 Weight Decomposition

Given a neural network, we decompose its weights into two complementary components: meta weights θ_{meta} and adaptive weights θ_{adapt}. Meta weights capture global patterns that are common across different distributions, representing the core knowledge acquired during meta-learning. Adaptive weights, on the other hand, serve as a small "residual" component that allows the model to be fine-tuned to the unique characteristics of specific data distributions, while still leveraging the global patterns encoded in the meta weights. The full model weights are then formed by adding the two components together: $\theta = \theta_{\text{meta}} + \theta_{\text{adapt}}$. By applying a small residual update to the pretrained meta weights, the model can adapt to new distributions without overwriting the essential pretrained knowledge.

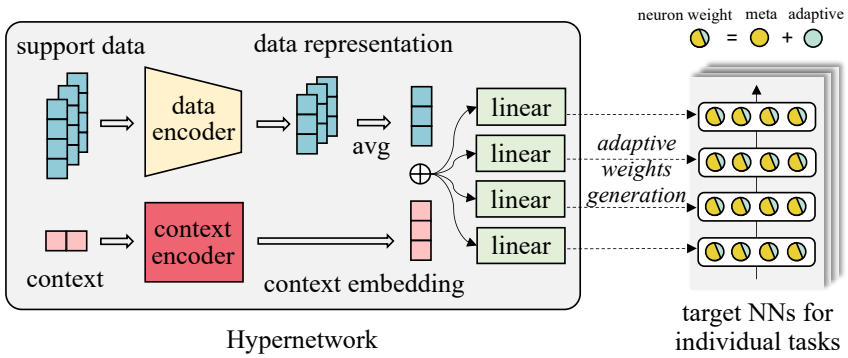


Figure 3.2. Architecture of the proposed hypernetwork in REACT.

3.4.2 Residual-Adaptive Weight Generation with Hypernetwork

We incorporate a hypernetwork to generate adaptive weights based on data characteristics and contextual information. The architecture of the hypernetwork is presented in Figure 3.2.

Data Encoding. Our hypernetwork includes a data encoder that processes data from the support set to produce feature representations. These representations are averaged and passed through a series of linear layers, with each layer producing the weights for a corresponding layer in the target network.

Context Encoding. To enhance the hypernetwork's ability to handle varying distributions, we integrate the contextual information c_i about distribution $\mathscr{P}_i$ as an additional input. This context offers semantic insights into the shifts and helps capture similarities across distributions. We incorporate a context encoder in the hypernetwork to transform contexts into embeddings, which are then added to the data representations for weight generation. The choice of context depends on the type of shift. For example, we use time information for temporal shifts, with positional encoding [207] generating embeddings. Details about context modeling for different tasks can be found in Section 3.6.1.

3.4.3 Alternating Optimization

We design an alternating optimization scheme to iteratively update the meta weights and the hypernetwork. This approach balances the learning dynamics and prevents mutual

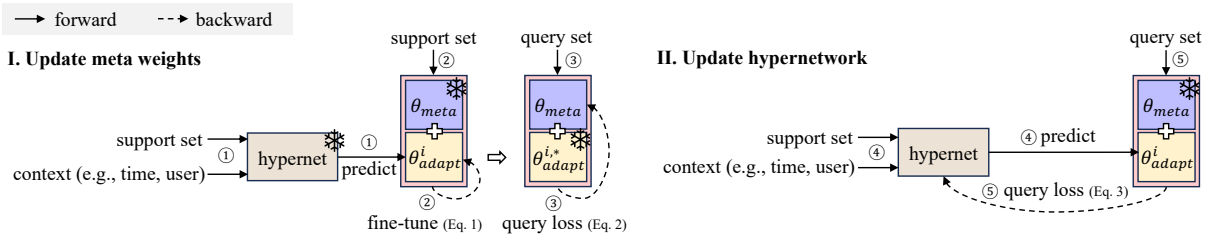


Figure 3.3. Alternating optimization in REACT. In each training iteration, we sample a set of tasks to update the meta weights, then sample another set to train the hypernetwork.

interference between the two components. Figure 3.3 illustrates the process.

Task Sampling for Meta Learning. To let the model learn how to adapt to new distributions, we first create a diverse set of tasks that reflect the expected variations in the application. We sample tasks from the training set by simulating the underlying data shifts. For instance, if the goal of adaptation is to address distribution shifts over time, the data can be grouped according to temporal factors such as day or month, with each time period forming a separate task. If the focus is on handling shifts across different users, the data can be grouped by users, with each user forming a task.

Update of Meta Component. Let θ^i_{adapt} denote the adaptive weights of task $\mathscr{T}_i$. In each iteration, we begin by updating the meta weights. We sample a set of tasks $\{\mathscr{T}_i\}_{i=1}^M$ and contextual information $\{c_i\}_{i=1}^M$. The hypernetwork $h(\cdot;\theta)$ is fixed and used to generate adaptive weights, $\theta^i_{\text{adapt}} = h(\mathbf{D}^i_{\text{support}}, c_i; \phi)$. The generated adaptive weights are then fine-tuned to derive the optimal model weight $\theta^{i,*}_{\text{adapt}}$ for task $\mathscr{T}_i$ by minimizing the empirical loss over the support set D^i_{support}:

$$\theta^{i,*}_{\text{adapt}} = \underset{\theta_{\text{adapt}}}{\text{argmin}} \sum_{x \in \mathbf{D}^i_{\text{support}}} \mathscr{L}(f(x; \theta_{\text{meta}}, \theta_{\text{adapt}})). \tag{3.1}$$

We then fix these fine-tuned adaptive weights and update the meta model by minimizing the loss on the query set $\mathbf{D}_{\text{query}}$. Let θ_{meta} be the learning rate for updating meta weights. The

update of meta weight after one gradient step is as follows:

$$\theta_{\text{meta}} \leftarrow \theta_{\text{meta}} - \eta_{\text{meta}} \nabla_{\theta_{\text{meta}}} \sum_{\mathscr{T}_i} \sum_{x \in \mathbf{D}^i_{\text{query}}} \mathscr{L}[f(x; \theta_{\text{meta}}, \theta^{i,*}_{\text{adapt}})]. \tag{3.2}$$

Update of Hypernetwork. Next, we sample another set of tasks $\{\mathscr{T}_j\}_{j=1}^{M}$, fix the meta weights learned in the previous step, and update the hypernetwork using the query sets. Let η_h be the learning rate for updating the hypernetwork. The weight update of hypernetwork after one gradient step is as follows:

$$\phi \leftarrow \phi - \eta_h \nabla_{\phi} \sum_{\mathscr{T}_j} \sum_{x \in \mathbf{D}^j_{\text{query}}} \mathscr{L}[f(x; \theta_{\text{meta}}, h(\mathbf{D}^j_{\text{support}}, c_j; \phi))]. \tag{3.3}$$

Regularization. We apply L2 regularization to the adaptive weights generated by the hypernetwork, encouraging them to act as residuals to the globally shared meta weights. The query loss for optimizing the hypernetwork, denoted as $\mathscr{L}_{\text{query}}$, is combined with the regularization as $\mathscr{L} = \mathscr{L}_{\text{query}} + \lambda \|\theta\text{adapt}^i\|_2^2$, where λ is the hyperparameter to control the regularization strength.

3.4.4 Adapting to New Distributions

When doing inference on a new distribution $\mathscr{P}_j$, the meta weights and the hypernetwork are fixed. This ensures the pre-trained knowledge is not "forgotten" during fine-tuning [27, 107, 135, 188], preserving generalizability of the model. A small number of support data D^j_{support} from $\mathscr{P}_j$ along with its contextual information c_j are fed into the hypernetwork to predict the adaptive weights $\theta^j_{\text{adapt}} = h(\mathbf{D}^j_{\text{support}}, c_j; \phi)$. With this initialization, the adaptive weights are then fine-tuned on $\mathbf{D}^j_{\text{support}}$ following Equation 3.2. Finally, the two parts of the weights are merged by summing them as if there is only one target network. The merged weights are used for inference on data from the new distribution $\mathscr{P}_j$.

3.5 Analysis

In this section, we provide convergence analysis of REACT on linear models. Let $\mathbf{X}^i$ be the matrix whose rows are the samples from the dataset of task $i \in \{1,...,M\}$, i.e., $\mathbf{D}_i$. The data matrix $\mathbf{X}^i$ can be split into support set $\mathbf{X}^i_s$ and query set $\mathbf{X}^i_q$. We assume that the relevant datasets are sampled at the beginning of the algorithm. Given linear model[1]

$$h(\mathbf{X};\phi) = \mathbf{X}\phi, \quad f(\mathbf{X};\theta_{\text{meta}},\theta_{\text{adapt}}) = \mathbf{X}(\theta_{\text{meta}} + \theta_{\text{adapt}}). \tag{3.4}$$

The following theorem provides convergence guarantees for REACT.

Theorem 3.5.1. *Consider* REACT *on the linear model in (3.4) with Eq. (3.1) being solved exactly. Let $\mathbf{X}^s_i$ and $\mathbf{X}^q_i$ satisfy $(\mathbf{X}^s_i)^\top \mathbf{X}^s_i = (\mathbf{X}^q_i)^\top \mathbf{X}^q_i = \sigma_i I$ for each task $i \in \{1,\dots,M\}$, where σ_i are the variances and I is the identity matrix. Learning rates are chosen as $\eta_{meta} < 1/\sum_{i=1}^M \sigma_i\lambda/(\sigma_i+\lambda)$ and $\eta_h < 1/\max\left(\sum_{j=1}^{n_h}\sigma_j(\sigma_j+\lambda), \|\mathbf{X}_s\|\right)$ where $\mathbf{X}_s = \sum_{j=1}^M \sigma_j(\mathbf{X}^j_s)^\top$. Then, for any $\varepsilon > 0$, there exists*

$$K = \mathscr{O}\left(\log_{\frac{1}{\rho_{meta}}}\left(\frac{1}{\varepsilon}\right) + \log_{\frac{1}{\rho_h}}\left(\frac{1}{\varepsilon}\right)\right),$$

for $\rho_{meta} = 1 - \eta_{meta}\sum_{i=1}^M \sigma_i\lambda/(\sigma_i+\lambda)$ and $\rho_h = 1 - \eta_h\sum_{j=1}^M \sigma_j(\sigma_j+\lambda)$ such that the K-iteration of Algorithm 2 satisfies

$$\|\theta^K - \theta^*\| \le \varepsilon, \quad \textit{and} \quad \|\phi^K - \phi^*\| \le \varepsilon,$$

where θ^* and ϕ^* are stationary points of the algorithm.

The proof is provided in Appendix A.2. Our results suggest that θ_{meta} and ϕ converge to stationary points at a linear rate which can be characterized based on the eigenvalues of the sample matrices in each task and hyperparameters considered in REACT.

[1]We note that we abuse the notation and set $h(\mathbf{X},c;\phi) = h(\mathbf{X};\phi)$,i.e., the context information is part of the input data.

Table 3.1. Experiment configurations and dataset statistics after preprocessing.

Dataset	# Train/Test	# Train/Test tasks	k	Shift by	Model
AnoShift	1.3M / 1.8M	50 / 110	100	Time	AutoEncoder
NSL-KDD	28K / 6K	8 / 6	10	Service	GOAD
Malware	15K / 5K	36 / 12	10	Time	DeepSVDD

3.6 Experiments

3.6.1 Experiment Setup

Datasets and Backbone Models. Our evaluation focuses on two key applications in cybersecurity, network intrusion detection and malware detection, and targets both temporal and domain shifts. REACT is compatible with various neural network architectures.

- **AnoShift** [37] is a benchmark for network intrusion detection under distribution shifts. It collects traffic logs from a university network between 2006 and 2015. Data shifts occur over time due to reasons such as user behavior changes and software updates. Each sample has 15 features (9 numerical and 6 categorical) and a label of whether it is an attack. We use the train-test split provided by the dataset, including training subsets from 2006 to 2010 and test subsets from 2006 to 2015. Each month is regarded as a separate task.
- **Malware** [68] contains executables collected between 2010 and 2014 from VirusShare2, an online malware analysis platform. Data shifts happen over time. Each executable has 482 counting features and a risk score (ranging from 0 to 1) indicating the probability of it being malware. These risk scores are converted to binary labels using thresholding, with executables labeled as malicious if the score is greater than 0.6 and benign if the score is less than 0.4 [68]. Following previous work [104], the dataset is split into training data from 2011 to 2013, validation data from 2010, and testing data from 2014. Each month is treated as a separate task.
- **NSL-KDD** [185] is another dataset for evaluating network intrusion detection. Each sample has 40 attributes describing the network traffic, with 6 categorical and 34 numerical features.

We simulate domain shifts by grouping data according to services (e.g., HTTP, Telnet) and randomly assigning half of the services as training tasks and the other half as test tasks. We use the official train-test split provided by the dataset and remove services not selected for the respective splits. Besides, services with fewer than 20 benign samples are excluded to ensure sufficient unseen data for testing.

We sample the datasets to form a 10% ratio of threat samples for both training and testing. In Section 3.6.4, we vary this ratio to test the robustness of REACT to the contamination of training data. For the NSL-KDD dataset, since GOAD is a semi-supervised method that trains only on benign data, we remove attack samples from the training set.

Backbone Models and Implementation. To assess the adaptation performance across different models, we employ three representative architectures as follows:

- **AutoEncoder (AE)** [2]: is an unsupervised model trained to reconstruct the input through an encoder-decoder structure. The key idea is that, threat samples, appearing less frequently, tend to have larger reconstruction losses, making them distinguishable by observing the loss. We implement the AutoEncoder with four linear layers followed by ReLU activation. These layers project the data into [64, 32, 64]-dimensional features and finally map the features to the original data dimension. Cross-entropy loss is applied to categorical features, and mean-square error is used for numerical features. We use it as the backbone model for AnoShift.
- **DeepSVDD (DSVDD)** [163]: is an unsupervised model which encodes data into feature representations and measures their distances from a learnable center. The encoder is a multi-layer perception (MLP) consisting of two linear layers with ReLU activation, mapping data to representations of dimension [64, 32]. Similar to the AutoEncoder, threat samples tend to have larger distances from the center. The smooth L1 loss [162] is used to measure the distances for its robustness against outliers. We use it as the backbone model for Malware.
- **GOAD** [15]: is a semi-supervised model that applies multiple transformations to the input data and uses a convolutional neural network (CNN) [96] to extract feature representations. We implement a 5-layer CNN with kernel size of 1. The loss function has two components: a

center triplet loss, which measures the distance between the learned representations and their mean, and a cross-entropy loss for predicting which transformation was applied to the data. It is used as the backbone model for NSL-KDD.

Table 3.1 summarizes the statistics and configurations of the datasets and backbone models in the experiments.

Baselines. We compare REACT with unsupervised methods from the anomaly detection benchmark [59], including linear and statistical models: ECOD [118], COPOD [117], OCSVM [168]; distance- and proximity-based methods: LOF [18], KNN [159]; ensemble methods: LODA [154], IForest [124]; and neural networks: AE [2], DSVDD [163], LUNAR [49]. These methods assume static environments and do not account for distribution shifts. In addition, we compare REACT with training from scratch, fine-tuning strategies, and state-of-the-art model adaptation methods. Brief descriptions are as follows:

- **Static model:** The model is trained on the training data and directly tested on each test task, i.e., the pretrained model.
- **Train-from-scratch:** For each task, a model is trained from scratch using k samples and is used for evaluation.
- **Fine-tuning:** For each task, the model is fine-tuned using k samples from the task based on the pretrained model.
- Continual learning: Starting from pretrained model, we sequentially fine-tune the model using k samples from each task.
- **Experience Replay (ER)** [25] is a method to mitigate catastrophic forgetting in continual learning. We maintain a memory buffer to store historical data. In each fine-tuning iteration, we sample a batch from this buffer and compute its loss. This loss is then weighted and combined with the loss from the new batch.
- **ACR** [104] is a zero-shot adaptation which adopts meta-learning to train the model and update the batch normalization (BN) layers with the batch statistics during inference. We add a BN layer after each linear or convolutional layer in the model.

- **OC-MAML** [44] is a few-shot one-class classification method. It extends MAML by modifying the episodic data sampling strategy. It forms one-class support sets to optimize the meta model.

Training and Adaptation Configurations. The size of support data k is set based on the data quantity, with $k = 100$ for AnoShift and $k = 10$ for Malware and NSL-KDD. The size of query data varies proportionally, with 1000 for AnoShift and 100 for Malware and NSL-KDD. In each meta-training iteration, we sample $M = 5$ tasks for Malware and NSL-KDD, and $M = 1$ for AnoShift. The number of fine-tuning epochs E is determined by the convergence rate of the learning task, with $E = 10$ epochs for AnoShift and Malware, and $E = 1$ for NSL-KDD due to its faster convergence. Regularization parameter $\lambda = 10$.

Choices of Contexts. For AnoShift and Malware whose shifts occur over time, we use time index as the context, which is modeled by positional encoding [207] to generate contextual embedding for each task. For NSL-KDD dataset whose shifts occur across services, we first feed these services names to GPT-4 [1] with the prompt "please briefly describe each of these web services, including the normal and anomalous patterns". The use of a large language model (LLM) is to reduce the reliance on domain experts. The LLM choice is generic, and other advanced models may be used. Then, we use Sentence Transformer4 to generate embeddings for the descriptions. Evaluation Metrics. For each test task, we adapt the model and evaluate its performance using AUROC and AUPR scores. All experiments are repeated for five times with the same set of random seeds, and the results are averaged across all test tasks and runs.

3.6.2 Main Results and Analysis

The results are presented in Table 3.2, where the left sub-table shows the performance of static methods and the right one focuses on fine-tuning and adaptation methods across three backbone models. The static methods (left table) generally show lower performance than models with adaptation (right table), highlighting the negative impact of distribution shifts on model performance. The right table also includes the performance of train-from-scratch using all data

Table 3.2. Main experiment results on threat detection (averaged over 5 runs). The left sub-table reports the performance of static methods, while the right focuses on fine-tuning and adaptation across three backbone models. REACT consistently outperforms all other methods. $|D_i|$ denotes the number of samples observed from each test task for gradient updates.

Method	AnoShift		Malware		NSL-KDD	
	AUROC	AUPR	AUROC	AUPR	AUROC	AUPR
KNN [159]	0.6714	0.4062	0.2732	0.1040	0.5323	0.2956
LOF [18]	0.6107	0.2873	0.2781	0.1101	0.4150	0.1827
OCSVM [168]	0.6903	0.3157	0.3880	0.1190	0.6649	0.3492
iForest [124]	0.6658	0.2830	0.2660	0.0722	0.7809	0.4798
LODA [154]	0.5723	0.2111	0.5190	0.1368	0.5207	0.2532
AE [2]	0.7110	0.3204	0.3789	0.1156	0.6057	0.2678
DSVDD [163]	0.7716	0.3895	0.5165	0.1644	0.6006	0.2848
COPOD [117]	0.7664	0.3831	0.4450	0.1102	0.7849	0.4471
ECOD [118]	0.7461	0.3727	0.5403	0.1390	0.8100	0.4706
LUNAR [49]	0.4449	0.2450	0.2719	0.0880	0.5243	0.2350

Method	$\|\mathbf{D}_i'\|$	AnoShift		Malware		NSL-KDD	
		AUROC	AUPR	AUROC	AUPR	AUROC	AUPR
train-from-scratch	all	0.7681	0.3689	0.6010	0.1822	0.9308	0.7107
static model	-	0.7110	0.3204	0.5165	0.1644	0.7420	0.5110
train-from-scratch	k	0.7398	0.3398	0.3659	0.1337	0.7382	0.4219
fine-tuning	k	0.7039	0.3333	0.5678	0.1797	0.8175	0.5098
continual learning	k	0.6087	0.3063	0.5879	0.1873	0.8285	0.5188
ER [25]	k	0.6144	0.2996	0.6022	0.1932	0.8356	0.5114
ACR [104]	-	0.7634	0.3753	0.5798	0.1794	0.7513	0.4658
OC-MAML [44]	k	0.7770	0.3811	0.6779	0.2334	0.8547	0.5504
REACT (ours)	k	**0.8226**	**0.4376**	**0.7252**	**0.2750**	**0.8673**	**0.5559**

from each individual test task (in grey). When sufficient data is available from the new tasks, training a model from scratch yields better performance than using a pretrained model without adaptation. When comparing the models trained from scratch, fine-tuning, and continual learning, it is shown that the pretrained model can be a poor initialization for shifted distributions, e.g., AnoShift. We also observe that experience replay slightly improves performance compared to continual learning without any strategy to prevent catastrophic forgetting. However, this improvement is limited.

REACT consistently outperforms the baselines and even surpasses the model trained from scratch using all data from individual tasks on two datasets. This is because REACT adapts from a model meta-trained on a larger and more diverse training set than each individual task, providing a stronger foundation for adaptation. Furthermore, the training data in AnoShift and Malware contains noise (10% threat ratio). Training a model on all data from an individual task increases the likelihood of exposing the model to many threat samples within that distribution, which, due to the specific training objectives of AutoEncoder and DSVDD, may cause the models to mistakenly learn malicious patterns as benign ones. In contrast, REACT is less prone to such overfitting as it utilizes the meta model and only updates the adaptive weights on a small set of new data. We note that on NSL-KDD, GOAD achieves high scores when trained from scratch using all data since it is trained on benign data only, but such training is impractical in the real

Table 3.3. Ablation studies. The results demonstrate that every component in our method contributes to the overall performance improvement.

Method	AnoShift		Malware		NSL-KDD	
	AUROC	AUPR	AUROC	AUPR	AUROC	AUPR
static model	0.7110	0.3204	0.5165	0.1644	0.7420	0.5110
fine-tuning	0.7039	0.3333	0.5678	0.1797	0.8175	0.5098
OC-MAML	0.7770	0.3811	0.6779	0.2334	0.8547	0.5504
REACT (ours)	**0.8226**	**0.4376**	**0.7252**	**0.2750**	**0.8673**	**0.5559**
w/o fine-tuning	0.7873	0.3977	0.7159	0.2696	0.7282	0.4838
w/o context	0.7865	0.3838	0.6772	0.2325	0.8503	0.5323
w/ random context	0.7922	0.3888	0.6892	0.2426	0.8597	0.5522
w/o regularization	0.6541	0.3379	0.6326	0.2001	0.8045	0.4705

world. When compared to other baselines using the same k samples from new tasks, REACT achieves the highest scores. Among state-of-the-art methods, ACR, which performs test-time adaptation, shows relatively lower performance as it does not apply gradient updates during inference, limiting its adaptation ability. OC-MAML achieves the second-best performance, demonstrating the strength of meta-learning. However, REACT outperforms OC-MAML by weight decomposition and incorporating a hypernetwork for contextual tuning. These designs help maintain generalizability and enhance adaptability beyond meta-learning alone.

3.6.3 Ablation Studies

We crafted four ablated versions of REACT by systematically removing each key component: (1) We disable fine-tuning during inference and use the merged weights from meta weights and the hypernetwork's prediction to do the inference directly, denoted as **w/o fine-tuning**. (2) We remove the use of context and only provide the support data for hypernetwork, denoted as **w/o context**. (3) We replace the context with fixed random vectors of the same shape as context embeddings sampled from a normal distribution, denoted as **w/ random context**. (4) We remove the regularization term on the hypernetwork's prediction, denoted as **w/o regularization**.

The results in Table 3.3 show that every component in REACT contributes to performance improvement. Fine-tuning and regularization have notable impacts. REACT w/o fine-tuning shows competitive performance on AnoShift and Malware compared to the baselines, indicating its potential for zero-shot adaptation. However, with just a few gradient updates, performance can be largely improved. Besides, adding regularization ensures the adaptive weights predicted by the hypernetwork do not overpower the full model, maintaining model generalizability. We present the results with different values of the regularization parameter in Section 3.6.4. REACT w/o context learns distribution patterns solely from the support data, which is less effective than incorporating contexts since the support data is limited and might not provide sufficient insights. REACT w/ random context introduces randomness into the encoded representations. This exposes the hypernetwork to variations during training and reduces reliance on specific patterns, enhancing robustness to noise. Therefore, it performs slightly better than w/o context. However, these random contexts do not provide task-specific knowledge to capture meaningful patterns. With additional information about tasks, REACT can model similarity among distributions more effectively. We anticipate that selecting indicative contexts for downstream applications and using advanced techniques like graph-based prompting [181] could further enhance context encoding. These explorations are left for future work.

3.6.4 Sensitivity Analyses

Number of Support Samples. We vary the number of support samples k for each task from 5 to 100 and compare REACT with the fine-tuning baseline. Figure 3.4 shows that REACT consistently outperforms the fine-tuning baseline across all datasets. This demonstrates REACT's robustness in data-scarce scenarios and highlights its ability to efficiently leverage available data for fast adaptation.

Number of Fine-Tuning Epochs. We vary the number of fine-tuning epochs for each new task from 1 to 10 and compare the performance of REACT with the fine-tuning baseline. Figure 3.5 shows that REACT outperforms the baseline in all settings. On NSL-KDD, the

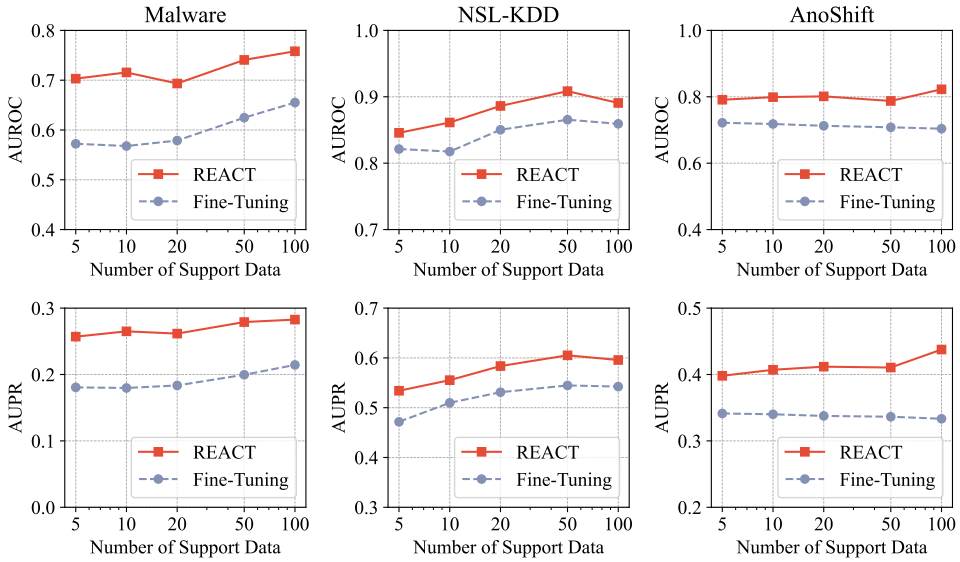


Figure 3.4. Sensitivity analysis of the number of support samples k.

improvement of REACT over the baseline becomes less significant with more fine-tuning epochs, as its tasks are simpler and the model is able to adapt to them with fewer epochs.

Contamination on Training Data. We evaluate the robustness of our system against contamination in the training data when applying to AutoEcoder and DeepSVDD models on AnoShift and Malware respectively—both unsupervised methods. We note that GOAD is a semi-supervised method trained solely on benign data (as applied to the NSL-KDD dataset) thus the evaluation is trivial to it. We fix the number of benign samples while varying the ratio of threat samples from 1% to 20%. Table 4 shows the AUROC scores. REACT consistently achieves higher AUROC scores across different contamination rates than the fine-tuning baseline, showing that it is robust to noise in training data.

Effect of Regularization. We experiment with different values of the regularization parameter λ as presented in Table 3.5. Adjusting λ controls the trade-off between adaptability and generalization. A larger λ reduces the norm of adaptive weights and emphasizes generalization, while a smaller λ encourages adaptability. Selecting an appropriate λ is essential for achieving optimal performance.

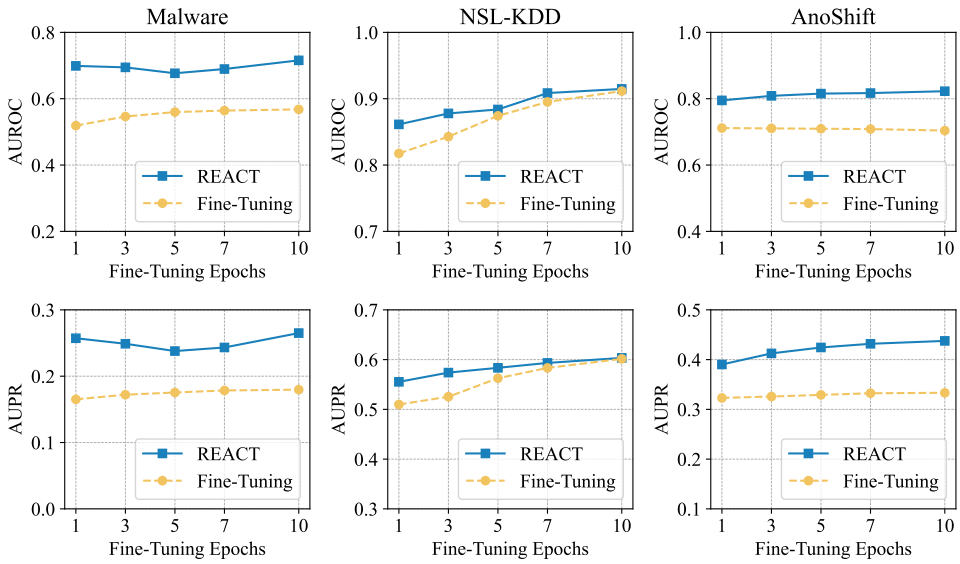


Figure 3.5. Sensitivity analysis of the number of fine-tuning epochs (E).

3.6.5 Computational Efficiency in Adaptation

The hypernetwork is fixed after training. During adaptation, it conducts a single forward pass for a new task which incurs minimal overhead. As adaptation typically occurs less frequently (e.g., once daily) than inference, this overhead is negligible. Table 6 shows the run time of hypernet forward pass (t_1) and gradient descent fine-tuning (t_2) during the adaptation of a new task. Experiments are performed with a NVIDIA Tesla T4 GPU.

3.6.6 Parameter-Efficient Fine-Tuning

Our method supports parameter-efficient fine-tuning, which is especially useful when working with large models. By incorporating adaptive weights into only a subset of the model's parameters and having the hypernetwork predict this subset of weights, we can reduce the number of parameters to be fine-tuned. We conduct experiments using an AutoEncoder on the AnoShift dataset to showcase REACT's ability in parameter-efficient fine-tuning. Specifically, we predicted adaptive weights for either the two symmetric linear layers closest to the input (denoted as REACT-Inner) or those closest to the latent representations (denoted as REACT-Outer). Full fine-tuning of REACT is denoted as REACT-Full. The results are shown in Figure

Table 3.4. Sensitivity analysis: AUROC scores under different contamination levels.

Method	Malware				AnoShift			
	1%	5%	10%	20%	1%	5%	10%	20%
static model	0.506	0.513	0.517	0.567	0.764	0.753	0.711	0.634
train-from-scratch	0.366	0.368	0.366	0.377	0.818	0.791	0.740	0.740
fine-tuning	0.550	0.545	0.568	0.580	0.812	0.765	0.704	0.605
continual learning	0.562	0.559	0.588	0.590	0.683	0.572	0.609	0.453
ER	0.582	0.585	0.602	0.602	0.734	0.669	0.614	0.577
ACR	0.544	0.570	0.580	0.570	0.785	0.774	0.763	0.773
OC-MAML	0.683	0.688	0.678	0.687	0.827	0.803	0.777	0.755
REACT (ours)	**0.725**	**0.738**	**0.725**	**0.719**	**0.832**	**0.813**	**0.823**	**0.775**

Table 3.5. Results with different regularization parameter λ.

λ	AnoShift		Malware		NSL-KDD	
	AUROC	AUPR	AUROC	AUPR	AUROC	AUPR
0	0.6541	0.3379	0.6326	0.2001	0.8045	0.4705
1	0.7276	0.3860	0.6878	0.2531	0.8260	0.5331
10	0.8226	0.4376	0.7252	0.2750	0.8673	0.5559
100	0.7889	0.3947	0.7048	0.2561	0.8192	0.5134

6. Both methods achieve better performance compared to the baselines, although they slightly underperformed compared to REACT-Full which fine-tunes all layers. Notably, REACT-Inner achieved a 5.75% higher AUC while updating 94.3% fewer parameters compared to conventional full fine-tuning, highlighting its efficiency.

3.6.7 Case Study

To understand how well REACT leverages contextual information, we analyze the weights generated by the trained hypernetworks. Specifically, we compare the adaptive biases of the last layer in the AutoEncoder for AnoShift across different months and calculate their cosine similarities. The results are presented in Figure 3.7a, with warmer colors indicating higher similarity. The high similarities around the diagonal indicate the weights generated for each month are similar to those of nearby months. This suggests that REACT effectively captures the

Table 3.6. Run time of REACT for adaptation.

Dataset	t_1 (ms per task)	t_2 (ms per task)	t_1/t_2
AnoShift	1.52	255.39	0.59%
NSL-KDD	1.58	59.42	2.66%
Malware	1.61	18.69	8.62%

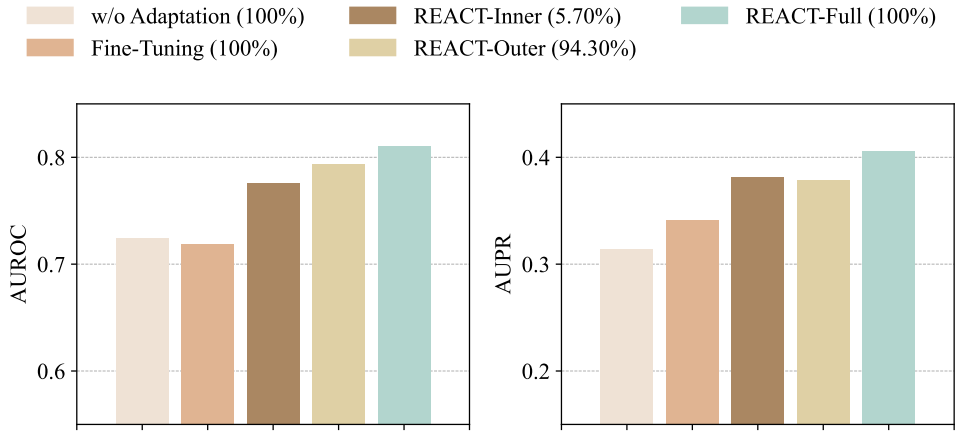
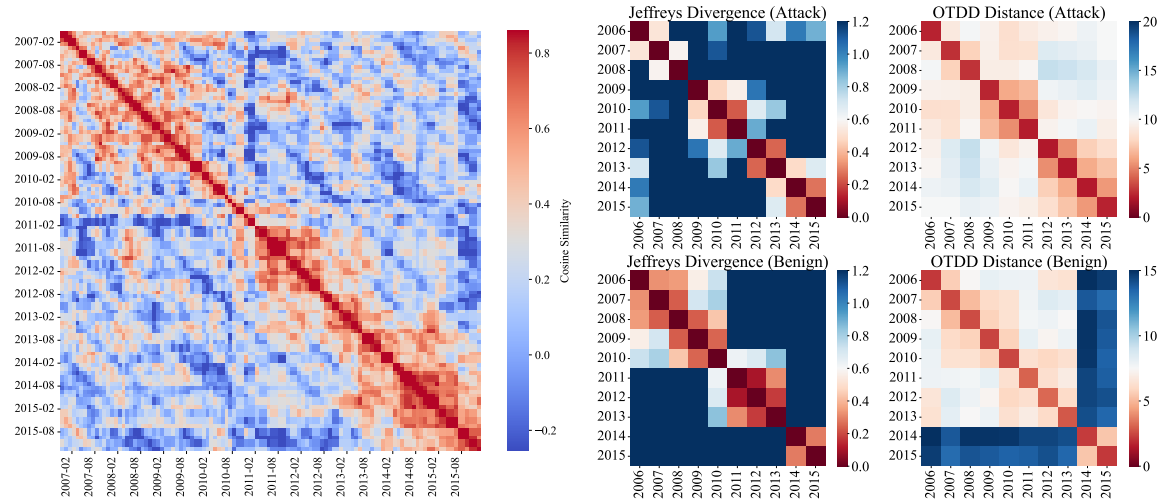


Figure 3.6. Parameter-efficient fine-tuning. Numbers in the legend indicate the percentage of fine-tuned parameters. Both REACT-Inner and REACT-Outer outperform the baselines.

temporal dynamics and smoothly adapts model weights over time. As a reference for how data shifts, we follow the analyses in [37] to calculate distances between data subsets of each year. Specifically, we measure the Jeffrey's Divergence [70] averaged over categorical features and the Optimal Transport Dataset Distance (OTDD) [5] across all features. As shown in Figure 3.7b, data from adjacent years exhibit smaller distances (in red). Besides, it presents block patterns where data from 2006–2010, 2011–2013, and 2014–2015 are more similar within their respective groups than with other years. This temporal shift corresponds with trends in weight similarity over time. The observations also hint at the potential for detecting shifts, a research question actively discussed in the literature [58, 91, 157]—by monitoring deviations in the hypernetwork's predictions compared to prior tasks, we may identify moments where shifts occur.



(a) Cosine similarity of adaptive weights generated by the hypernetwork.

(b) Distances between data subsets of each year.

Figure 3.7. Case study. The adaptive weights generated for each month are similar to those of nearby months, reflecting the data shift pattern.

3.7 Summary

Our work sheds light on how to approach the distribution shift problem—from both meta-learning and fine-tuning perspectives. We propose a novel method, REACT, that decomposes the weights of a neural network into the sum of meta and adaptive components, following a meta-learning paradigm to train the components. By integrating a hypernetwork to generate adaptive weights, REACT enables knowledge sharing and adjusts weights for new distributions with minimal fine-tuning effort. The approach is model-agnostic, generally applicable to arbitrary neural networks. It works effectively with unlabeled and imbalanced data, making it broadly applicable to various threat detection models and objectives. While focused on cybersecurity, the principles and methods developed in our research can be adapted to other fields facing similar distribution shift challenges, such as finance [47, 53, 195] and healthcare [72, 84, 169]. Our study provides insights for studies in the general machine learning community, fostering a more comprehensive understanding of adaptation and fine-tuning by showcasing applications in cybersecurity. One of the future directions is to incorporate a lightweight mechanism for updating

the meta model within our method. A potential solution could involve applying aggregation of the predicted adaptive weights into the meta model. This approach could enhance the ability to continuously adapt to evolving distributions, especially for scenarios with significant distribution shifts over a long period of time.

Chapter 3 incorporates material from the publication "REACT: Residual-Adaptive Contextual Tuning for Fast Model Adaptation in Threat Detection", by Jiayun Zhang, Junshen Xu, Bugra Can, Yi Fan, published in Proceedings of the ACM Web Conference 2025. The dissertation author was the primary investigator and the lead author of this paper.

Part II

Addressing System Heterogeneity

Chapter 4

Collaborative Training in Resource-Skewed Environments

The previous chapters addresses data heterogeneity during training and inference stages. In the second part, we extend our focus to system heterogeneity when coordinating training across distributed nodes. In this chapter, we identify a practical scenario where nodes in the federated network are skewed in data volume and computing resources. We leverage a graph hypernetwork to enable effective knowledge sharing.

Real-world deployment of federated learning requires orchestrating clients with widely varied compute resources, from *strong* enterprise-grade devices in data centers to *weak* mobile and Web-of-Things devices. Prior works have attempted to downscale large models for weak devices and aggregate shared parts among heterogeneous models. A typical architectural assumption is that there are equally many strong and weak devices. In reality, however, we often encounter *resource skew* where a few (1 or 2) strong devices hold substantial data resources, alongside many weak devices. This poses challenges—the unshared portion of the large model rarely receives updates or gains benefits from weak collaborators.

We aim to facilitate reciprocal benefits between strong and weak devices in resource-skewed environments. We propose RECIPFL, a novel method featuring a server-side graph hypernetwork. This hypernetwork is trained to produce parameters for personalized client models adapted to device capacity and unique data distribution. It effectively generalizes knowl-

edge about parameters across different model architectures by encoding computational graphs. Notably, RECIPFL is agnostic to model scaling strategies and supports collaboration among arbitrary neural networks. We establish the generalization bound of RECIPFL through theoretical analysis and conduct extensive experiments with various model architectures. Results show that RECIPFL improves accuracy by 4.5% and 7.4% for strong and weak devices respectively, incentivizing both devices to actively engage in federated learning.

4.1 Introduction

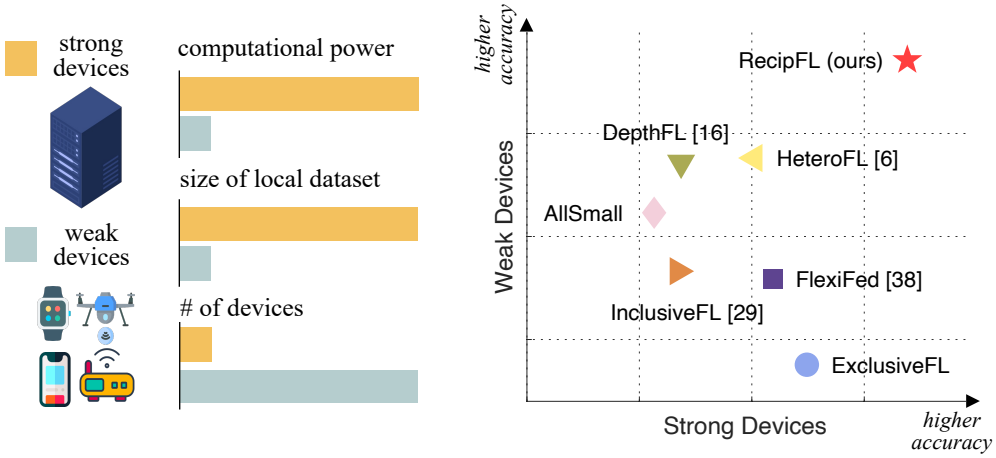
The real-world deployment of federated learning needs to deal with heterogeneous edge computing environments [93, 187, 74]. Typically, a few devices, often owned by large enterprises, can be powerful enough to afford large models, while the vast majority are 'weak' devices that can only host small models, such as mobile and Web-of-Things (WoT) devices owned by individuals. Universally deploying homogeneous small models as required by traditional methods [88, 89] not only wastes available compute resources but also compromises performance. Ideally, we need models scaled to fit varying device capacities and perform effective model aggregation.

To support collaboration among heterogeneous models, prior methods [102, 147, 33, 127, 82] downscale the large model for weak devices and perform aggregation on common components. These works typically assume there are equally many strong and weak devices [82, 200, 127]. However, in reality, we often see a skewed computing environment where a small number of strong devices operated by enterprises are accompanied by a large number of user-owned weak devices. For example, a smartwatch company wants to develop an activity recognition system. The company trains a large model using a vast dataset gathered from controlled environments, while its smartwatch users join via federated learning to train small models using personal data in the wild. Although small models are expected to benefit from the large model [62, 10], their contribution to the large model is dubious given their limited capability.

In light of this gap, we explore a new research question: *Can strong devices benefit from weak devices in resource-skewed environments?* We consider an extreme scenario where limited (1 or 2) strong devices and numerous weak devices engage in the learning, as depicted in Figure 4.1a. In this scenario, the learning system heavily leans on weak devices, leaving the unshared portion of the large model rarely being updated or deriving benefits from others. This presents a significant challenge in improving strong devices.

Existing approaches employ either width-scaling to prune channels or neurons in each layer of the large model [102, 147, 33], or depth-scaling for layer-wise pruning [127, 82]. They rely on weight-averaging aggregation [138, 111] to update shared layers. However, it can be destructive when layers or neurons in small models are ill-aligned with those in large models. For example, if the first block of ResNet [61] operates as an independent model for a complete vision recognition process, its layers function differently compared to their counterparts within an entire ResNet. When a few full ResNets are aggregated with many of their smaller versions (i.e., first blocks only), the first block may only extract shallow vision features, leading to performance decline. Even facilitated with knowledge distillation [82], improvements are not guaranteed if knowledge is transferred from numerous small models biased by non-IID data, as corroborated by our experiment results.

To effectively align and aggregate heterogeneous models, we propose RECIPFL, a novel federated learning method that empowers the server with a graph hypernetwork tasked with producing personalized model parameters for clients. Clients retain the flexibility to adapt the model to their capacities, through pruning or architectural changes. The server transforms client models into directed acyclic graphs to delineate their computation flow among layers. Figure 4.2 presents the overview of RECIPFL. Unlike traditional weight-averaging aggregation, which requires layers to have uniform operations, sizes, and computational flows, graph hypernetwork supports collaboration among arbitrary model architectures. This is achieved by encoding the computational graphs of client models with a graph neural network (GNN) [166, 115] and decoding parameters with multi-layer perceptrons (MLPs). It captures shared patterns among



(a) Resource Skew **(b)** Performance Summary

Figure 4.1. Illustration of problem setting and the performance of prior methods in resource-skewed environments.

model architectures, such as residual block and convolution patterns, and generalizes knowledge across them. The hypernetwork is trained using feedback (i.e., updated weights) from clients during federated learning. The computations of hypernetwork are executed by the server and therefore do not add extra communication or computation overhead to edge devices. We further augment weak devices by distilling knowledge from large models to smaller ones on strong devices.

We theoretically analyze the generalization bound of RECIPFL and empirically evaluate RECIPFL across four datasets for image classification and natural language inference. We simulate non-IID client distributions and evaluate personalized client models on their own test data as real-world WoT and mobile computing applications typically require. The results show RECIPFL outperforms state-of-the-art methods across different scaling strategies and various model architectures with significant margins. Notably, RECIPFL yields improvements for both strong and weak devices, demonstrating that even devices with limited computational resources can contribute meaningfully to the learning system, thereby providing incentives to both devices to participate in federated learning.

Our contributions are summarized as follows:

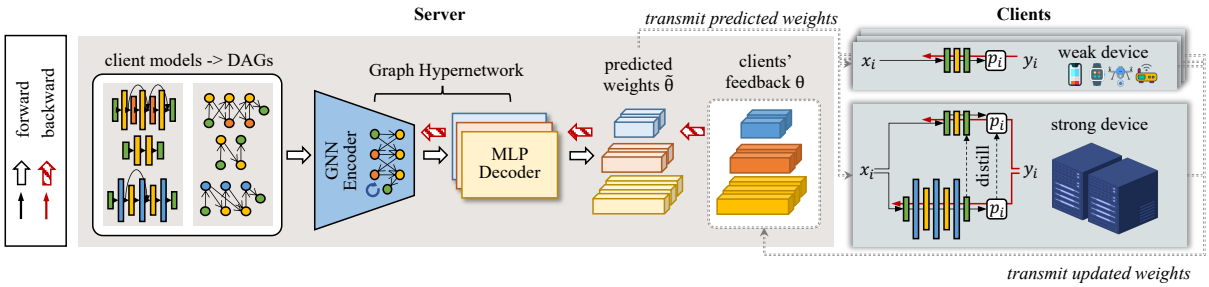


Figure 4.2. Overview of RECIPFL. The server transforms client models into directed acyclic graphs (DAGs) to represent the computation flow among operations and trains a graph hypernetwork to generate weights for customized client models.

- We address a new research question in federated learning: *Can strong devices benefit from weak devices in resource-skewed environments?* We show the existing approaches do not guarantee improvement for both types of devices.
- We propose a novel method RECIPFL to effectively generate weights for heterogeneous client models based on graph hypernetwork, compatible with arbitrary model scaling strategies.
- We establish the generalization bound of RECIPFL through theoretical analysis and validate its performance through extensive experiments. RECIPFL outperforms various state-of-the-art methods with significant margins and demonstrates that weak devices can also contribute effectively to the learning of strong devices.

4.2 Preliminaries

4.2.1 Problem Definition

We aim to build a federated learning system with M clients that allows the clients to have customized model architectures $\{\mathscr{G}_m | m \in [M]\}$ that fit their specific running capabilities. Within the M clients, there are a few (e.g., 1 or 2) strong devices that have enough running capacity to hold large models and the rest are weak devices having limited computing power. Denote the training set on client m as $\mathbf{D}_m = \left\{\left(x_i^{(m)}, y_i^{(m)}\right)\right\}_{i=1}^{N_m}$, where $x_i^{(m)}$ is the input data and $y_i^{(m)}$ is the label, and the data distribution of client m as $\mathscr{P}_m$. Denote ℓ as the loss function. The goal is to

learn a personalized model $f_m(\cdot;\theta_m)$ for every client m that works on its own data distribution:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{(x,y)\sim\mathscr{P}_m} \ell(f_m(x;\theta_m), y), \tag{4.1}$$

where $\boldsymbol{\theta}$ is the set of client model weights: $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_M\}$.

4.2.2 Resource-Skewed Computing Environments

Existing works often assume the strong and weak devices are equally distributed [127, 82, 200]. Among these, a tangential sensitivity evaluation [127] indicates that when strong devices are the minority, the large models deployed on them converge slowly and the performance is even worse than training them without the participation of weak devices. Yet, there is a lack of analysis or solutions for this resource skew issue.

To understand how existing methods perform in skewed computing environments, we summarize the results from Section 2.5 by averaging the accuracies across all datasets for strong and weak devices respectively. Our experiments use a majority of weak devices (e.g., 5, 20, 50, 500) and 1 or 2 strong devices, allocating 50% of data to strong devices and the rest to weak devices under Dirichlet distributions. We craft two naive baselines: AllSmall, which trains small models on all devices via federated learning, and ExclusiveFL, which trains large models on strong devices and small models on weak devices, with weight aggregation carried out separately within each group. In addition, we include comparisons with existing federated methods for heterogeneous models [82, 127, 200, 33]. We evaluate client models on their own test data sampled from clients' data distributions. The summary is presented in Figure 4.1b, from which we draw the following observations:

- *Small models are insufficient for strong devices:* By comparing AllSmall and ExclusiveFL on strong devices, we see that training small models with collaboration from weak devices yields lower accuracy compared to training large models independently.

- *Weak devices benefit from collaboration with strong devices:* By comparing ExclusiveFL and

other methods on weak devices, we observed that all other methods show higher accuracy than training small models independently.

- *Strong devices derive minimal benefits from weak devices with existing methods:* For strong devices, we see the accuracy of existing methods is generally lower than ExclusiveFL.

- *Existing methods could enhance the performance of one type of model but struggle to improve both.* For example, DepthFL achieves high accuracy on weak devices but does not perform well on strong devices. FlexiFed achieves higher accuracy on strong devices than DepthFL but shows less improvement on weak devices.

Recognizing these limitations, we aim to enable mutual benefits between both types of devices in resource-skewed environments.

4.3 Methodology

RECIPFL employs a graph hypernetwork at the server that learns from client feedback during federated learning. This hypernetwork encodes clients' computational graphs, enabling it to generalize knowledge across different model architectures and produce personalized client parameters. Algorithm 3 outlines the pseudo code.

4.3.1 Federated Training

In each training round, the server initiates the process by randomly selecting a subset of clients, denoted as $\mathbf{S}_t$, to conduct local updates. The server utilizes the graph hypernetwork to produce model weights $\{\tilde{\theta}_m | m \in \mathbf{S}_t\}$, sends the weights to selected clients and waits for their feedback. At the client side, the client performs local updates by training the client model f_m with its local dataset $\mathbf{D}_m$. The training objective at client m is to minimize the loss:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathscr{L}_m(\theta_m) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N_m} \sum_{i=1}^{N_m} \ell\left(f_m\left(x_i^{(m)}; \theta_m\right), y_i^{(m)}\right), \tag{4.2}$$

Algorithm 3: Pseudo-code of RECIPFL

Input : Communication rounds T, number of selected clients per round $|\mathbf{S}_t|$, local training epochs E, client descriptors $\{a_m | m \in [M]\}$ and model architectures $\{\mathscr{G}_m | m \in [M]\}$.

Output : A graph hypernetwork that generates personalized model weights for heterogeneous client models.

Server executes:
for $t = 1, \ldots, T$ **do**
  Select a subset $\mathbf{S}_t$ of clients at random;
  for $m \in \mathbf{S}_t$ **do**
    $\tilde{\theta}_m \leftarrow \text{GHN}(\mathscr{G}_m, a_m; \phi)$;
    $\theta_m \leftarrow$ **ClientUpdate**$(m, \tilde{\theta}_m)$;
    $\Delta\theta_m \leftarrow \theta_m - \tilde{\theta}_m$;
  Update GHN: $\phi \leftarrow \phi - \eta_s \sum_{m \in \mathbf{S}_t} (\nabla_\phi \theta_m)^T \Delta\theta_m$;
return $\text{GHN}(\cdot; \phi)$;
ClientUpdate$(m, \tilde{\theta}_m)$:
$\theta_m \leftarrow \tilde{\theta}_m$;
for $e = 1, \ldots, E$ **do**
  Partition $\mathbf{D}_m$ into mini-batches $\bigcup_{i=1}^{j_m} B_i^{(m)}$;
  for $i = 1, \ldots, j_m$ **do**
    $\theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} \mathscr{L}_m(\theta_m; B_i^{(m)})$;
return θ_m to server ;

where N_m is the number of samples in the local dataset $\mathbf{D}_m$. Let η_c be the learning rate for local training at the client. Starting with the initial value $\theta_m = \tilde{\theta}_m$, the client updates θ_m as follows:

$$\theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} \mathscr{L}_m(\theta_m). \tag{4.3}$$

After local training, the clients send the updated model weights back to the server. The server then calculates the change in local model parameters $\Delta\theta_m = \theta_m - \tilde{\theta}_m$, and uses the chain rule $\nabla_\phi \mathscr{L}_m = (\nabla_\phi \theta_m)^T \nabla_{\theta_m} \mathscr{L}_m$ to update the graph hypernetwork parameter ϕ:

$$\phi \leftarrow \phi - \eta_s \sum\nolimits_{m \in \mathbf{S}_t} (\nabla_\phi \theta_m)^T \Delta\theta_m, \tag{4.4}$$

where η_s is the learning rate for updating the graph hypernetwork. By adopting the graph hypernetwork, we modify Equation 4.1 and formulate the new learning objective as:

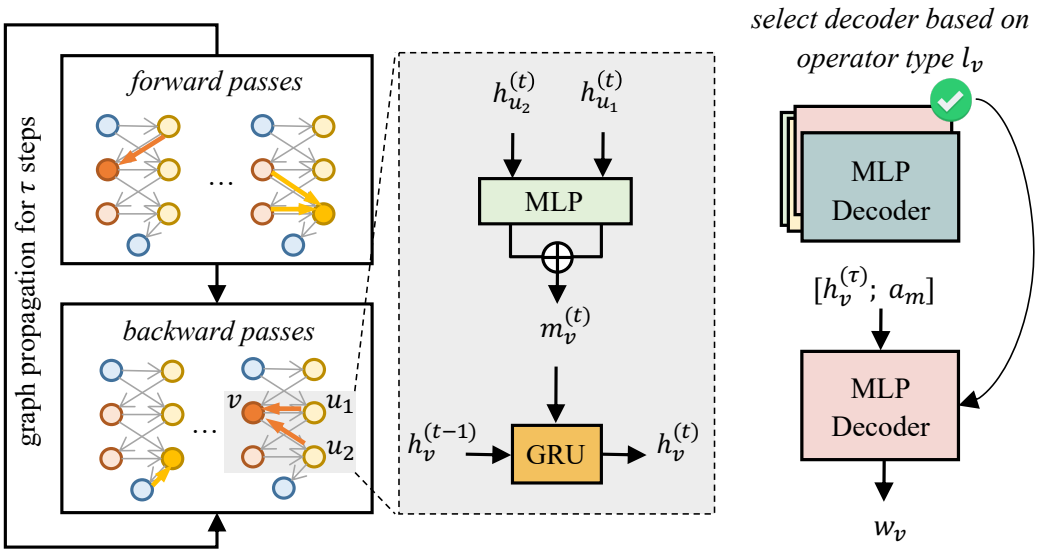
$$\text{argmin}_\phi \hat{\mathscr{L}}(\phi, \boldsymbol{D}), \tag{4.5}$$

where $\hat{\mathscr{L}}(\phi, \boldsymbol{D}) = \frac{1}{M} \sum_{m=1}^{M} \mathscr{L}_m(\text{GHN}(\mathscr{G}_m, a_m; \phi))$ is the average empirical loss on dataset $\boldsymbol{D} = \{\boldsymbol{D}_m\}_{m=1}^{M}$.

Importantly, the graph hypernetwork resides on the server, with all computations related to the graph hypernetwork executed solely by the server. This design ensures it does not impose additional communication or computational overhead on clients.

4.3.2 Weight Generation with Graph Hypernetwork

Representation of target network architectures. We represent the computational graph of a neural network model as a directed acyclic graph, denoted as $\mathscr{G}(\mathscr{V}, \mathscr{E})$, where the nodes $\mathscr{V}$ are the operators (e.g., convolution, pooling, linear layer, etc.) and the directed edges $\mathscr{E}$ describe the computation flow in the order of forward propagation among the operators. Conventional graph hypernetworks are inefficient in dealing with repeated similar local connection patterns in



(a) Encoding **(b)** Decoding

Figure 4.3. Graph hypernetwork architecture in RECIPFL.

deep networks such as the ResNet blocks in ResNet-152. To enhance the ability to distinguish local connection patterns in target networks, we inform the graph hypernetwork about the model parameters of the target network at the current training round. To do so, the node features $\{h_v|v \in \mathcal{V}\} = \{[l_v, q_v]|v \in \mathcal{V}\}$ consist of two parts: (1) one-hot vectors l_v indicating the operations performed by the node, and (2) the current parameters q_v of the operators. A linear embedding layer transforms the one-hot vector l_v to a dense vector and a Transformer encoder [192] maps the variable-length node parameters q_v to a fixed-dimensional vector. The linear embedding layer and the Transformer encoder are learnable and are updated during training.

Graph hypernetwork architecture. As depicted in Figure 4.3, the graph hypernetwork consists of an encoding process that extracts features from node information and a decoding process that predicts weights for parametric operators according to the encoded features.

In the encoding phase, a graph neural network (GNN) [166, 115] is employed to conduct τ steps of graph propagation within $\mathcal{G}(\mathcal{V}, \mathcal{E})$ of the target network. During this graph propagation, the GNN topologically traverses the nodes in both forward and backward directions, iteratively conducting message passing and updating node features. For the t-th propagation step, the GNN

first forward traverses nodes. Every node v receives messages from its incoming nodes and sends messages to its outgoing nodes. Denote the incoming nodes to node v as $\mathbf{IN}(v)$. The message function is modeled with an MLP shared among all the nodes. The message received by node v at step t is:

$$m_v^{(t)} = \sum_{u \in \mathbf{IN}(v)} \text{MLP}(h_u^{(t)}) \tag{4.6}$$

The node feature vector $h_v^{(t)}$ is then updated based on the aggregated message $m_v^{(t)}$ and the feature vector of node v at step $t-1$ using a Gated Recurrent Unit (GRU) cell [28]:

$$h_v^{(t)} = \text{GRU}(h_v^{(t-1)}, m_v^{(t)}) \tag{4.7}$$

After traversing $\mathscr{G}(\mathscr{V}, \mathscr{E})$ in forward propagation, the GNN reverses the traversal direction and updates the node features again, i.e. receives messages from its incoming nodes along backward passes and sends to its outgoing nodes.

In the decoding phase, we use an individual MLP as the decoder for each type of parametric operator to generate parameters. To further support personalization, we introduce client descriptors $\{a_m | m \in [M]\}$ that describe the data characteristics of every client m. This descriptor is provided as input to the MLP decoder. Specifically, we use the class distribution of local training samples as the client descriptor. Alternatively, the client descriptor can simply be the client IDs, and in that case, a linear embedding layer can be used to transform them into client embeddings, enabling the learning of client features through training. Let $\text{MLP}_l(\cdot)$ represent the decoder for the l-type operator. MLP_l operates on the concatenation of the node embedding and the client embedding, denoted as $[h_v^{(\tau)}, a_m]$, and generates parameters for the node. The resulting set of generated weights for the target network is:

$$w = \{w_v | v \in \mathscr{V}\} = \{\text{MLP}_{l_v}([h_v^{(\tau)}, a_m]) | v \in \mathscr{V}\} \tag{4.8}$$

To handle different dimensionalities of layers within the same operator type, the outputs

of the decoder are reshaped through tiling and concatenation to match the shape of the target layers following common practices in graph hypernetworks [85, 227].

4.3.3 Strong-to-Weak Device Knowledge Transfer

To further enhance the learning of small models, we leverage the computing resources on strong devices and employ regularizations to distill knowledge from large models to small ones.

For strong devices, we let the central graph hypernetwork generate weights for both small and large models. Denote the small and large model at the strong device m as f_m^S and f_m^L respectively and the corresponding model parameters are θ_m^S and θ_m^L. After training the large model f_m^L, we proceed to train the small model and distill knowledge from the large one. We introduce an additional cross-entropy loss term $CE(\cdot)$ to let the small model mimic the prediction probabilities of the large model. In addition, if the representations generated by the last hidden layers of the models have the same dimension, we add a KL-divergence loss term $D_{KL}(\cdot)$ to align the feature spaces. Denote the softmax probability distributions of features generated by the last hidden layers of the small model and the large model as p_i^S and p_i^L respectively. The optimization objective for training small model f_m^S on strong device m is to minimize the following loss:

$$\mathscr{L}_m^S(\theta) = \frac{1}{n}\sum_{i=1}^{n}[CE(f_m^S(x_i;\theta_m^S), y_i) + CE(f_m^S(x_i;\theta_m^S), f_m^L(x_i;\theta_m^L)) + D_{KL}(p_i^L \| p_i^S)]. \tag{4.9}$$

After local training, the updated weights of both small and large models are sent to the server and used for the update of the graph hypernetwork. This knowledge transfer mechanism helps small models benefit from the insights learned by strong devices.

4.4 Analysis

In this section, we establish the generalization bound of RECIPFL.

Consider a training set on clients $\mathbf{D}_m = \left\{\left(x_i^{(m)}, y_i^{(m)}\right)\right\}_{i=1}^{N}$ for some natural number $N \geq 1$, i.e. we sample uniformly N training data from each data distribution $\mathscr{P}_m$ on client m for

$m = 1, \cdots, M$.

Assume the loss function ℓ takes value in $[0,1]$, or equivalently with rescaling, ℓ is bounded. Let d be the dimension of the hypernetwork parameter ϕ and assume $\phi \in [-R,R]^d$ for some large $R > 0$. Finally, assume the loss l is Lipschitz with respect to ϕ with Lipschitz constant $K > 0$, i.e. $|\ell\left(f_m\left(x; \mathrm{GHN}(\mathscr{G}_m, a_m; \phi)\right), y\right) - \ell\left(f_m\left(x; \mathrm{GHN}(\mathscr{G}_m, a_m; \phi')\right), y\right)| \leq K\|\phi - \phi'\|$ for all x, y and $m = 1, \ldots, M$. Here $\|\cdot\|$ denotes the Euclidean distance on $\mathbb{R}^d$. Define the expected loss as:

$$\mathscr{L}(\phi) = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{(x,y)\sim\mathscr{P}_m} \ell(f_m(x; (\mathscr{G}_m, a_m; \phi)), y). \tag{4.10}$$

Theorem 4.4.1. *If the number of samples on each client satisfies*

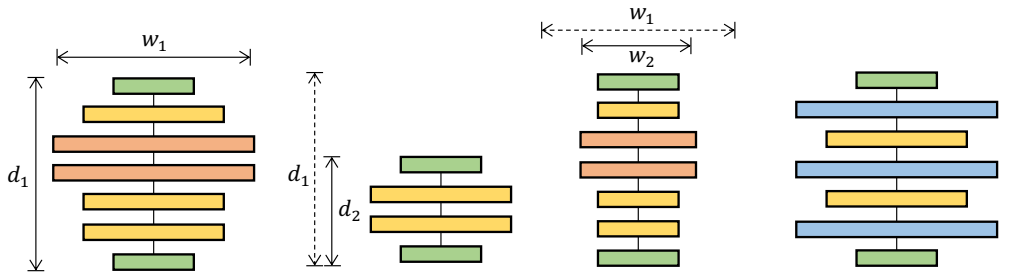
$$N \geq \max\left\{ \frac{4d}{M\varepsilon^2} \log\left\lceil \frac{4RK\sqrt{d}}{\varepsilon} \right\rceil + \frac{4}{M\varepsilon^2}\log\frac{4}{\delta}, \frac{1}{\varepsilon^2} \right\}, \tag{4.11}$$

then with probability at least $1-\delta$ with respect to the probability distribution on $\boldsymbol{D} = \{\boldsymbol{D}_m\}_{m=1}^M$, $\mathscr{L}(\phi) < \hat{\mathscr{L}}(\phi, \boldsymbol{D}) + \varepsilon$ for every ϕ.

The proof and more details are given in Appendix A.3. From Equation 4.11, we observe that the number of training samples N per device required for generalization is negatively related to the number of devices M, which suggests that introducing new weak devices to the system can help lower the threshold for generalization. Moreover, when there is a strong device possessing a large amount of data, it can also lower the threshold for weak devices. For example, if there is one strong device and M weak devices, we can regard the strong one as k virtual devices, which increases the total number of devices to $M+k$, and thereby lowers the threshold for the number of samples on weak devices. The only requirement is that the strong device then needs to take on k times more data samples than that is required for a weak device.

Table 4.1. RECIPFL is compatible with various ways of model scaling, showing more flexibility than existing solutions.

Scaling Strategy	HeteroFL	InclusiveFL	FlexiFed	DepthFL	RECIPFL (ours)
depth-wise		✓	✓	✓	✓
width-wise	✓				✓
architecture-wise					✓



(a) original model (b) depth-wise (c) width-wise (d) architecture-wise

Figure 4.4. Illustration of model scaling strategies. The rectangle blocks represent the layers in neural networks. Different colors indicate different operations (e.g., convolution).

4.5 Experiments

4.5.1 Experiment Setup

Configurations are summarized in Table 4.2. We provide details below.

Datasets. We evaluate RECIPFL on two fundamental categories of machine learning tasks: image classification with CIFAR-10 [90], CIFAR-100 [90], MNIST [95], and natural language inference with MNLI [209]. We simulate quantity skew where strong devices possess a dominant amount of data, as it often occurs in realistic resource-skewed environments. We allocate 50% of the entire dataset to the strong devices, while the weak devices evenly share the remaining half. This ensures the total amount of the data owned by weak devices is comparable to that owned by strong devices, making it possible for weak devices to contribute to the model enhancement of strong devices. Note that we conduct exploration studies in Section 4.5.5 to investigate the impact of data ratio by changing this configuration. To simulate non-IID client distributions, we follow the prior work [64, 82] and employ Dirichlet distribution $Dir(\alpha = 0.5)$

Table 4.2. Federated learning configurations.

Dataset	# of devices		Data allocation		Large model	# of parameters			Pretrained?
	Strong	Weak	Strong	Weak		Original	Depth-scaled	Width-scaled	
CIFAR-10	1	5	50%	10%	ResNet-18	11M	450K	444K	✗
CIFAR-100	1	50	50%	1%	DenseNet-121	1M	258K	276K	✗
MNIST	2	500	25%	0.1%	LeNet-5	44K	5.6K (scaled in depth & width)		✗
MNLI	1	20	50%	2.5%	BERT	110M	67M (DistilBERT)		✓

to sample the class distribution for every client. For the strong device, we assume it follows the universal distribution due to its substantial data volume.

Model architectures. To evaluate the robustness of our method, we experiment with various popular neural network architectures. The large models include ResNet-18 [61], DenseNet-121 [66], LeNet-5 [95] and BERT [32]. Our method is compatible with different ways of model scaling and we test all three scaling strategies shown in Figure 4.4. For depth-scaling, we follow [82] and regard the first block of ResNet-18 and DenseNet-121 as the small models. For width-scaling, we follow [33] and shrink the channels and hidden layers of the large model based on a scaling ratio. In order to achieve comparable model sizes with depth-scaling, we carefully set the scaling ratio for width-scaling by comparing the parameters in the depth-scaled models to those in the large models. In addition, we craft a smaller version of LeNet-5 which reserves the first block of LeNet-5 and scales the rest layers along the width. By doing this, we enable both depth- and width-wise aggregation for the comparison of existing methods. For architecture-wise scaling, We use DistilBERT [165] as the smaller version of BERT.

Enabling fine-tuning from pretrained models. RECIPFL can support fine-tuning by inserting adapters [63], a small set of new parameters, and classification heads into pretrained models. During training, only the adapters and classification heads are updated and communicated between the server and clients, while the other layers are fixed at local. We initialize BERT[1] and DistilBERT[2] with pretrained weights provided by HuggingFace.

[1]https://huggingface.co/bert-base-uncased

[2]https://huggingface.co/distilbert-base-uncased

Compared methods. First, we construct two naive baselines based on the classical federated learning algorithm FedAvg [138]:

- **AllSmall**: All clients deploy the small models to compromise the smallest running capacity and conduct federated learning.
- **ExclusiveFL**: Clients with the same level of capacity are equipped with the same model, i.e., strong devices deploy large models while weak devices deploy small models. Each type of device performs weight aggregation exclusively.

The performance of weak devices under AllSmall and that of strong devices under ExclusiveFL serve as reference points for assessing whether a method enhances the performance of weak or strong devices. We then compare RECIPFL with state-of-the-art methods for federated learning with heterogeneous models:

- **HeteroFL** [33] adopts width-scaling where channels and hidden layers are scaled according to a fixed ratio. The global layer updates a subset of parameters correspondingly from scaled layers and all parameters from unscaled layers by weight averaging.
- **FlexiFed** [200] identifies common base layers across client models and clusters personal layers into groups. The same group of personal layers have identical operations and sizes. Then, it fuses the knowledge contained in common base layers and clustered personal layers by weight averaging.
- **InclusiveFL** [127] adopts depth scaling. The shared layers are aggregated via weight averaging. It also distills knowledge from the classifier of the large model to its shallow counterpart by calculating a gradient momentum as the average over updates of the deep layers (pruned in the small model) in the large model and injecting it to the last encoding layer in the small model.
- **DepthFL** [82] scales the large model along the depth and creates local models with multiple classifiers at different depths. The shared layers are averaged for aggregation. It is further

equipped with a self-distillation strategy to transfer knowledge among deep and shallow classifiers if available at local. For inference, the client uses the ensemble of all internal classifiers.

Table 4.1 showcases the downscaling strategies that prior methods are designed for. For architecture-wise scaling, these methods identify common layers (e.g. classification heads) for aggregation.

Federated learning configuration. We evaluate each client model on its respective test data drawn from the client's data distribution. To achieve personalization, for all methods, we fine-tune client models on their local training dataset for one round after receiving parameters from the server. Communication rounds T are set based on the convergence rate of each task. Specifically, we set $T = 50$ rounds for CIFAR-10 and MNLI, $T = 100$ rounds for MNIST, and $T = 500$ rounds for CIFAR-100. At each round, the server randomly selects $|\mathbf{S}_t| = \min(M, 10)$ clients. Since RECIPFL trains both small and large models on strong devices for knowledge transfer, we also train both types of models on strong devices for compared methods (i.e., FlexiFed, HeteroFL, InclusiveFL, and DepthFL) to ensure a fair comparison. During evaluation, only the target client model is evaluated. The experiments are repeated 5 times. Following [113], we report the average accuracy and standard deviations of the last 20% rounds for strong and weak devices respectively.

4.5.2 Main Results and Analysis

The experiment results are presented in Table 4.3. Note that the average accuracy on weak devices may appear higher than that on strong devices since the evaluation is based on every client's data distribution and the weak devices may only have a small subset of classes, making it easier to get higher accuracy. We observe that no scaling strategy consistently outperforms others. For example, width-scaling works better than depth-scaling on CIFAR-10 and CIFAR-100 with the ResNet and DenseNet architectures but it (i.e., the result of HeteroFL) lags depth-scaling on MNIST with LeNet. With architecture-wise scaling on the MNLI dataset, HeteroFL

Table 4.3. Experiment results (average accuracy and standard deviation). RECIPFL consistently outperforms the compared methods across all datasets and model scaling strategies, benefiting both strong and weak devices.

Scaling	Method	CIFAR-10		CIFAR-100	
		Strong	Weak	Strong	Weak
Depth	AllSmall	64.34±2.14	68.50±3.42	17.86±2.56	25.56±3.11
	ExclusiveFL	84.85±1.85	59.11±4.22	32.21±3.81	19.22±2.07
	FlexiFed [200]	82.86±1.77	67.66±3.93	28.60±3.48	27.84±2.98
	InclusiveFL [127]	83.22±0.47	67.66±3.14	18.98±3.49	28.71±2.87
	DepthFL [82]	73.90±1.49	78.16±1.48	22.08±3.58	36.83±2.87
	RECIPFL	**85.28±0.22**	**78.65±1.35**	**41.63±2.24**	**45.52±3.12**
Width	AllSmall	82.86±1.77	78.90±2.87	29.80±3.32	37.90±2.83
	ExclusiveFL	83.96±1.97	70.65±3.99	32.22±6.66	24.49±3.52
	HeteroFL [33]	84.76±1.19	77.93±2.92	26.51±2.70	39.05±2.82
	RECIPFL	**85.06±0.13**	**82.88±1.29**	**43.64±2.84**	**42.00±3.88**

Method	MNIST		MNLI	
	Strong	Weak	Strong	Weak
AllSmall	91.73±3.94	77.05±7.47	73.47±0.52	82.13±2.89
ExclusiveFL	92.97±1.98	77.85±5.14	80.20±0.20	70.52±6.04
FlexiFed [200]	92.70±2.72	73.89±8.08	79.65±0.18	82.31±6.15
InclusiveFL [127]	84.77±3.12	75.72±6.27	79.87±0.30	81.17±4.31
DepthFL [82]	94.33±1.95	78.39±7.52	77.11±0.90	80.92±6.64
HeteroFL [33]	89.53±3.22	75.95±8.01	79.65±0.18	82.31±6.15
RECIPFL	**97.07±1.87**	**86.36±6.60**	**82.78±0.57**	**83.37±4.72**

and FlexiFed become equivalent, since the fine-tuning layers, which are positionally aligned across models, have identical sizes (i.e., their scaling ratio is 1). RECIPFL outperforms the compared methods across all datasets, regardless of the model scaling strategies, demonstrating its capability to generalize knowledge across different model architectures. Notably, RECIPFL shows its ability to improve the model performance on both strong devices and weak devices. Moreover, RECIPFL also outperforms the baselines in fine-tuning from the pre-trained weights of BERT and DistilBERT. Figure 4.5 shows the performance of all compared methods with the increase in communication round. We observe that RECIPFL often achieves higher accuracy in fewer rounds compared to the baseline methods.

4.5.3 Ablation Study

In Section 4.3.3, we introduced the knowledge transfer mechanism within our RECIPFL method to enhance the performance of weak devices. To assess its effectiveness, we craft an ablated version of RECIPFL without knowledge transfer, denoted as **RECIPFL w/o KT**, and evaluate the model performance on weak devices across four datasets. The CIFAR datasets are tested under the depth scaling setting as examples. We compare the performance of RECIPFL, RECIPFL w/o KT, AllSmall, and DepthFL (the best-performing baseline with depth scaling). As shown in Figure 4.6, RECIPFL w/o KT already exhibits significant improvements over

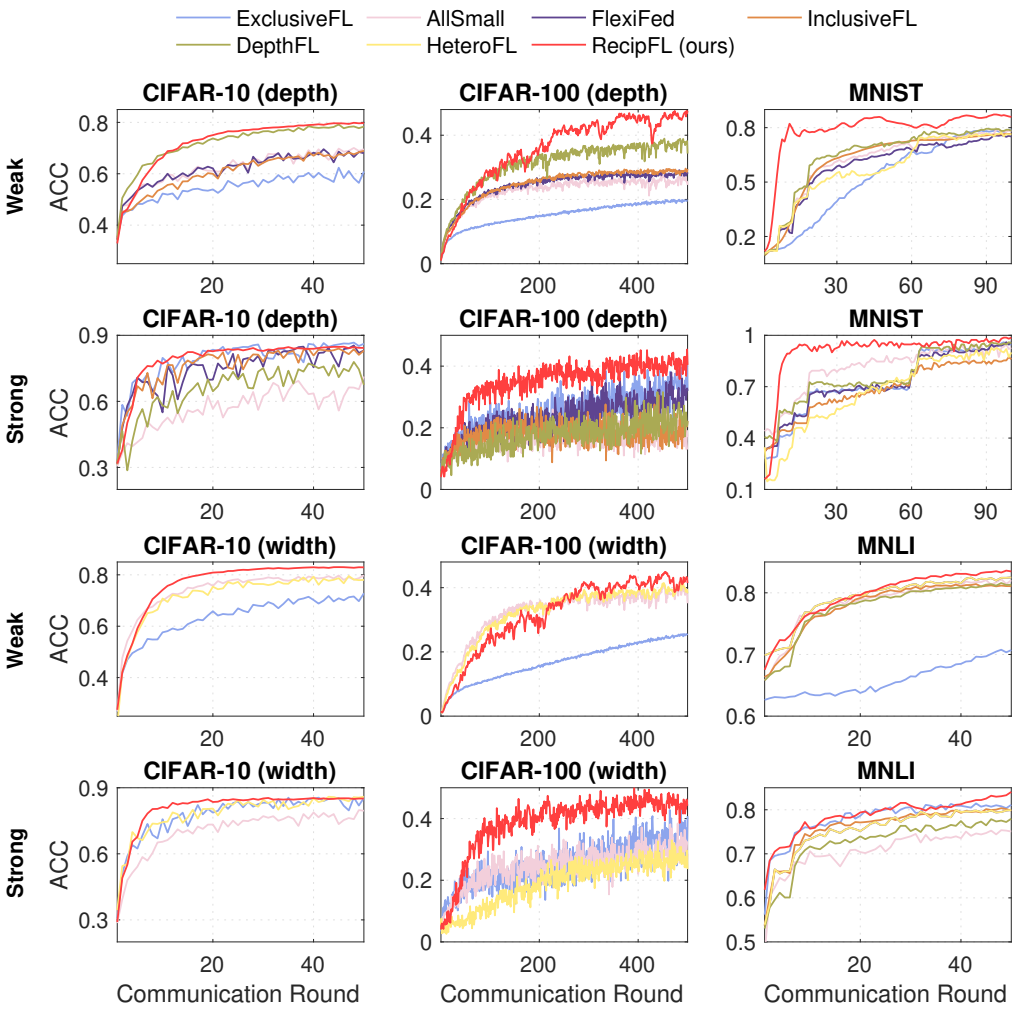


Figure 4.5. Performance w.r.t. communication round.

the naive baseline AllSmall, and it can often outperform the state-of-the-art method DepthFL. However, the comparison between RECIPFL and RECIPFL w/o KT indicates that integrating knowledge transfer leads to even better small models. The knowledge (i.e., prediction and feature distribution) from strong devices contribute to this improvement.

4.5.4 More Diverse Device Capacities

RECIPFL is not limited to the setup of one large and one small model architecture and can work with diverse device capacities. To support this claim, we conduct an experiment involving a more heterogeneous set of network architectures: 16 small clients training LeNet-5, three

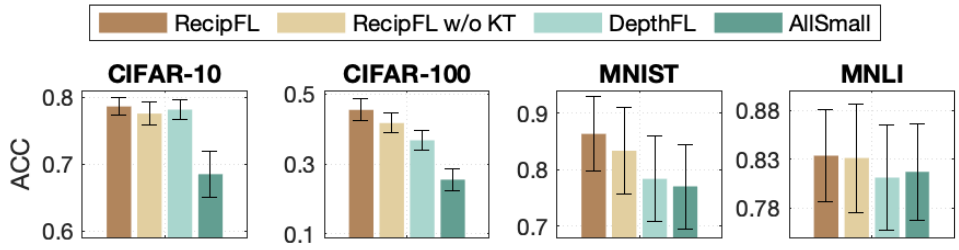


Figure 4.6. Ablation study: performance of weak devices.

Table 4.4. Performance with more diverse device capacities.

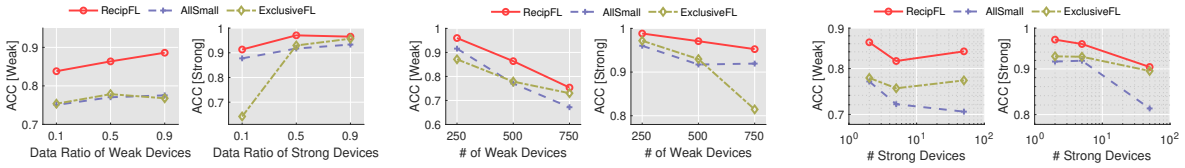
Method	Small (LeNet-5)	Medium (ResNet-101)	Large (VGG-16)
AllSmall	69.22±2.16	66.58±1.84	59.06±1.41
ExclusveFL	46.00±3.71	72.38±3.80	79.86±2.52
RECIPFL	**70.12±2.33**	**86.37±1.72**	**81.23±0.41**

medium clients training ResNet-101, and one large client training VGG-16. With RECIPFL, strong and medium clients conduct knowledge transfer for models smaller than their respective capacities. We use the CIFAR-10 dataset as an example, with data partitioning of 15% to small clients, 35% to medium clients, and 50% to the large client. The non-IID data on small and medium clients are sampled using Dirichlet distributions $Dir(\alpha = 0.5)$. As shown in Table 4.4, RECIPFL improves performance on each type of device compared to AllSmall and ExclusiveFL.

4.5.5 Exploratory Studies

To get deeper insights into the performance of the federated systems under various resource skew conditions, we conduct exploratory studies using the MNIST dataset with LeNet models.

Impact of data ratio between two types of devices. We aim to understand how much data on weak devices is necessary to help improve strong devices. We vary the ratio of data allocated to each type of device to the entire dataset among $\{0.1, 0.5, 0.9\}$. The number of devices remains the same as in the main experiment, i.e., 500 weak devices and 2 strong devices. Results are presented in Figure 4.7a. On both weak and strong devices, models perform better



(a) Impact of data ratio **(b)** Varying number of weak devices **(c)** Varying number of strong devices

Figure 4.7. Exploratory studies. RECIPFL exhibits superior scalability and robustness across a range of resource skew scenarios compared to the baselines, consistently enhancing the performance of both strong and weak devices.

with higher data ratios. It is worth noting that with less than 50% data allocated to strong devices, the performance gap between RECIPFL and ExclusiveFL becomes more evident. This suggests when weak devices hold comparable data amounts to strong devices, they are more likely to contribute significantly to strong devices.

Scalability and skewness. To evaluate the scalability of the federated systems, we first vary the number of weak devices among $\{250, 500, 750\}$. The number of strong devices and the data ratio are kept the same as in the main experiments, where the two large devices own 50% of the whole dataset and the weak devices share the rest. The results are shown in Figure 4.7b. When increasing the number of weak devices, the strong devices get selected for local updates less frequently. Consequently, within the same communication rounds, the performance of large models degrades. Meanwhile, with fewer data allocated per weak device (as all weak devices collectively share 50% of the dataset), the performance of small models also declines. Despite these challenges, RECIPFL demonstrates an impressive ability to memorize model parameters and generalize them across different architectures. As a result, even with reduced client sampling ratios, clients still achieve better performance compared to AllSmall and ExclusiveFL. This suggests RECIPFL is more scalable than the baselines. Then, we increase the number of strong devices from 2 to 5 and 50 while keeping weak devices at 500. The strong devices equally share 50% of the entire dataset. As shown in Figure 4.7c, RECIPFL always achieves better performance than the two baselines on both strong and weak devices. These results highlight the robustness of RECIPFL in different levels of skewness.

4.6 Related Work

Federated learning with heterogeneous models. Traditional federated learning methods [88, 89, 111, 80, 236] have primarily focused on homogeneous models across devices. These methods often fail to address the inherent system heterogeneity found in real-world edge computing environments. Recent studies of federated learning in the context of diverse computational capacities have proposed novel approaches that facilitate collaboration among heterogeneous models [237, 103, 39, 4], focusing on two directions: (1) how to scale the large model and (2) how to effectively aggregate the models with different sizes. In the first direction, methods are proposed to prune the model along depth by pruning the deepest layers [127, 82] or along layer width by scaling the width of hidden channels [102, 147, 33]. In the second direction, typical practices [200] are to identify shared patterns (e.g., layers) in local models and aggregate the common parts. Recent methods like InclusiveFL [127] and DepthFL [82] further leverage knowledge distillation for transferring knowledge among deeper layers and shallow layers to enhance the performance of small models. These approaches have shown promise in accommodating device-specific requirements and resource constraints. However, the reliance on a particular scaling strategy and the naive weight averaging-based aggregation constrain model performance in the presence of resource skew. Our work introduces a more effective way to generalize knowledge across different models by training a graph hypernetwork.

Hypernetworks in Federated Learning. Hypernetworks [56] have demonstrated the potential in meta-learning scenarios [193], facilitating fast adaptation to new tasks, as they capture the common knowledge among tasks via the weight generation mechanism. Prior work [170] has explored its use in federated learning by training a hypernetwork at the server to generate personalized model weights while preserving the effective parameter-sharing feature of hypernetworks. This previous work uses a linear-structured hypernetwork that only works with homogeneous model architectures. Graph hypernetwork [227, 85] was originally proposed for neural architecture search as it can effectively encode the computational graph information of

various neural networks. There has been an initial try on leveraging graph hypernetworks for generating weights across different client models [122]. However, the prior work trains local hypernetworks at clients and aggregates them by weight averaging at the server following a typical federated training process which requires high computational budgets at clients and is impractical for resource-constrained devices. In contrast, RECIPFL equips the graph hypernetwork at the server and we design ways to update the graph hypernetwork based on predicted weights and clients' feedback. The computations of hypernetwork are executed only by the server and therefore do not add any additional overhead to the edge devices.

4.7 Summary

We study the problem of federated learning in the presence of resource skew among devices, specifically, when the majority are weak devices and there are only limited (1 or 2) strong devices. We show that existing methods do not guarantee performance improvement for both types of devices. We propose RECIPFL, training a central graph hypernetwork that enables the collaboration of clients with heterogeneous model architectures to fit specific running capacities. RECIPFL is agnostic to model scaling strategies and can generalize knowledge about model weights across different neural network architectures. RECIPFL outperforms state-of-the-art methods with significant margins and demonstrate that even weak devices can contribute effectively to the learning system, providing both devices with an incentive to participate. In future work, we plan to design mechanisms to adaptively adjust the model size in response to the dynamic changes in the running capacity of devices caused by user usage. This will enable efficient utilization of computing resources during learning. Together with our proposed method, we anticipate our solutions will create a viable, more powerful, and useable alternative to current large model services, alleviating privacy and efficiency concerns by facilitating edge-based learning without the need to transmit user input to central servers.

Chapter 4 incorporates material from the publication "How Few Davids Improve One

Goliath: Federated Learning in Resource-Skewed Edge Computing Environments", by Jiayun Zhang, Shuheng Li, Haiyu Huang, Zihan Wang, Xiaohan Fu, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang, published in Proceedings of the ACM Web Conference 2024. The dissertation author was the primary investigator and the lead author of this paper.

Chapter 5

Asynchronous Aggregation for Efficient Learning

In this chapter, we adress system heterogeneity from another aspect—synchronization. While the previous chapter enables collaboration among devices with varying model architectures, the distributed nodes still exhibit significant divergences in training speed, leading to considerable idle time and inefficient resource utilization. To mitigate this issue, we introduce an asynchronous federated learning approach that allows nodes to contribute updates at their own pace without waiting for slower participants. This approach accelerates the training process, enhancing efficiency and scalability in distributed environments.

Asynchronous federated learning mitigates the inefficiency of conventional synchronous aggregation by integrating updates as they arrive and adjusting their influence based on staleness. Due to asynchrony and data heterogeneity, learning objectives at the global and local levels are inherently inconsistent—global optimization trajectories may conflict with ongoing local updates. Existing asynchronous methods simply distribute the latest global weights to clients, which can overwrite local progress and cause model drift. In this work, we propose ORTHOFL, an orthogonal weight calibration mechanism that decouples global and local learning progress and adjusts global shifts to minimize interference before merging them into local models. In ORTHOFL, clients and the server maintain separate model weights. Upon receiving a client update, the global weights are updated via a moving average. For client weights, the

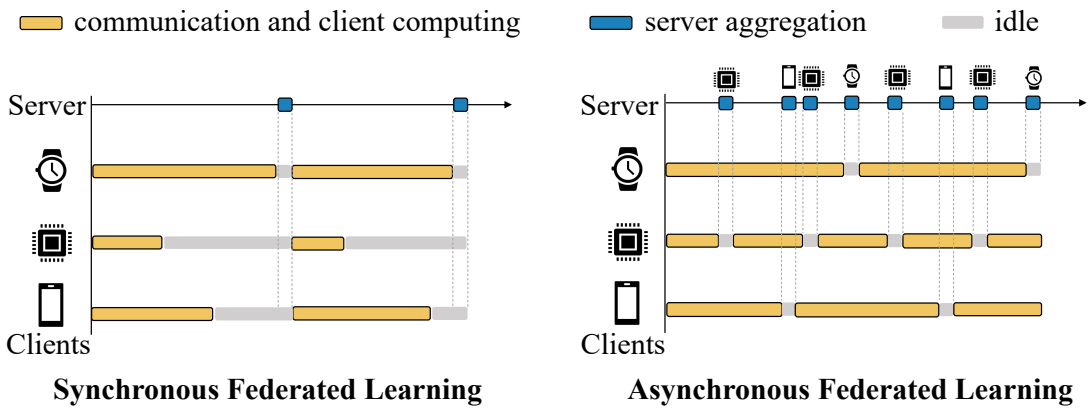


Figure 5.1. Time synchrony in federated learning. Asynchronous methods reduce idle time and improve resource utilization, suited for large-scale heterogeneous environments.

server calculates the global weight shift accumulated during the client's delay and removes the components aligned with the direction of the received update. The resulting parameters lie in a subspace orthogonal to the client update and preserve the maximal information from the asynchronous global progress. The calibrated global shift is then merged into the client's model weights for further training. Extensive experiments on multiple tasks show that ORTHOFL improves accuracy by 9.6% over synchronous methods and achieves a 12× speedup. Moreover, ORTHOFL consistently outperforms state-of-the-art asynchronous baselines under various delay patterns and heterogeneity scenarios.

5.1 Introduction

The most widely adopted federated learning protocols [138, 111, 80, 109, 199, 236, 231] follow a *synchronous* update procedure. In each round, the server waits for all selected clients to finish local training before aggregating their updates. This synchronization becomes inefficient under heterogeneous resource conditions, where clients differ in computational power, network bandwidth, and data volumes, due to distinct device configurations and user-system interaction patterns. Therefore, synchronous aggregation can lead to poor resource utilization and extended convergence time.

Asynchronous federated learning offers an alternative approach that aggregates client

updates as they arrive (illustrated in Figure 5.1), reducing idle time caused by slower clients. In this setting, when a client is performing local training, the server continuously aggregates updates from other clients, shifting the global model to new states. By the time the client's update reaches the server, it may be stale. Existing methods manage such staleness by applying a decay factor to updates before aggregating them into the global model [213, 125, 226, 179]. The updated global model is directly returned to the clients for further training. Although down-weighting a stale update reduces its negative impact on global progress, it also diminishes the integration of meaningful knowledge from the client. Moreover, due to data heterogeneity, the optimization objectives of the global and client models are inherently inconsistent—while the global model aims to optimize for the overall data distribution, individual clients minimize loss on their local data. Distributing the latest global parameters to clients for subsequent training can introduce conflicts with their local optimization steps, potentially reversing local gains and leading to oscillations in training.

To address the challenge, we propose to decouple global and local learning progress and calibrate the directions of weight shifts to reduce interference during client weight merging. The key insight is that, in the high-dimensional parameter space of neural networks, there are multiple viable directions for effective optimization [211]. Some of these directions severely disrupt performance on previously learned distributions, while others have little impact. This opens an opportunity to avoid disruptive components in asynchronous update directions and preserve both global progress and client-specific contributions.

We introduce ORTHOFL, an orthogonal weight calibration mechanism for asynchronous federated learning. Our design is motivated by two goals: (1) minimizing interference between global and local optimization by sharing information from the global weight shift perpendicular to client updates and (2) selecting the most informative direction within the orthogonal hyperplane to maximize knowledge sharing. Specifically, ORTHOFL maintains separate global and client model weights to accommodate their distinct optimization objectives. When the server receives a client's update, the global weights are updated via a moving average with an adaptive decay

factor accounting for staleness. For client model aggregation, ORTHOFL identifies the global weight shift (induced by other clients) during the client's delay. It projects this shift onto the direction of the received client update and subtracts this projected component. The remaining parameters lie in a subspace orthogonal to the client's update. Through analysis, we show that this orthogonal calibration strategy keeps maximal global progress while minimizing interference with the client's local update. The calibrated global shift is then merged with the client model for local training.

We evaluate ORTHOFL on five datasets of different application scenarios including image classification, text classification, and human activity recognition. Our evaluations incorporate realistic delay distributions to reflect the heterogeneous nature of real-world deployments. Results show that ORTHOFL achieves an averaged 9.6% accuracy improvement across the datasets compared to the synchronous methods within the same training time and a $12\times$ speedup in reaching a target accuracy. Moreover, it outperforms state-of-the-art asynchronous baselines. We also perform exploratory studies with various simulated delay distributions and data heterogeneity levels to understand their impact on model performance and convergence speed. In summary, our contributions are as follows:

- We analyze the key challenges of asynchronous federated learning—the inconsistency of global and local objectives and the detrimental effect of stale updates in heterogeneous environments.
- We propose a novel orthogonal calibration method, ORTHOFL, that maintains separate global and local model weights. It projects global shifts onto orthogonal subspaces of local updates before sharing them with clients. This approach reduces interference, preserves meaningful contributions from both global progress and local updates, and enhances knowledge sharing.
- We demonstrate the effectiveness and robustness of ORTHOFL through comprehensive experiments on multiple datasets and various delay scenarios, providing insights on practical design considerations for large-scale federated learning systems.

5.2 Preliminaries

In this section, we define the asynchronous system architecture and present a motivating study to analyze the key challenges.

5.2.1 Asynchronous System Architecture

In an asynchronous federated learning setup, a central server coordinates the training of a global model W using data distributed across M clients. Each client $m \in \{1, 2, \ldots, M\}$ possesses its own local dataset $\mathbf{D}_m$. The data distribution of client m is denoted as $\mathscr{P}_m$. The objective is to train a global model W that generalizes well across the combined data distribution of all clients. Formally, we aim to solve the following optimization problem:

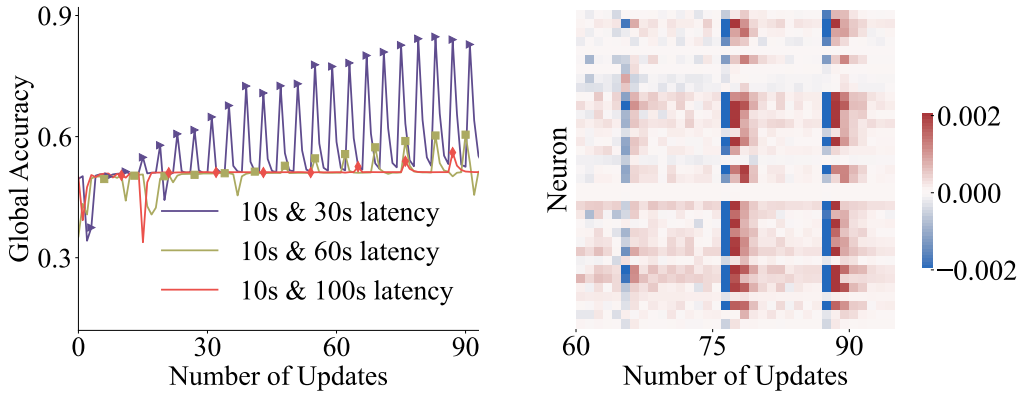
$$W^* = \arg\min_W \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{(x,y)\sim\mathscr{P}_m} \ell\left(f(x;W), y\right), \tag{5.1}$$

where W denotes the global weights, ℓ the loss function, and $f(x;W)$ the prediction of the model on data x with model weights W.

Clients perform local training and communicate their updates to the central server at different times. Let T be the number of global rounds. For $t \in \{1, \ldots, T\}$, denote $m_t \in \{1, \ldots, M\}$ as the client that communicates with the server at the t-th round, and τ_t as the round when client m_t last communicated with the server. We define the staleness of the client update as follows:

Definition 1 (Staleness)**.** *Staleness quantifies the delay between a client's updates, representing the number of global rounds since the client last communicated with the server. Formally, let t be the current global round, and τ_t the global round when the server last received updates from client m_t. The staleness of client m_t is defined as $t - \tau_t$, where $t - \tau_t \geq 1$, with a staleness of 1 indicating no delay.*

For simplicity, we will drop the subscripts on m_t and τ_t with no ambiguity from now on.



(a) Global accuracy w.r.t. update. Markers represent updates from the slow client.

(b) Global weight change w.r.t. update. Colors indicate update directions.

Figure 5.2. A motivating study with a fast client (10s latency) and a slow client (30/60/100s) assigned non-overlapping classes. Objective inconsistency causes fluctuations in global accuracy and oscillations in weight update directions.

5.2.2 A Motivating Study

We conduct an experiment on MNIST [31] with a LeNet5 [95] model to analyze the challenges in asynchronous federated learning. We simulate the scenario with two clients: one with a 10-second latency and the other with 30, 60, or 100 seconds. We adopt the asynchronous method, FedAsync [213], where client updates are aggregated with decay factors based on latency. Let $W^{(t)}$ denote the global weights at t-th round before aggregation, and $W^{(t_+)}$ the global weights after aggregation. Similarly, let $W_m^{(t)}$ represent the model weights of client m at t-th round. The aggregation follows:

$$\beta_t = (t-\tau)^{-a} \cdot \beta, \tag{5.2}$$

$$W^{(t_+)} = (1-\beta_t)W^{(t)} + \beta_t W_m^{(t)} \tag{5.3}$$

where β and a are hyperparameters set to $\beta = 0.6$ and $a = 0.5$ as reported in FedAsync. To create non-IID data, each client is assigned a non-overlapping half of the MNIST classes.

The global performance is shown in Figure 5.2a, where markers represent updates from the "slow" client with longer latency. We observe an increase in accuracy when the server

aggregates updates from the slow client, as these updates introduce knowledge of previously unseen classes. However, this gain is gradually lost, with accuracy declining to around 0.5 after several updates from the faster client. This suggests the fast client's updates override the contributions of the slower client. Moreover, as the latency of the slower client increases, the decay factor β_t for integrating its updates decreases. This weakens its contribution to the global model and slows convergence, especially under non-IID data, as valuable knowledge from the slower client is not fully utilized.

Figure 5.2b visualizes changes in global model weights in the final hidden layer before the classifier in the case where the latency of the two clients is 10 and 100 seconds respectively. The y-axis represents neurons, and the x-axis represents the number of updates. The color indicates the direction and degree of global weight changes, with red representing an increase and blue a decrease. We observe abrupt shifts occur when switching between clients. Updates from the slow client often decrease the neuron weights (blue), while subsequent updates from the fast client increase the weight values (red), pulling the model in opposite directions. The antagonistic behavior is due to objective inconsistency—while the global model optimizes for the overall distribution, client updates follow distinct local objectives, driving oscillations in weight aggregation.

5.3 Methodology

In this section, we introduce our ORTHOFL algorithm, present the mathematical intuition for orthogonalization, and visualize an example of the optimization trajectories for illustration.

5.3.1 ORTHOFL Algorithm

Once receiving a client update, ORTHOFL immediately integrates it into the global model. ORTHOFL maintains *separate* variables for global and client models. Before merging global weight shift to the client model, the server orthogonalizes the global shift against the received client update. This orthogonality allows the client to incorporate global progress while continuing

Algorithm 4: Pseudo-code of ORTHOFL Algorithm

Input : Total number of updates T, local training epochs E, initial global model weights $W^{(0)}$.

Output : Global model weights $W^{(T_+)}$.

Server execution:

Send $W^{(0)}$ to all available clients;

for $t = 1, \ldots, T$ **do**

  Receive update from a client;

  Calculate delay $t - \tau$;

  Compute global weight shift: $\Delta W = W^{(t)} - W^{(\tau_+)}$;

  Compute client weight change: $\Delta W_m = W_m^{(t)} - W_m^{(\tau_+)}$;

  Orthogonalize each layer: $\Delta W^{l\perp} = \Delta W^l - \frac{\Delta W^l \cdot \Delta W_m^l}{\Delta W_m^l \cdot \Delta W_m^l} \Delta W_m^l$;

  Merge weights for client: $W_m^{(t_+)} = W_m^{(t)} + \Delta W^{\perp}$;

  Update global model: $W^{(t_+)} = (1 - \beta_t) W^{(t)} + \beta_t W_m^{(t)}$;

  ClientUpdate(m, $W_m^{(t_+)}$)

return $W^{(T_+)}$;

ClientUpdate(m, $\tilde{W}$):

Initialize local model: $W_m \leftarrow \tilde{W}$;

for $e = 1, \ldots, E$ **do**

  Partition $\mathbf{D}_m$ into mini-batches $\bigcup_{i=1}^{j_m} B_i^{(m)}$;

  for $i = 1, \ldots, j_m$ **do**

    Update local weights: $W_m \leftarrow W_m - \eta_c \nabla_{W_m} \mathscr{L}_m(W_m; B_i^{(m)})$;

return W_m to server;

its local optimization without disruption. The pseudo-code is presented in Algorithm 4.

Global Aggregation via Moving Average. Denote $W^{(t)}$ as the global model weights at the t-th round before client update and $W^{(t_+)}$ after update. Similarly, let $W_m^{(t)}$ be the client m_t's local model weights at the t-th round before update and $W_m^{(t_+)}$ after update. Note that $W^{(t+1)} := W^{(t_+)}$ as the global model weights stay unchanged after communication with a client before the next client update. We update the global model with a moving average:

$$W^{(t_+)} = (1 - \beta_t) W^{(t)} + \beta_t W_m^{(t)}, \tag{5.4}$$

where β_t controls the contribution of client m's update to the global model. We let $\beta_t :=$

$s_a(t-\tau)\cdot\beta$ with $\beta \in (0,1)$ and $s_a(x) = x^{-a}$ for some $a > 0$ so that update with a larger staleness has a smaller contribution to global weight update, and thereby decreasing the influence of client update with long delay.

Calibration on Client Updates. To minimize interference caused by asynchronous updates, we orthogonalize the global weight change that occurs between the client's successive updates against client's local udpate before sending it to the client for the next round of training. Formally, when the server receives an update from client m, if the staleness $t-\tau > 1$, it calculates the local weight change from its last update to its current update:

$$\Delta W_m = W_m^{(t)} - W_m^{(\tau_+)}. \tag{5.5}$$

Similarly, the server calculates the global weight shift due to aggregating updates from other clients during this period:

$$\Delta W = W^{(t)} - W^{(\tau_+)}. \tag{5.6}$$

To update client m with global progress, the server computes the orthogonal component of ΔW with respect to ΔW_m. The orthogonalization is done for the weight of each layer through removing the component that is parallel to ΔW_m. Let ΔW^l and ΔW_m^l be the change in the layer l of the global weights and the local weights respectively. The component of ΔW^l orthogonal to ΔW_m^l is:

$$\Delta W^{l\perp} = \Delta W^l - \text{proj}_{\Delta W_m^l}(\Delta W^l) = \Delta W^l - \frac{\Delta W^l \cdot \Delta W_m^l}{\Delta W_m^l \cdot \Delta W_m^l}\Delta W_m^l. \tag{5.7}$$

Let $\Delta W^{\perp}$ represent the aggregation of $\Delta W^{l\perp}$ across all layers. This orthogonal component $\Delta W^{\perp}$ ensures that the updates from the other clients during delay $t-\tau$ do not interfere with the client's local progress, as it removes any component of the global weight change during delay along the direction of the local update. $\Delta W^{\perp}$ is then sent to current client m to form the new

client model weight for the subsequent round of local training on client m:

$$W_m^{(t_+)} = W_m^{(t)} + \Delta W^{\perp}. \tag{5.8}$$

5.4 Analysis

5.4.1 Mathematical Basis of Orthogonalization

A gradient update orthogonal to previously accumulated gradients helps preserve existing model behavior and minimizes unintended changes to its outputs [42]. Due to the high-dimensional parameter space of neural networks, there are multiple directions that are orthogonal to the stale local weight update. We have the following lemma showing that the orthogonalization strategy in ORTHOFL preserves the maximal information from the global weight shift vectors perpendicular to the local update direction.

For $v, w \in \mathbb{R}^d$, let $\langle v, w \rangle := \sum_{i=1}^{d} v_i w_i$ be the standard inner product on $\mathbb{R}^d$.

Lemma 5.4.1. *Let $v \in \mathbb{R}^d$ and $\mathscr{U} = \{u_1, \cdots, u_k\}$ be an orthonormal set for some $k < d$. Then for any $w \in (\operatorname{span} \mathscr{U})^{\perp}$,*

$$\|v - v^{\perp}\| \leq \|v - w\|, \tag{5.9}$$

where $v^{\perp} := v - \sum_{i=1}^{k} \langle v, u_i \rangle u_i$ denote the component of v orthogonal to $\mathscr{U}$. Moreover, the angle between v and $v^{\perp}$ is less than any angle between v and w for $w \in (\operatorname{span} \mathscr{U})^{\perp}$.

The proof is given in Appendix. We apply the above lemma in our case when $v = \Delta W^l$ and $\mathscr{U} = \{\Delta W_m^l\}$. It implies $\Delta W^{l\perp}$ in equation (5.7) is the unique vector among those perpendicular to ΔW_m^l with the smallest magnitude of $\|\Delta W^l - \Delta W^{l\perp}\|$ and the smallest angle with ΔW^l. This allows the server to pass the maximum knowledge from other clients to client m without interfering with its most recent learning progress.

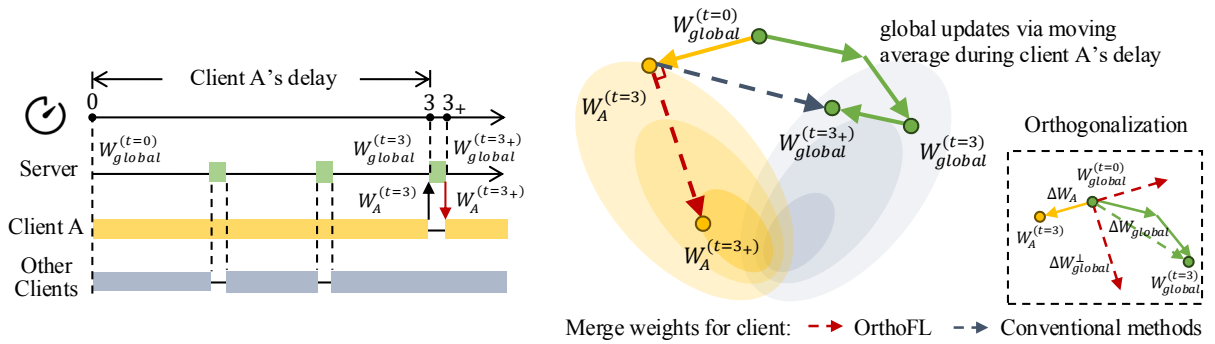


Figure 5.3. Optimization trajectories. Shaded regions represent iso-loss contours for client A (yellow) and other clients (gray). Deeper colors indicate lower loss. ORTHOFL removes conflicting components via orthogonalization, merging updates with minimal interference.

5.4.2 Visualization of Optimization Trajectories

We illustrate the advantage of our method by visualizing an example of optimization trajectories. As shown in Figure 5.3, client A begins local training from the global state $W_{\text{global}}^{(t=0)}$. Before A's update arrives, the server aggregates two updates from other clients into the global model. Consequently, just before aggregating A's update at $t = 3$, the global model has evolved to $W_{\text{global}}^{(t=3)}$. Meanwhile, client A finishes local training and submits the updated parameters $W_A^{(t=3)}$. The shaded regions show the iso-loss contours for client A (yellow) and the collective optimization space of other clients (gray). Deeper colors indicate lower loss areas.

The global weight is updated via a moving average and becomes $W_{\text{global}}^{(t=3_+)}$. In conventional asynchronous methods, this global weight is directly assigned to client A (blue dashed line). This would push the model farther from A's optimization objective than $W_A^{(t=3)}$, reversing A's learning progress. ORTHOFL mitigates this by removing the component of the global weight shift that is parallel to ΔW_A. This ensures that the calibrated parameters are orthogonal to A's update direction. Finally, the calibrated global shift is merged into A's model, which becomes $W_A^{(t=3+)}$. This way, ORTHOFL reduces interference due to staleness and objective inconsistencies while preserving meaningful contributions at both global and local levels.

5.5 Experiments

We evaluate ORTHOFL in terms of accuracy and speed. We also examine its sensitivity to hyperparameters and robustness under varying delay patterns and data heterogeneity.

5.5.1 Experiment Setup

Compared Methods. We consider the following baselines:

- **FedAvg** [138]: The classical synchronous algorithm where the server selects a subset of clients to conduct training in each round and synchronizes updates from these clients before aggregation.
- **FedProx** [111]: A synchronous method that addresses data heterogeneity by incorporating L2 regularization during local training to constrain the divergence between global and client models.
- **FedAdam** [160]: A synchronous algorithm that integrates Adam optimizer for the server. It adapts learning rates for each parameter using first- and second-moment estimates, improving convergence and accelerating training under data heterogeneity.
- **FedAsync** [213]: A fully-asynchronous method that lets the server immediately aggregate the client updates into the global model upon receipt. It uses a weighting mechanism as in Equation 5.3 to account for the staleness of the updates.
- **FedBuff** [146]: A semi-asynchronous method that introduces a buffered aggregation strategy. It maintains a buffer to collect client updates. Once the buffer is full, the server aggregates the client updates in the buffer and updates the global model.
- **CA^{2}FL** [204]: Another semi-asynchronous method based on buffered aggregation. It caches the latest update from every client on the server and uses them to estimate the clients' contribution to the update of the current round and calibrate global updates.

Datasets and Models. We conduct experiments on five datasets spanning three applications: image classification, text classification, and human activity recognition. To assess the robustness of our method across different neural network architectures, we pair each dataset with a model suited to its data type. Table 5.1 summarizes the setups. The details are as follows:

1. **Image Classification.** We evaluate our method on three widely used image datasets: MNIST [31], CIFAR-10, and CIFAR-100 [90]. For MNIST, we use LeNet5 [95], a lightweight convolutional network. For CIFAR-10, we adopt VGG11 [176], a deeper convolutional architecture. For CIFAR-100, we employ MobileNetV2 [164], a compact and efficient model ideal for large-scale image classification tasks.

2. **Text Classification.** We experiment with the 20 Newsgroups dataset [94], a benchmark dataset for multi-class text categorization. We adopt DistillBERT [165], a small transformer model suitable for resource-constrained devices. To evaluate ORTHOFL's performance in parameter-efficient fine-tuning (PEFT) settings [60], we employ a pretrained DistilBERT [165] from Hugging Face[1] for evaluations with the 20 Newsgroups dataset and fine-tune it using Low-Rank Adaptation (LoRA) [65], which reduces the number of trainable parameters.

3. **Human Activity Recognition.** We use the HAR [6] dataset, which contains time-series sensor data for different physical activities. We adopt the 1D version of ResNet18 [61], a modified ResNet architecture for processing 1D sequential data.

Data Heterogeneity. For the HAR dataset, clients are naturally divided based on the individual subjects, as each subject represents a distinct client. For the other datasets, we set the number of clients equal to the number of classes in each dataset. To create non-IID client distributions, we follow prior work [64] and use a Dirichlet distribution $Dir(\alpha = 0.1)$ to derive class distribution.

Delay Simulation. To ensure controlled evaluation and avoid variability in federated learning deployments, we simulate client delays using measurements (including communication

[1] https://huggingface.co/distilbert/distilbert-base-uncased

Table 5.1. Datasets and models in the experiments. The datasets cover 3 different applications, and we evaluate both full-weight training and parameter-efficient fine-tuning (PEFT).

Datasets	Clients	Avg. $\lvert\mathbf{D}_m\rvert$	Model	PEFT?	Data Type
CIFAR-10	10	4000	VGG11	✗	image
MNIST	10	6000	LeNet5	✗	image
20 Newsgroups	20	566	DistilBERT	✓	text
HAR	21	350	ResNet18	✗	time-series
CIFAR-100	100	400	MobileNetV2	✗	image

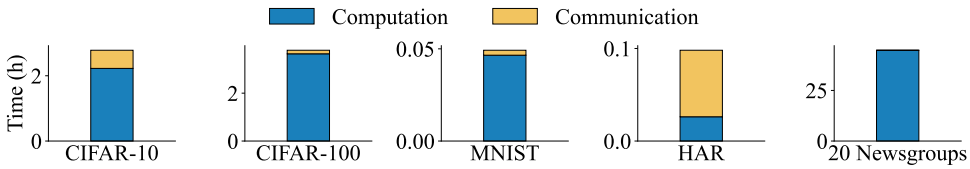


Figure 5.4. Average latency per round across clients. For 20 Newsgroups, the converted computational time is sufficiently long, making communication time negligible.

and computational latency) from prior work [222], which were collected using Raspberry Pi (RPi) devices in different home environments with wireless connectivity. To account for differences in the size of models and datasets, we use an RPi 4B which has comparable computational capabilities as the reported ones to measure latency for training one round under each model and dataset configuration. Each configuration is tested five times to derive the average training time. The computational latency for each dataset is then adjusted based on the ratio between our measured time and the computation latency in [222]. Similarly, communication latency was scaled based on the model size in bytes compared to the model used in the original delay collection. Figure 5.4 presents the average communication and computation latency on Raspberry Pis scaled for the dataset and model configuration in our experiments. Using these measurements, we calculate the mean and variance of latency for each device. During simulations, we assign the statistics to each client randomly and sample latency for each round from a Gaussian distribution parameterized by the assigned statistics. For fair comparisons, the order of client updates is kept consistent with a fixed set of random seeds. Besides these real-world measurements, we also investigate model performance under additional delay distributions in Section 5.5.5.

Table 5.2. Summary of main experiment results including average accuracy, standard deviation, and time relative to FedAvg. ORTHOFL reaches the target accuracy more quickly and achieves better final accuracy.

Methods	MNIST		CIFAR-10		20 Newsgroups		HAR		CIFAR-100	
	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time
FedAvg [138]	0.9215±0.0107	1×	0.7376±0.0235	1×	0.5810±0.0313	1×	0.8422±0.0061	1×	0.2216±0.0035	1×
FedProx [111]	0.9052±0.0092	1.09×	0.7333±0.0217	0.90×	0.5855±0.0305	0.98×	0.8401±0.0094	0.98×	0.2040±0.0055	1.18×
FedAdam [160]	0.9379±0.0234	0.91×	0.7457±0.0147	0.78×	0.5894±0.0176	1.08×	0.8707±0.0333	0.68×	0.4211±0.0100	0.36×
FedAsync [213]	0.9536±0.0070	0.39×	0.7426±0.0108	0.48×	0.6183±0.0376	0.27×	0.8764±0.0190	0.26×	0.4791±0.0088	0.20×
FedBuff [146]	0.9359±0.0211	0.46×	0.7356±0.0322	0.55×	0.6206±0.0204	0.34×	0.8741±0.0104	0.30×	0.6227±0.0045	0.12×
CA^{2}FL [204]	0.9611±0.0141	0.30×	0.6973±0.0804	0.60×	0.6567±0.0158	0.24×	0.8784±0.0192	0.29×	0.6112±0.0071	0.09×
OrthoFL (ours)	**0.9818±0.0029**	**0.18×**	**0.7653±0.0326**	**0.19×**	**0.6615±0.0126**	**0.09×**	**0.8974±0.0101**	**0.23×**	**0.6297±0.0056**	**0.03×**

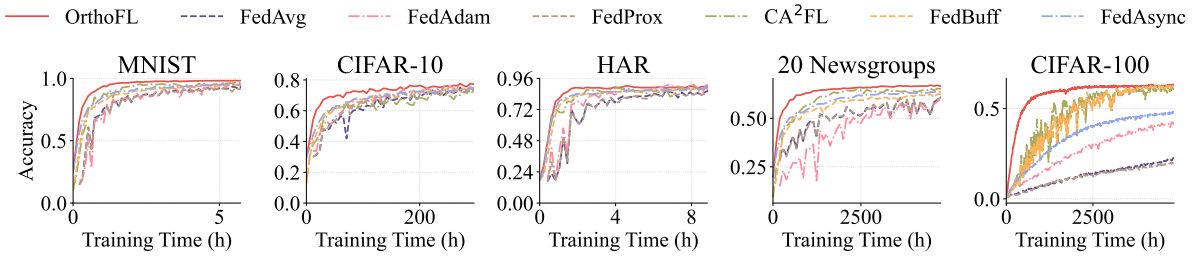


Figure 5.5. Accuracy w.r.t. training time.

Federated Learning Configuration. The number of local training epochs $E = 5$. For the FedAvg algorithm, the number of sampled clients at each round is 10. The aggregation hyperparameters in Equation 5.4 are set to $\beta = 0.6$ and $a = 0.5$, following the values used in prior work [213]. The learning rate for local training at all clients is 5×10^{-5} for the 20 Newsgroups dataset and 0.01 for the other datasets.

Evaluation Metrics. We report accuracy after training for sufficient clock cycles to ensure the method reaches stable performance. For a fair comparison, we fix the same training time across all methods. In addition, we compare the time spent in reaching a target accuracy—set as the 95% of the lowest final accuracy among all compared methods. FedAvg serves as the baseline of time consumption (i.e., 1×), and we report the relative time for other methods.

5.5.2 Main Experiment Results

Table 5.2 summarizes the results. ORTHOFL consistently outperforms the compared methods across all datasets—it not only converges faster but also achieves better final accuracy.

Table 5.3. Aggregation time of ORTHOFL.

Dataset	Model	Train Params	Time
MNIST	LeNet5	44K	0.003s
HAR	ResNet18	119K	0.199s
20 Newsgroups	DistilBERT	753K	0.110s
CIFAR-10	VGG11	9.2M	0.188s
CIFAR-100	MobileNetV2	2.4M	0.512s

Notably, the advantage becomes increasingly obvious in the setting with a larger number of clients, as seen in CIFAR-100. This is because, for synchronous methods, the client sampling rate decreases as the number of clients increases (e.g., only 10 out of 100 clients are sampled in each round), leading to longer wait times and slower convergence. Similarly, for baseline asynchronous methods, although their aggregation mechanisms are designed to mitigate the effects of model staleness, they fail to effectively address the challenges posed by data heterogeneity. In contrast, the orthogonal calibration mechanism in ORTHOFL mitigates both the impact of stale model updates and the model divergence caused by data heterogeneity, ensuring faster convergence and improved performance.

Figure 5.5 presents the accuracy over training time. ORTHOFL demonstrates reduced fluctuations in accuracy over time. These fluctuations, observed in baseline methods, are caused by conflicting updates from clients with highly divergent data distributions.

5.5.3 Overhead Analysis

Compared to baseline methods, ORTHOFL does not introduce any additional communication overhead. The extra computational overhead occurs on the server during the orthogonalization process.

The overhead of orthogonal aggregation depends on the model size, as orthogonalization is performed through matrix operations on the weight changes of each layer. As shown in Table 5.3, for the model configurations used in our experiments, the orthogonalization operation adds between 3 ms (for MNIST) and 512 ms (for CIFAR-100) when running aggregation on a

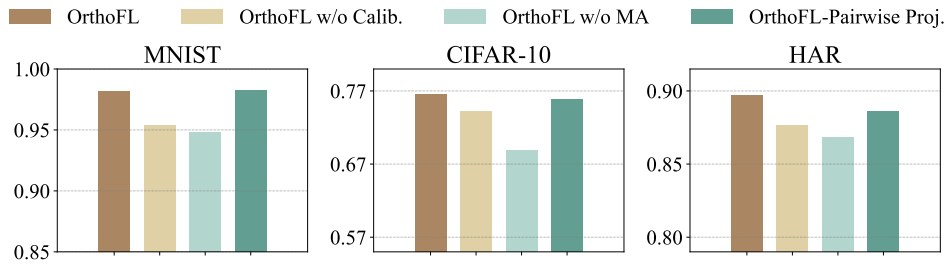


Figure 5.6. Ablation studies. ORTHOFL w/o Calib. corresponds to FedAsync, as it represents an ablation without calibration.

server equipped with an AMD EPYC 7713 64-Core Processor (3.72 GHz max clock) and 3.9 TiB RAM. This additional computation time for weight aggregation is negligible compared to the computational and communication latency on clients, as illustrated in Figure 5.4.

5.5.4 Ablation Studies

We conduct ablation studies to evaluate our key design choices. First, we set FedAsync as a baseline, denoted as ORTHOFL *w/o Calib.*, since it can be viewed as our ablation without client calibration. Second, we assess the role of global aggregation by removing the moving average and directly loading the calibrated client weight into the global model, denoted as ORTHOFL *w/o MA*. Furthermore, we investigate an alternative orthogonalization strategy which projects the incoming client update onto the orthogonal subspace of the most recent updates from all other clients, denoted as ORTHOFL-*Pairwise Proj.*. The orthogonality is achieved through the same process (i.e., removing parallel components) as in ORTHOFL.

We observe performance decreases when clients and the global model share the same weights (ORTHOFL w/o Calib. and ORTHOFL w/o MA). ORTHOFL w/o MA performs particularly poorly as the absence of a global moving average causes overfitting to individual client distributions. These results emphasize the need to decouple global and local learning to address objective inconsistencies. Furthermore, ORTHOFL-pairwise Proj. achieves performance comparable to ORTHOFL, suggesting that effective orthogonal calibration can be realized through multiple viable approaches. We expect that fine-tuning the orthogonalization strategy could

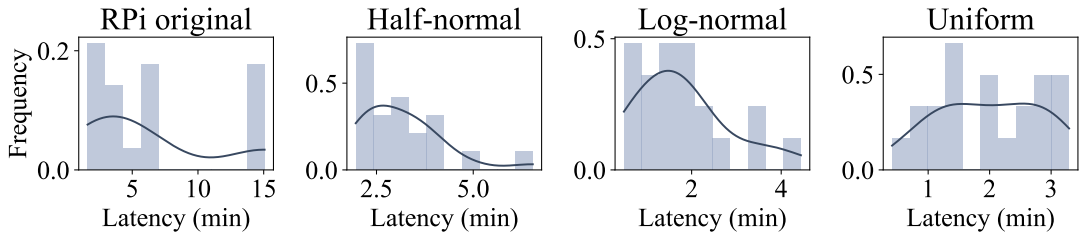


Figure 5.7. Delay patterns under different distributions.

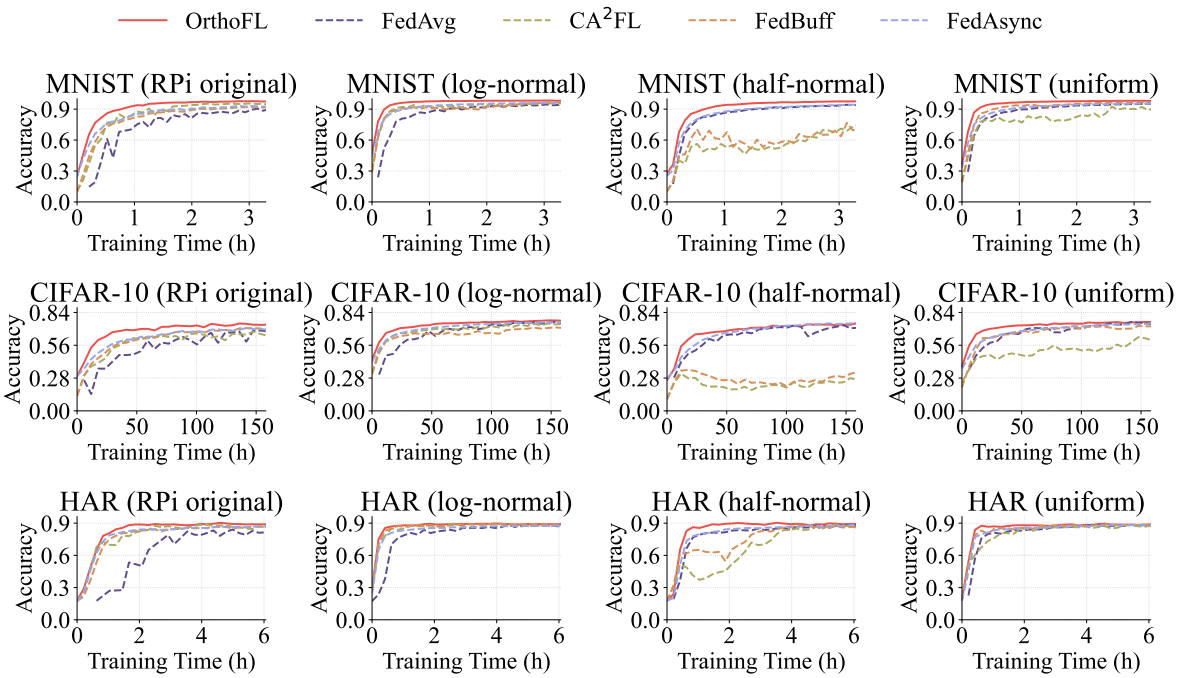


Figure 5.8. Performance with different delay distributions.

further improve the performance. We leave the exploration for future work.

5.5.5 Exploratory Studies

How do algorithms perform under different delay distributions? Since real-world deployment of federated learning may present diverse delay patterns, we explore other possible delay distributions in real-world setups, such as following log-normal, half-normal [180] and uniform distributions [146]. The derivations of these delay distributions are as follows:

- **Lognormal distribution**: The parameters μ (mean) and σ (standard deviation) of the natural logarithm of delays are derived from the measurements on Raspberry Pis. Specifically, μ_R and

σ_R represent the arithmetic mean and standard deviation of the measured delays for all rounds, respectively.

$$\sigma = \sqrt{\ln\left(\frac{\sigma_R^2}{\mu_R^2} + 1\right)}, \quad \mu = \ln(\mu_R) - \frac{\sigma^2}{2}$$

Then, the latency for each client is sampled from the lognormal distribution to capture skewed and heavy-tailed delays.

- **Half-normal distribution**: The mean and standard deviation of the delays (i.e., μ_R and σ_R) are calculated from the delay measurements on Raspberry Pis. Then, for each client, its latency is sampled from the half-normal distribution, ensuring non-negative values and a skewed distribution toward smaller delays.

- **Uniform distribution**: Client latency is sampled from a uniform distribution with bounds set between the 5th and 95th percentiles of the measurements from Raspberry Pis. This ensures outliers are excluded while covering the majority of the observed range.

Figure 5.7 shows the simulated latency for 100 clients, each running CIFAR-100 with MobileNetV2 over 10,000 rounds. The real-world measured RPi distribution exhibits discrete peaks and high variability. The half-normal distribution has a long tail and a peak at low latency. The lognormal distribution has a heavier tail and its concentration is closer to zero, reflecting occasional large latency. The uniform distribution assumes equal probability within a bounded range. Figure 5.8 presents the accuracy curves for each algorithm under different delay distributions. With real-world measured RPi latency, some clients experience substantially longer latencies, causing synchronous methods like FedAvg to converge more slowly as the server waits for stragglers. In this case, asynchronous methods generally have better performance than synchronous ones. Under the lognormal, half-normal, and uniform delay distributions, extreme latencies are less common, so the performance gap between synchronous and asynchronous methods narrows. However, the two buffer-based semi-asynchronous methods are sensitive to the delay patterns as they show lower performance under half-normal and uniform latency. In

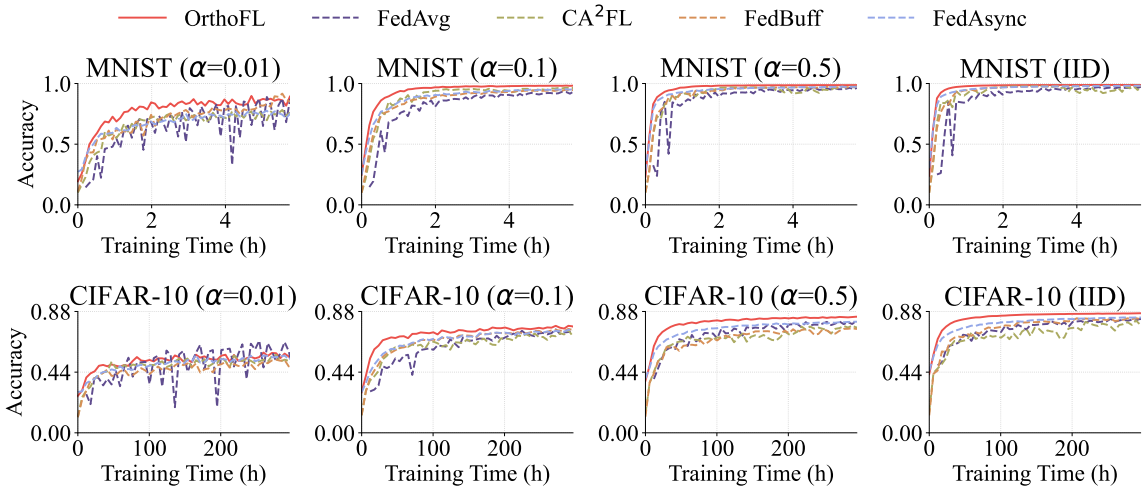


Figure 5.9. Performance under varying data heterogeneity.

general, OrthoFL performs the best across all scenarios, demonstrating its robustness against different delay patterns.

How does data heterogeneity impact performance? To control the level of data heterogeneity, we change α for Dirichlet distribution from $\{0.01, 0.1, 0.5, 10^4\}$, where $\alpha = 10^4$ simulates the IID case. We present experiments on MNIST and CIFAR-10 as shown in Figure 5.9. As the client data distribution becomes more heterogeneous (i.e., lower values of α), we observe the performance of baseline methods has more fluctuations and decreases in final accuracy. This is because client models trained on non-IID data distributions have larger divergences in their weights. Aggregating divergent updates amplifies inconsistencies, leading to slower convergence and lower accuracy for baseline methods. By contrast, OrthoFL exhibit stable performance across all settings. In the IID case, where data is uniformly distributed across clients, OrthoFL still outperforms the compared methods. This can be attributed to OrthoFL's ability to address model staleness through orthogonal calibration.

5.5.6 Sensitivity Analyses

Aggregation Hyperparameter. The parameter β in Equation 5.4 acts as a smoothing factor that balances the contribution of client updates to the global model and the retention

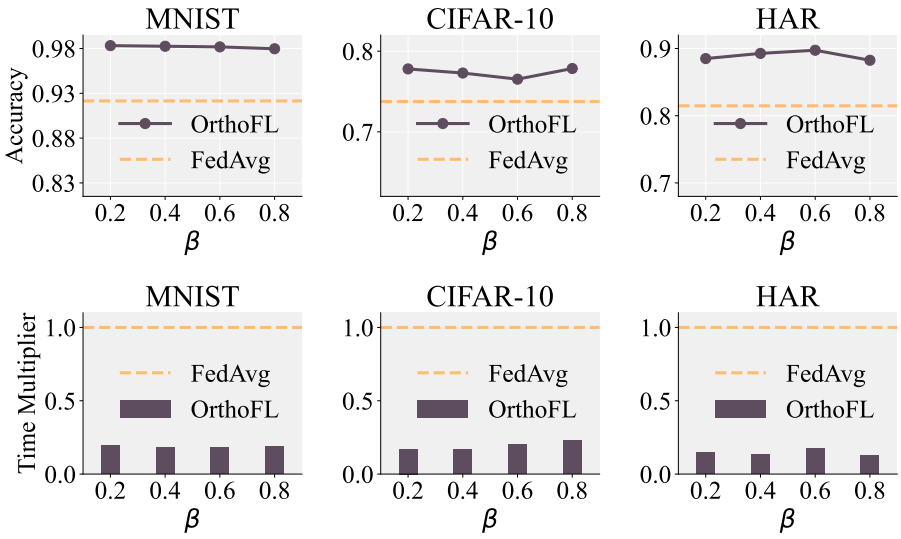


Figure 5.10. Sensitivity analysis of aggregation hyperparameter β. OrthoFL is robust to β, maintaining higher accuracy than FedAvg and a low relative time.

of the global model's weights from the previous round. A larger β allows the global model to incorporate more of the client updates, potentially accelerating learning, while a smaller β preserves more of the global model's previous state, enhancing stability. We vary the value of β from $\{0.2, 0.4, 0.6, 0.8\}$. As shown in Figure 5.10, OrthoFL is robust to different β values and always achieves higher accuracy than FedAvg after a fixed training time, and the relative time of OrthoFL is low.

Number of Local Training Epochs. The number of local training epochs E determines how much information is learned during a round and impacts the overall convergence speed. A larger E allows clients to learn more information from their local data, potentially improving local model performance. However, it has the risk of larger divergence among client models and global models. On the other hand, setting a smaller E ensures closer alignment between client and global models but comes at the cost of higher communication latency due to more frequent synchronization. We vary the value of E from $\{1, 5, 10\}$ and show the results in Figure 5.11. The upper row shows the final accuracy after a fixed training time and the bottom row presents the relative time to reach 95% of FedAvg's target accuracy when $E = 5$. We observe that when $E = 1$, both OrthoFL and FedAvg reach lower accuracy compared to larger E. This is due

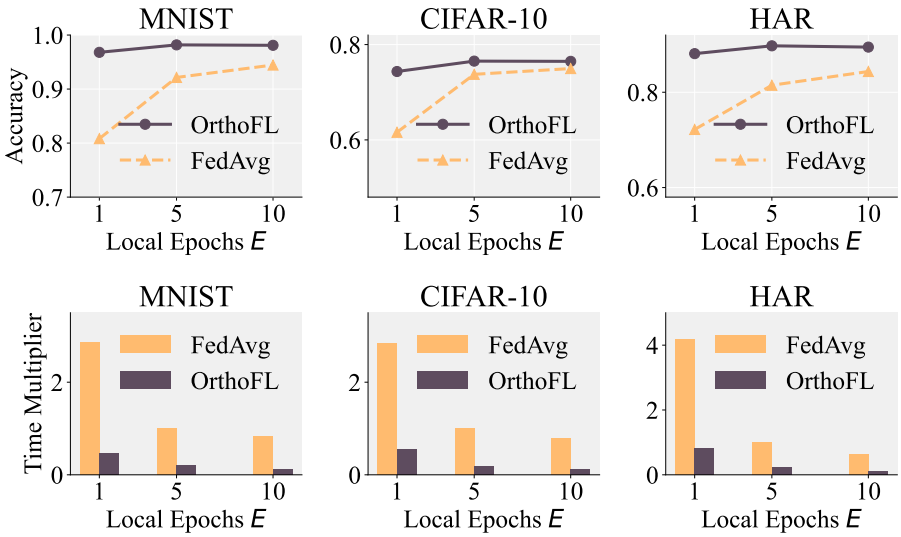


Figure 5.11. Sensitivity analysis of the number of local training epochs E. An appropriately chosen E improves accuracy within the same training duration and expedites convergence.

to insufficient local learning, requiring more communication rounds to achieve comparable performance. ORTHOFL achieves similar final accuracy when $E = 5$ and $E = 10$. Notably, ORTHOFL outperforms FedAvg across different E values. The speedup of ORTHOFL is more obvious at smaller E values (e.g., $E = 1$).

5.6 Related Works

Federated Learning and Heterogeneity Problem. Federated learning [138] is a distributed learning paradigm that allows multiple parties to jointly train machine learning models without data sharing, preserving data privacy. Despite the potential, it faces significant challenges due to heterogeneity among participating clients, which is typically classified into two main categories: data heterogeneity and system heterogeneity. Data heterogeneity appears as clients own non-IID (independent and identically distributed) data [111, 80, 199, 236]. The difference in data distribution causes the local updates to deviate from the global objective, making the aggregation of these models drift from the global optimum and deteriorating convergence. System heterogeneity refers to variations in client device capabilities, such as computational power, network bandwidth, and availability [199, 237, 103, 39, 4, 231]. These disparities lead to uneven

progress among clients, and the overall training process is delayed by slow devices. Traditional federated learning approaches rely on synchronization for weight aggregation [138, 111, 160], where the server waits for all clients selected in a round to complete and return model updates before proceeding with aggregation. This synchronization leads to inefficient resource utilization and extended training times, particularly in large-scale deployments involving hundreds or thousands of clients. Addressing the heterogeneity issues is a critical problem for improving the scalability and efficiency of federated learning systems in real-world deployment.

Asynchronous Federated Learning. Much of the asynchronous federated learning literature focuses on staleness management by assigning weights for aggregating updates according to factors including staleness [213], divergence from the global model [179, 226] and local losses [125]. For example, [213] lets the server immediately aggregate client updates into the global model with a weight determined by staleness. Another line of research caches client updates at the server and reuses them to calibrate global updates [51, 204]. For example, [204] maintains the latest update for every client to estimate their contribution to the current aggregation and calibrate global update. Furthermore, semi-asynchronous methods [146, 226] balance between efficiency and training stability. For example, [146] buffers a fixed number of client updates before aggregation. We select representative methods from each category for our comparisons. Besides, some works improve efficiency from a different perspective—through enhanced parallelization. Methods include decoupling local computation and communication [8] and parallelizing server-client computation [228]. In addition, asynchronous architectures have been explored in other paradigms such as vertical [239] and clustered [123] federated learning. While these directions complement our work, they fall outside the scope of this study.

Asynchronous Stochastic Gradient Descent. Asynchronous stochastic gradient descent (SGD) is closely related to asynchronous federated learning and has provided theoretical and empirical foundations for scalable distributed training. Early studies analyzed error-runtime trade-offs, showing that incorporating stale gradients can alleviate system bottlenecks without significantly compromising accuracy [38]. Subsequent work refined convergence bounds based

on maximum [177] or average [87] delay and demonstrated that asynchronous SGD can converge faster than traditional minibatch SGD [141]. To tackle challenges such as gradient staleness, communication delays, and convergence guarantees, various strategies have been proposed, such as filtering out outlier gradients [214, 29], adjusting update steps according to delay [142, 9], and approximating gradients to compensate for delayed information [247]. However, unlike federated learning, most asynchronous SGD formulations do not explicitly address non-i.i.d. data distributions or the strict data privacy constraints inherent in federated settings, which limits their direct applicability.

5.7 Summary

We introduce ORTHOFL, an orthogonal calibration mechanism for asynchronous federated learning. ORTHOFL exploits the high-dimensional parameter space of neural networks and projects the global weight shift during a client's delay onto a subspace orthogonal to its stale update. This projection ensures global progress is integrated into client models without disrupting local learning. Experiments demonstrate that ORTHOFL consistently outperforms state-of-the-art synchronous and asynchronous baselines, achieving notable gains in accuracy, convergence speed, and robustness under diverse delay patterns and data heterogeneity.

Chapter 5 incorporates material from the publication "Orthogonal Calibration for Asynchronous Federated Learning", by Jiayun Zhang, Shuheng Li, Haiyu Huang, Xiaofan Yu, Rajesh K. Gupta, Jingbo Shang, published as Preprint arXiv:2502.15940, 2025. The dissertation author was the primary investigator and the lead author of this paper.

Part III

Deriving Auxiliary Knowledge for Bridging Distributed Domains

Chapter 6

Contextual Inference Under Minimal Supervision

In Part I and Part II, contextual knowledge, such as domain semantics, temporal signals, prior knowledge about class distributions, are used as a crucial input to the learning algorithms for understanding correlations among domains (e.g., clients, users) in distributed environments. However, in practice, such information is often sparsely collected, limiting its direct applicability. In this chapter, we address this challenge by introducing methods to infer contextual information from sparsely-labeled data. By enriching the available context, our approach enhances the quality of training in heterogeneous distributed systems. Specifically, we explore demographic inference from mobility data as a case study, since mobility data is commonly seen in many mobile applications.

The context about trips and users from mobility data is valuable for mobile service providers to understand their customers and improve their services. Existing inference methods require a large number of labels for training, which is hard to meet in practice. In this work, we study a more practical yet challenging setting—contextual inference using mobility data with minimal supervision (i.e., a few labels per class and massive unlabeled data). A typical solution is to apply semi-supervised methods that follow a self-training way to bootstrap a model based on all features. However, using a limited labeled set brings a high risk of overfitting to self-training, leading to unsatisfactory performance. We propose a novel collaborative distillation method

STCOLAB. It sequentially trains spatial and temporal modules at each iteration following the supervision of ground-truth labels. In addition, it distills knowledge to the module being trained using the logits produced by the latest trained module of the other modality, thereby mutually calibrating the two modules and combining the knowledge from both modalities. Extensive experiments on two real-world datasets show STCOLAB achieves significantly more accurate contextual inference than various baselines.

6.1 Introduction

The prevalence of location-based mobile services offers new opportunities for businesses to better understand the context of trips (e.g., transportation mode and purpose) and their customers (e.g., ethnicity, disability, and socioeconomic status). Such information can facilitate a wide spectrum of mobile applications, including human mobility recovery [40], urban planning [130], and personalized location recommendation [203]. In practice, given the sensitive nature, very few users would share the contextual information, especially at the beginning of the business [167, 156, 173]. As such, we study the problem of contextual inference from mobility data *under minimal supervision*, which is an extreme case of semi-supervised learning when using a few labels (e.g., 10) per class. Specifically, we study the inference of people's demographic attributes.

Related work on demographics inference from human mobility [202, 249, 215, 196] requires a large number (e.g., tens of thousands) of users to share labels for training and is prone to overfitting in the minimally-supervised setting. To mitigate the label scarcity problem, existing semi-supervised methods [23, 98, 114] typically follow a self-training approach that bootstraps a single model using all features at once. However, for mobility data, simple concatenation of spatial and temporal features does not always guarantee improvement in predictions. Due to different network sizes for different modalities, the model inference might lean heavily on one of the modalities [45]. Especially when training data is limited, the model does not have enough

supervision to find the optimal combination of different modalities, leading to unsatisfactory performance and poor generalization.

To better learn with limited supervision, we propose to alternatingly train two separate modules—one for spatial and one for temporal information—and then let them iteratively distill knowledge from each other following a novel collaborative distillation method STCOLAB (see Figure 6.1).

Instead of building one large model that fuses the spatial and temporal modules, we separate the training of the two modules to sufficiently learn features of both modalities with supervision and avoid one modality dominating the learning. Both modules in STCOLAB are supervised by the limited labeled data for contextual inference in an alternating manner—the spatial module first learns geographic features from maps describing where a user has visited; the temporal module then utilizes the features extracted by the trained spatial module and learns cyclic temporal patterns.

In addition to training each module using labeled data, we propose a novel collaborative distillation method that combines the knowledge of spatial and temporal modules to improve model generalization in an iterative manner. We use the latest trained spatial and temporal modules as the teacher model to guide the training of the current spatial/temporal module. Specifically, we construct guidance based on the unlabeled data for which the latest trained modules have consistent and confident predictions. We regulate the learning of the module being trained at each iteration by forcing it to approximate the logits produced by the teacher model on such selected unlabeled data. The two modules give complementary supervision from different views to each other and calibrate the predictions. This way, we combine spatial and temporal information to improve model generalization.

We conduct extensive experiments on two real-world mobility datasets collected from two metropolitan cities in different countries: Chicago in the United States and Brasilia in Brazil. We show that with a small number of labeled samples per class (e.g., 10), STCOLAB can infer important demographic attributes about users with reasonable accuracy, significantly

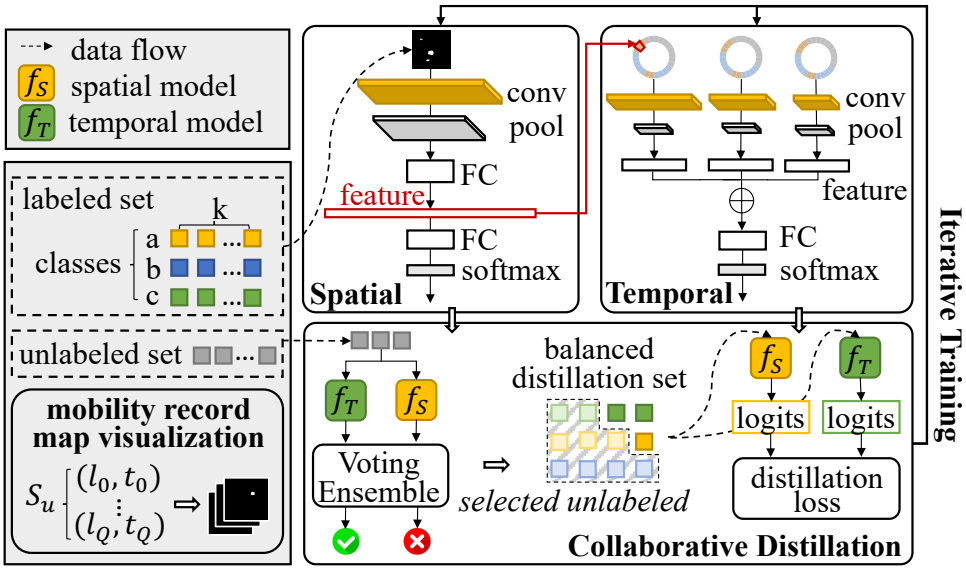


Figure 6.1. Overview of STCOLAB. It consists of two modules: the spatial module takes a map and learns spatial features; the temporal module then utilizes the spatial features and learns temporal patterns in trajectories to make predictions. A collaborative distillation process distills knowledge from the latest trained modules to guide the training of both. The process iterates in self-training cycles.

outperforming the state-of-the-art methods. To the best of our knowledge, we are the first to address the contextual inference problem using mobility data with minimal supervision. We make the following contributions:

- We study the problem of contextual inference from mobility data under the challenging yet practically important minimally supervised setting, where only a few annotated samples are available per class.
- We propose a novel method called STCOLAB, which learns from spatial and temporal modalities iteratively and distills knowledge from both modalities collaboratively to improve generalization using unlabeled data.
- We conduct extensive experiments on two real-world mobility datasets to predict demographic attributes. Results show STCOLAB can predict such information with reasonable accuracy, improving upon state-of-the-art methods.

6.2 Preliminaries

6.2.1 Concepts

Definition 2 (Region)**.** *A region is a geographic unit with a corresponding polygon in the coordinates, denoted as l.*

The two datasets we used in the experiments divide cities into regions according to certain criteria. In the *Chicago* dataset, the city is divided into 866 *census tracts*, each of which is defined for the purpose of taking a census. In the *Brasilia* dataset, the city is divided into 233 *micro zones*, which is defined by the government for statistical purposes. The divisions are shown as grey grids in Figure 6.2a. Note that STCOLAB can also deal with location data of other forms such as fine-grained GPS coordinates. The format of location data is flexible and driven by the available dataset.

Definition 3 (Daily Mobility Record)**.** *An entry of mobility records is a triplet (u,l,t), which denotes user u visits region l during time period t in the day. By sorting the records of user u by time, we get a sequence of time-location pairs:*

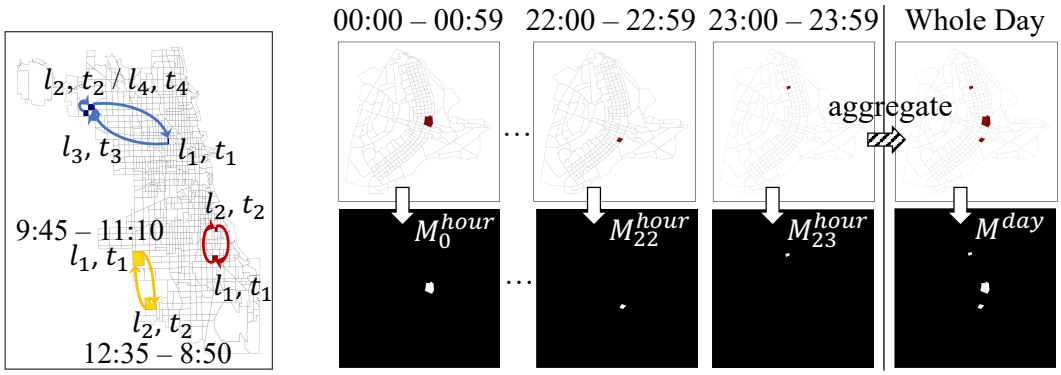
$$\mathscr{S}_u = [(l_0,t_0),(l_1,t_1),\ldots,(l_Q,t_Q)],$$

where Q is the total number of records of user u and l_q and t_q are the region and time period of the q-th record. The time range of each mobility record from a user is one day.

Figure 6.2a shows the daily mobility records of three random users from the Chicago dataset in different colors. For example, the yellow trajectory shows the user stays at l_1 during t_1 (9:45 a.m. - 11:10 a.m.) and stays at l_2 during t_2 (12:35 p.m. - 8:50 a.m.).

6.2.2 Problem Definition

We aim to predict demographic attributes from people's mobile data under minimal supervision. This is an extreme case of semi-supervised learning, where the number of labeled



(a) Mobility Records **(b)** Visualized Maps

Figure 6.2. Examples of mobility records and visualized maps: (a) mobility records of three random people in Chicago; (b) visualized hour maps and day map generated from a person's mobility records.

data is very limited. For each class of a demographic attribute, only a few (e.g. 10) ground truth labels are available.

The number of samples known in each class is denoted as k. The training dataset $\mathscr{D}$ consists of two parts: the labeled set $\mathscr{D}_{lb} = \{(x,y)|x \in \mathscr{X}, y \in \mathscr{C}\}$ and the unlabeled set $\mathscr{D}_{ul} = \{x|x \in \mathscr{X}\}$, where x and y denote the input features and the label of a sample, $\mathscr{X}$ is the set of mobility data and $\mathscr{C}$ is the set of classes. The total number of labeled samples $|\mathscr{D}_{lb}| = k * |\mathscr{C}|$. $|\mathscr{D}_{ul}| \gg |\mathscr{D}_{lb}|$. We aim to learn a model for each prediction task to assign an attribute class label a to each person u given the daily mobility records.

6.3 Methodology

As shown in Figure 6.1, STCOLAB consists of two modules—a spatial module and a temporal module—to learn from different modalities. We design a collaborative distillation process to combine knowledge from both modules in a self-training manner. The pseudo code is presented in Algorithm 5.

Algorithm 5: Iterative Collaborative Distillation

Require : labeled set $\mathscr{D}_{lb}$, unlabeled set $\mathscr{D}_{ul}$

Initialize iteration $t \leftarrow 1$;
while $t \leq MaxIteration$ **do**
  Train spatial model $f_S^{(t)}$ according to Eq. 6.3;
  if $t > 1$ **then**
    Construct distillation dataset $\tilde{\mathscr{D}}_S^{(t)}$ using $f_S^{(t-1)}$ and $f_T^{(t-1)}$ based on voting ensemble;
    Distill knowledge to $f_S^{(t)}$ using $f_T^{(t-1)}$ as teacher according to Eq. 6.2.
  Train temporal model $f_T^{(t)}$ according to Eq. 6.4;
  if $t > 1$ **then**
    Construct distillation dataset $\tilde{\mathscr{D}}_T^{(t)}$ using $f_S^{(t)}$ and $f_T^{(t-1)}$ based on voting ensemble;
    Distill knowledge to $f_T^{(t)}$ using $f_S^{(t)}$ as teacher according to Eq. 6.2.
  $t \leftarrow t+1$;

6.3.1 Iterative Collaborative Distillation

Conventional methods for integrating spatial and temporal information involve fusing features from two sources within one large model, or ensembling the predictions of the spatial and temporal modules. However, training a model with data from spatial and temporal sources in a single pass is likely to bias the model to one modality [45]. To sufficiently learn features from both modalities, we propose an iterative learning process that alternates the training of the spatial and temporal modules in several iterations.

The small training set shows very limited information about the data distribution so the model is prone to overfitting. While labels are hard to collect, the unlabeled data itself provides valuable information for model generalization. Moreover, the spatial and temporal modules learn from two different views of the data. By utilizing the knowledge learned by both modules, they can reduce confirmation bias [7] and mutually calibrate each other. Thus, we design a novel collaborative distillation process to distill knowledge between the two modules by using unlabeled samples.

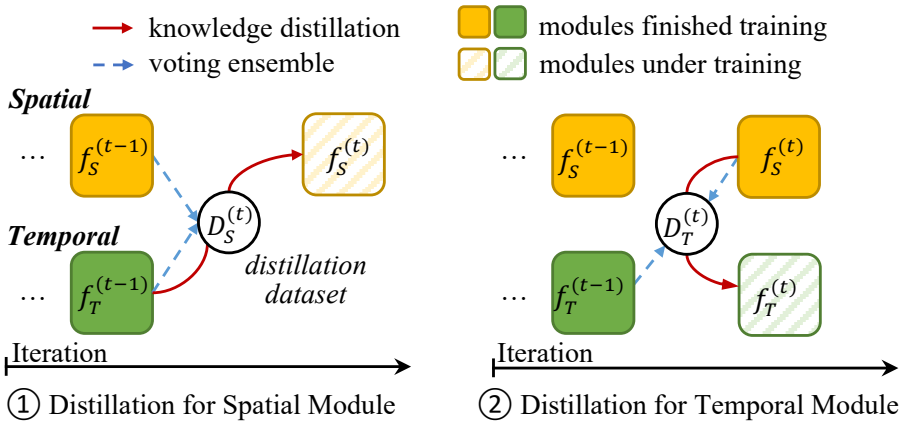


Figure 6.3. Collaborative distillation in a self-training cycle. At iteration t ($t \geq 2$), ① for spatial module $f_S^{(t)}$, we use $f_S^{(t-1)}$ and $f_T^{(t-1)}$ to form distillation dataset via voting ensemble and use $f_T^{(t-1)}$ as teacher for distillation. Then, ② for temporal module $f_T^{(t)}$, we use $f_T^{(t-1)}$ and $f_S^{(t)}$ for voting ensemble and use $f_S^{(t)}$ as teacher.

In the conventional form of knowledge distillation, knowledge is distilled to the new model by training it to approximate the output of a teacher model on a distillation dataset [62]. However, the model trained on a very small training set is likely to make random predictions on unseen data, which causes knowledge given by the teacher model to be noisy. Iterative learning from such noisy knowledge can act like a negative feedback loop and degrade performance [7]. To increase the chance that the teacher model gives correct predictions, we design a voting ensemble method by evaluating the consistency and confidence of its predictions to select samples.

Figure 6.3 explains the iterative collaborative distillation process. In the first iteration, both modules are trained using only the labeled training data. Starting from the second iteration, in addition to training with ground-truth data, we further use the latest trained models from both modalities to conduct a voting ensemble for constructing a distillation dataset and use the latest trained model as the teacher model for distilling knowledge to the current module. Denote the spatial model and temporal model at iteration t as $f_S^{(t)}$ and $f_T^{(t)}$ respectively. Without loss of generality, we illustrate the process of distillation for the temporal model $f_T^{(t)}$.

Voting Ensemble. There is a higher chance that the modules give a correct prediction if

both modules give the same prediction to one sample, compared to the case when the two modules disagree on the prediction. Thus, we use $f_T^{(t-1)}$ and $f_S^{(t)}$ to make predictions on all unlabeled samples and only select those for which the two modules give the same predictions. In addition, the predicted probability can be regarded as the prediction confidence which indicates how certain the model thinks the prediction is correct. We further use percentile scores and choose a subset of the unlabeled samples whose prediction probabilities given by $f_S^{(t)}$ are above the r-th percentile. The threshold of the prediction confidence of class a is $\mathbb{T}_a^{(t)} = \text{percentile}(f_{S,a}^{(t)}(*), r)$, where $f_{S,a}^{(t)}(*)$ are the prediction confidence of all unlabeled samples with respect to class a given by $f_S^{(t)}$. Combined with the voting condition, the selected unlabeled set at the t-th iteration is:

$$\tilde{\mathscr{D}}_{T_{ul}}^{(t)} \leftarrow \{x_i | f_{S,a}^{(t)}(u_i) \geq \mathbb{T}_a^{(t)} \text{ and } \hat{y}_{i,a}^S = \hat{y}_{i,a}^T = 1\}_{a \in \mathscr{C}}, \tag{6.1}$$

where x_i is the input of user u_i. We do upsampling to get a balanced distillation dataset. Denote the number of samples in $\tilde{\mathscr{D}}_{T_{ul}}^{(t)}$ that are predicted to be class a (i.e., $\hat{y}_{i,a}^S = 1$) as $\hat{N}_a$. For each class a, we randomly sample $max\{\hat{N}_m\}_{m \in \mathscr{C}} - \hat{N}_a$ samples from the original training set, making all classes have the same amount of samples. Denoted the labeled dataset sampled from the original training set as $\tilde{\mathscr{D}}_{T_{lb}}^{(t)}$. The resulting balanced distillation dataset $\tilde{\mathscr{D}}_T^{(t)} = \tilde{\mathscr{D}}_{T_{ul}}^{(t)} \cup \tilde{\mathscr{D}}_{T_{lb}}^{(t)}$.

Knowledge Distillation. We use the latest trained model $f_S^{(t)}$ as the teacher to make predictions on the unlabeled set $\tilde{\mathscr{D}}_{T_{ul}}^{(t)}$. Denote $f_S^{(t)}(u_i)$ as the predicted probability of user u_i given by $f_S^{(t)}$. We let $f_T^{(t)}$ approximate the predicted probabilities of samples in $\tilde{\mathscr{D}}_{T_{ul}}^{(t)}$ and the ground-truth of samples in $\tilde{\mathscr{D}}_{T_{lb}}^{(t)}$. The distillation loss is:

$$\tilde{\mathscr{L}}_T^{(t)} = -\frac{1}{|\tilde{\mathscr{D}}_{T_{ul}}^{(t)}|} \sum_i f_S^{(t)}(u_i) \log p_i^{(t)} - \frac{1}{|\tilde{\mathscr{D}}_{T_{lb}}^{(t)}|} \sum_j y_j \log p_j^{(t)}, \tag{6.2}$$

where $p_i^{(t)}$ is the predicted probability of user u_i given by $f_T^{(t)}$. In this way, the knowledge from the latest spatial model $f_S^{(t)}$ is distilled to the temporal model $f_T^{(t)}$. A similar process applies to the distillation for spatial model $f_S^{(t+1)}$, which uses $f_T^{(t)}$ and $f_S^{(t)}$ for voting ensemble and $f_T^{(t)}$ as

the teacher for distilling knowledge.

6.3.2 Spatial Module

To utilize spatial information, we visualize the mobility records as *visualized maps*. The maps contain rich information about the spatial structure. Even with minimal supervision, locations that are not present in the labeled set may still have a similar geographical distribution on a map. This enables the model to better generalize to unseen locations. A visualized map is a one-channel image M showing the regions that the user has been to during a time period. Given the set of regions $\mathscr{L}$ that a user has been to during time period t, we visualize the regions on the map of the city by highlighting the regions in $\mathscr{L}$ in white and marking the others in black. The images are rendered using the GeoPandas Library [76]. The maps of two time periods t_1 and t_2 can be aggregated by taking the maximum in each entry of the two maps, describing the regions that the user has been to during t_1 and t_2. In STCOLAB, we leverage maps of two different temporal granularities: *hour map* and *day map*. The hour map includes the locations visited in each hour and the day map is the aggregated map of the 24 hours. Figure 6.2b gives examples of the visualized maps generated from a random person's daily mobility records in the Brasilia dataset.

The spatial module learns spatial features, such as the geographic location and distance of people's daily trajectories, from the visualized map M. The spatial module performs convolution operation upon M as $c = \text{conv}(W_c, M) + b_c$, where W_c is the weight matrix and b_c is the bias term. We use Parametric ReLU (PReLU) as the activation function and use max pooling after the convolution operations. Then, a Fully-Connected (FC) Layer is applied to generate vector $z \in \mathscr{R}^d$ representing the spatial features learned from the map image.

To train the spatial module, we aggregate the visualized maps of each hour to get the day map M_i^{day}. The spatial module extracts spatial features z_i^{day} from M_i^{day} and applies an FC layer to make predictions. We denote p_i^{spatial} as the spatial module's predicted probability and y_i as the

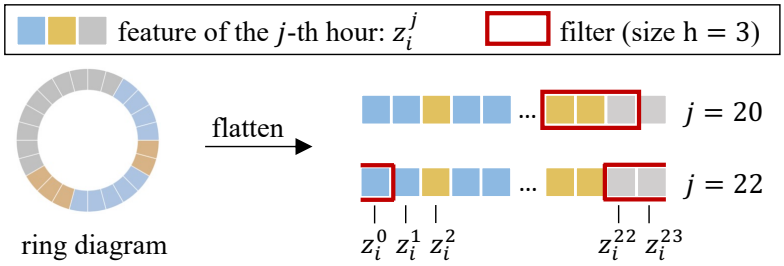


Figure 6.4. Illustration of periodic convolution.

ground truth. The learning objective is to minimize the cross entropy loss:

$$\mathscr{L}_S = -1/|\mathscr{D}_{lb}| \sum_i y_i \log p_i^{\text{spatial}}. \tag{6.3}$$

6.3.3 Temporal Module

The temporal module takes the sequence of spatial features in each hour of the day as inputs and is equipped with convolution layers with different filters sliding over the sequence to extract temporal features within different time periods.

Let $z_i^j \in \mathscr{R}^d$ be the spatial representation vector corresponding to the j-th hour of user u_i. The input to the temporal module is a sequence of the spatial features of each hour. We denote the concatenation of the spatial features from the j-th to the $(j+h)$-th hours of user u_i as $z_i^{j:j+h}$. A convolution filter of size $h \times d$ moves along the time dimension and is applied to a window of h hours to produce a new feature $e^{j:j+h}$ each time. For example, a feature e_j is generated from a window (of size h) of spatial features $z_i^{j:j+h}$ by $e_j = W_e \cdot z_i^{j:j+h} + b_e$, where W_e is the weight matrix of filter kernel and $b_e \in R$ is the bias term. This filter is applied to each possible window of spatial features in the sequence.

A person's temporal mobile pattern can be represented as a cyclic ring, where the beginning and the end of a day are continuous and connected to each other. To capture the cyclic patterns, we employ periodic convolution operations through circular padding. Specifically, the start of the sequence is padded with features from the end of the sequence, and vice versa. Figure 6.4 illustrates how periodic convolution works.

We use PReLU as the activation function and apply a max-pooling operation over the spatial features to get the features corresponding to a particular filter. The module uses N_f filters (with varying window sizes) to obtain multiple features and then concatenates them together to get the temporal features. An FC Layer is applied to make predictions. We denote p_i^{temporal} as the temporal module's predicted probability. The learning objective is to minimize the cross-entropy loss:

$$\mathscr{L}_T = -1/|\mathscr{D}_{lb}| \sum_i y_i \log p_i^{\text{temporal}}. \tag{6.4}$$

6.4 Experiments

6.4.1 Experimental Setup

Datasets and Prediction Tasks. We conduct experiments on *Chicago Dataset*[1] and *Brasilia Dataset*[2]. They collect mobility data in two cities in the United States and Brazil. We remove visits outside of the designated city and filter out users whose mobility records span less than 12 hours. For the Chicago dataset, we predict employment status (i.e., employed or not) and ethnicity (i.e., Caucasian, African American, or others). For the Brasilia dataset, we predict employment status, education level (i.e., whether the person has a college degree), and age group (i.e., ≤ 17, 18-59, or ≥ 60).

Implementation Details. The dimension of spatial features $d = 64$. The filter sizes in the temporal module are $[3, 5, 12]$. The convolution layers in the spatial module output 32 channels. The maximum number of iterations is 5. All experiments are repeated 5 times with a fixed set of random seeds.

Metrics. Due to label imbalance, we adopt macro- and micro-F1 scores to evaluate the performance. For all compared methods, we rank the prediction probabilities of the test data and assign classes according to label distribution [139, 225]. The label distributions are estimated by randomly sampling $N_{est} = 100$ pieces of data from the training set and calculating the ratio of

[1] https://datahub.cmap.illinois.gov/dataset/traveltracker0708
[2] https://metro.df.gov.br/?page_id=47685

each class. We show STCOLAB is robust to N_{est} via sensitivity analysis.

6.4.2 Compared Methods

We compare STCOLAB with the state-of-the-art methods designed for demographic inference from mobility data and for general-purpose spatio-temporal tasks.

- **L2P** [249] is a tensor factorization-based method for demographic inference from location check-ins. It extracts spatial and temporal semantics from check-ins and mines location knowledge from social networks and customer reviews. User representations are obtained by tensor factorization and are used to train classifiers.
- **SUME** [215] is an embedding-based method that learns mobility patterns by modeling a heterogeneous network that describes relations among users and locations. SVM is adopted for classification with learned embeddings.
- **Transformer** [192] is a neural network model which learns temporal patterns from sequential data and utilizes multi-head attention mechanism to select important inputs. We organize the data of each user into a sequence of location IDs, showing the main locations where the person stays during each hour of the day.
- **ConvLSTM** [175] is a recurrent neural network for spatio-temporal prediction. It receives a sequence of visualized maps as input, uses the convolutional networks to extract features from the maps, and feeds the features into the LSTM networks in chronological order.

We also craft two strong baselines by using some modules in STCOLAB. For a fair comparison with the state-of-the-art methods, the modules in both baselines are combined through late fusion by taking the average of the outputs as the final prediction and are updated together in back-propagation.

- **CNN+Transformer** uses convolutional networks for the spatial module (same as STCOLAB) and Transformer for the temporal module to learn patterns from location IDs.
- **CNN+PeriodicCNN (our ablation)** is equipped with the same spatial and temporal modules in STCOLAB.

Table 6.1. Experimental results averaged over 5 runs. The first section of the table compares different neural architectures. The second section focuses on different fusion solutions. The third section shows different self-training methods. We use * to mark the ablations of STCOLAB.

Method	Comment	Chicago				Brasilia					
		Employment		Ethnicity		Employment		Education		Age	
		Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1
L2P [249]	baseline models for demographic inference and general-purpose spatio-temporal tasks	49.70	61.40	25.80	54.50	49.70	69.10	49.20	50.30	31.12	55.54
SUME [215]		50.40	59.30	37.20	46.10	41.00	70.00	51.20	52.50	30.85	54.71
Transformer [192]		50.42	62.51	30.20	40.89	50.36	67.79	50.05	51.11	28.14	51.43
ConvLSTM [175]		53.10	64.10	48.37	59.54	52.87	69.45	50.72	52.16	29.06	53.55
CNN+Transformer		53.30	64.71	53.67	65.28	50.56	67.97	50.39	51.48	29.91	50.53
CNN+PeriodicCNN*		54.82	65.82	54.46	66.71	52.54	69.24	48.83	49.93	29.20	52.14
S+T	late fusion	54.82	65.82	54.46	66.71	52.54	69.24	48.83	49.93	29.20	52.14
STFC	intermediate fusion	53.34	64.67	54.06	65.93	51.76	68.73	51.48	52.54	30.48	52.10
ST*	alternating training	55.68	65.61	56.08	68.97	52.43	69.13	52.97	54.06	30.75	54.52
ST w/ CL [23]	w/ self-training	54.85	65.79	54.66	65.92	54.76	70.62	49.98	51.07	33.25	55.46
STCOLAB		**56.47**	**67.02**	**58.87**	**75.27**	**54.77**	**70.67**	**59.34**	**60.21**	**35.64**	**59.37**

Additionally, we use the same spatial and temporal modules in STCOLAB and compare different ways to combine them.

- **STFC** adopts intermediate fusion: the last hidden outputs of the two modules are concatenated and an FC layer is applied toward the concatenated features to make predictions.
- **S+T** adopts the late fusion which takes the average of the outputs given by the two modules as the final prediction.
- **ST (our ablation)** follows the alternating training in STCOLAB and is trained for only one iteration.

We denote our proposed method as **STCOLAB**. We compare STCOLAB with a state-of-the-art self-training method by pairing it with the same base model as in STCOLAB.

- **ST w/ CL** adopts curriculum labeling (CL) [23]. It uses a self-paced curriculum and re-initializes the model at each round to avoid concept drift.

6.4.3 Main Experimental Results and Analysis

The results are shown in Table 6.1. Overall, STCOLAB performs the best compared to all the baseline models. The existing methods for demographic inference from mobility data and for

general-purpose spatial-temporal tasks show inferior performance in the minimally-supervised setting. The performance of L2P, SUME, and Transformer indicates the limitation of these methods in capturing the geographic distribution. By comparison, the models which learn from visualized maps (i.e., ConvLSTM, CNN+Transformer, and CNN+PeriodicCNN) show better results.

The comparison among STFC, S+T and ST shows the advantage of alternating training over using conventional ways of modal fusion. Simply combining the predictions or intermediate hidden features of the two modules together does not fully leverage their respective strengths and may even yield worse results than using either module alone.

Finally, the comparison among ST, ST w/ CL, and STCOLAB shows the effectiveness of iterative collaborative distillation. Applying CL does not guarantee improvement and even causes performance degradation on some tasks. This suggests that, in the minimally-supervised setting, iteratively guiding the model with pseudo labels generated by the same model, hence the same view, may harm model performance. By comparison, STCOLAB provides guidance from two different views, which lets the two modules give complementary supervision for each other and mutually enhance themselves.

6.4.4 Ablation Studies and Sensitivity Analysis

More Comparisons with State-of-the-Art Self-Training Method. We further compare STCOLAB with ST w/ CL by applying them to three different model architectures. (1) **CNN & Transformer** uses convolutional networks as the spatial module (same as STCOLAB) and uses Transformer as the temporal module to learn temporal patterns from location IDs. The average of the predictions given by the two modules is taken as the final results. (2) **CNN & LSTM** uses convolutional networks to process the visualized maps and uses LSTM as the temporal module to process the spatial features of each hour generated by the convolutional networks in chronological order. (3) **CNN & PeriodicCNN** is the architecture in STCOLAB. As shown in Figure 6.5, STCOLAB is robust to different model architectures and always brings improvement

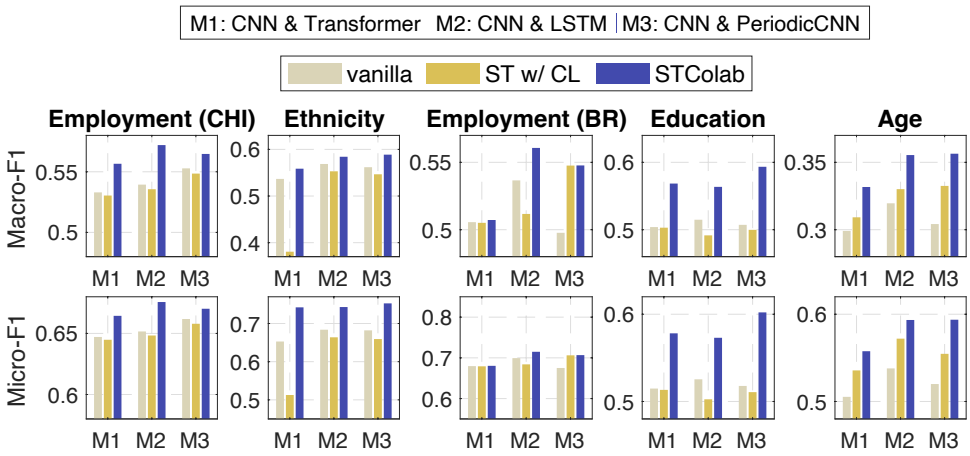


Figure 6.5. Performance of STCOLAB and ST w/ CL applied to different model architectures. Base models without applying self-training strategies are denoted as vanilla. (CHI: Chicago, BR: Brasilia)

to the vanilla models, while applying CL to the models may lead to worse performance.

Key Designs in Iterative Collaborative Distillation. We examine three ablations of STCOLAB. (1) **STCOLAB w/o vote** removes the voting ensemble strategy and uses all unlabeled samples as the distillation set. (2) **STCOLAB w/ union** uses the union of the prediction probabilities from both the latest trained spatial and temporal models to distill knowledge at every distillation process. (3) **STCOLAB w/o balance** removes the upsampling for getting a balanced distillation dataset and uses the dataset selected by the voting ensemble directly. The results are shown in Table 6.2. We notice performance degradation after replacing the key designs with the ablations, which indicates the importance of these designs. The voting ensemble helps the model choose the appropriate distillation set, giving better guidance during distillation. Moreover, by letting two modules alternate as teachers, the student model acquires complementary knowledge from the other modality and avoids confirmation bias. Furthermore, the balancing strategy ensures the number of samples for each class is the same and does not affect learning.

Contributions of Spatial and Temporal Modules. We conduct experiments on the model with spatial module only (denoted as **Spatial**), and the model with temporal module only

Table 6.2. Ablation studies on the contributions of the key designs in iterative collaborative distillation and the spatial and temporal modules.

Method	Chicago				Brasilia					
	Employment		Ethnicity		Employment		Education		Age	
	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1
Spatial	54.56	65.62	55.33	68.50	50.42	67.86	49.85	50.93	29.69	52.58
Temporal	53.00	64.51	53.03	64.92	51.64	68.76	52.95	53.98	30.13	53.03
ST	55.68	65.61	56.08	68.97	52.43	69.13	52.97	54.06	30.75	54.52
SCOLAB	54.10	65.31	56.28	70.47	50.57	67.92	56.39	57.32	33.63	56.43
TCOLAB	53.75	65.07	52.38	68.11	52.62	68.54	58.93	59.85	35.00	57.61
STColab w/o vote	56.08	66.77	57.35	70.21	51.67	68.63	59.18	60.05	32.99	55.02
STCOLAB w/ union	55.92	66.64	57.84	71.74	51.70	68.71	58.61	59.50	33.50	56.92
STCOLAB w/o balance	56.00	66.68	56.06	73.01	52.67	69.32	50.43	51.50	28.94	51.91
STCOLAB	**56.47**	**67.02**	**58.87**	**75.27**	**54.77**	**70.67**	**59.34**	**60.21**	**35.64**	**59.37**

which is trained from scratch including the spatial feature extractor (denoted as **Temporal**). The models are trained for only one pass without knowledge distillation. We also examine the single modules with knowledge distillation. We use the latest trained model at the previous iteration to construct the distillation dataset based on percentile score and to distill knowledge at the current iteration. **TCOLAB** and **SCOLAB** are the ablations of STCOLAB with only the temporal module and only the spatial module respectively. As shown in Table 6.2, removing either module will cause performance degradation. The performance of ST is slightly better than that of Temporal. This indicates the spatial features extracted by the pretrained spatial model are label-indicative, which improves the inference ability. By comparing SCOLAB, TCOLAB, and STCOLAB, we observe that knowledge distillation with a single module does not guarantee better performance. This again demonstrates the importance of the proposed collaborative distillation strategies for combining the modalities.

Number of Labeled Samples. We evaluate the system performance with even less label information. To do so, we decrease the number of training samples per class k and compare the performance of ST, ST w/ CL, and STCOLAB. The results are shown in Figure 6.6. In general, STCOLAB outperforms the vanilla model and the model applying CL.

Robustness of Class Distribution Estimation.. The parameter N_{est} is used when

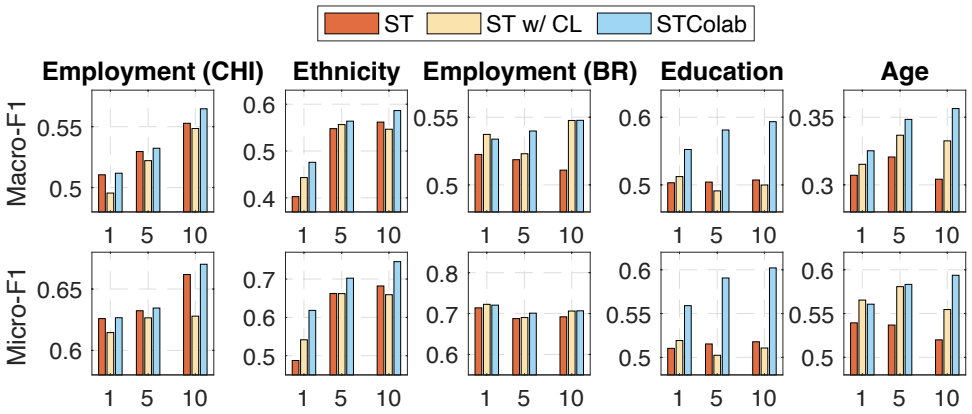


Figure 6.6. Performance of ST, ST w/ CL and STCOLAB w.r.t the number of labeled samples per class. (CHI: Chicago, BR: Brasilia)

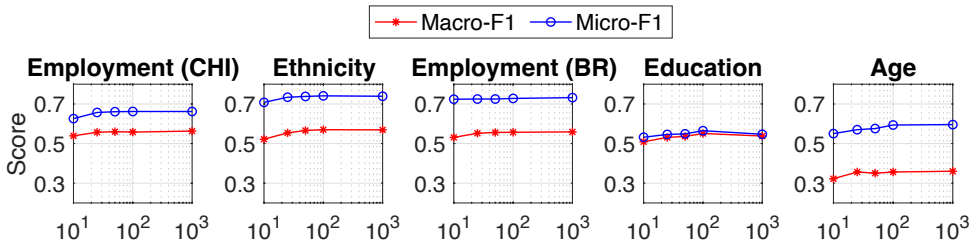


Figure 6.7. Performance of STCOLAB w.r.t number of sampled data N_{est} for class distribution estimation. (CHI: Chicago, BR: Brasilia)

randomly selecting training samples for estimating the class ratio. To test the robustness of STCOLAB with respect to this parameter, we change N_{est} to different values. As shown in Figure 6.7, the differences in performance are small when increasing or decreasing N_{est}, which shows that STCOLAB is robust to N_{est}.

6.5 Related Work

Contextual Inference from Human Mobility. Our work studies one of the important lines in contextual inference—inferring the demographic attributes of mobile users. The demographic inference problem has been studied with the support of abundant behavioral data from various fields, such as web and social media activities [16, 30, 206], transactions [201, 83, 235] and ratings [171]. Mobile data, which is ubiquitous in life, has been proven to have correlations with people's demographics [131, 238]. Several methods have been proposed for demographic

inference from human mobility including tensor factorization-based methods, [249, 143] and network embedding-based method [215]. These studies typically require a large number (e.g., thousands) of users to share labels for model training. By contrast, we seek to develop a data-efficient method that can achieve meaningful results with a very small amount of annotated data. To the best of our knowledge, we are the first to address the contextual inference problem using mobility data with minimal supervision.

Minimally Supervised Classification. Minimally supervised classification is an extreme case of semi-supervised learning. It is a challenging problem due to the scarcity of labeled training data. The problem has recently received much attention [245, 241]. A similar setting is few-shot learning which focuses on learning from limited data samples. The key difference between the two settings is that few-shot learning does not consider the availability of unlabeled data. Common solutions for few-shot classification such as metric learning [86] and meta-learning [43] are not ideal for our scenario, as they do not utilize underlying data distribution from massive unlabeled data. Self-training is arguably the most popular semi-supervised method for mitigating label scarcity. Self-training strategies [23, 114, 174] achieve remarkable results by iteratively utilizing the predictions of unlabeled data from previous rounds to augment the training set and support subsequent rounds of training. Mobility data usually consists of information from two different modalities. Traditional self-training solutions that bootstrap a single model using all features at once would incur a high risk of overfitting. Different from traditional methods, we propose collaborative distillation between the spatial and temporal modules in self-training cycles, which fuses information and improves generalization.

6.6 Summary

We studied the problem of contextual inference in the context of minimal supervision. We proposed STCOLAB and demonstrated its capability in predicting demographic attributes from human mobility. STCOLAB learns mobility features from spatial and temporal information

alternatingly and adopts iterative collaborative distillation to enhance model generalization in self-training cycles. STCOLAB achieves reasonable accuracy with only a small amount of annotated data (i.e., 10 samples per class), outperforming the state-of-the-art methods.

Chapter 6 incorporates material from the publication "Minimally Supervised Contextual Inference from Human Mobility: An Iterative Collaborative Distillation Framework", by Jiayun Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang, published in Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI), 2023. The dissertation author was the primary investigator and the lead author of this paper.

Chapter 7

Conclusion and Future Works

7.1 Summary of Contributions

This dissertation explores methods for effective learning in distributed environments and addresses the heterogeneity problem in terms of data and computation. In particular, we focus on federated learning and pretraining with fine-tuning paradigms.

In the first part, we leverage semantic class names and contextual information to align and adapt models across nodes with varying data distributions. These strategies enhance the robustness of machine learning models in non-IID settings. In the second part, we incorporate graph hypernetwork to facilitate collaboration among heterogeneous models across nodes and design an asynchronous method to accelerate training while maintaining convergence quality. These approaches accommodate diverse computing resources and varying network conditions, improving learning efficiency. In the third part, we further study methods to retrieve auxiliary contextual information that can be integrated with prior methods. This contextual knowledge could bridge heterogeneous sources and environments across distributed nodes, enhancing model quality.

These methods are generally model-agnostic, making them applicable to different model architectures that can handle various data types including image, language, and time-series data. Through theoretical analyses and extensive experiments, we demonstrate that these approaches show effective across diverse real-world applications, including healthcare, activity recognition,

language modeling, network anomaly monitoring.

7.2 Limitations and Future Directions

This dissertation addresses effectiveness in three key aspects: robustness, scalability, and quality. However, this is not the definitive form of an effective learning framework. Several open challenges remain from different perspectives. Below, we outline potential directions for future research.

Heterogeneous Modalities. This dissertation mainly focuses on unified feature structures across distributed nodes such as textual, image, or sensor data. However, many practical applications involve multi-modal inputs (e.g., images, audio, and text combined) [230, 223, 79, 149, 110]. In a distributed setting, individual nodes may only collect a subset of certain modalities, and the subsets vary across nodes [246]. This increases the complexity of data alignment and model aggregation. We plan to develop multi-modal fusion methods for incomplete modalities across different nodes to reduce the requirement for unified modality collection and enhance the robustness of learning.

Dynamic Node Participation. In distributed machine learning systems, it is common for new nodes to join the training process [150] or for previous nodes to leave the network over time [218, 220, 133, 36]. Scalability to new clients and tasks is essential for long-term performance. Ensuring that models adapt effectively to these changing conditions while retaining prior knowledge remains a significant challenge. We plan to extend our methods with lifelong learning [193, 218, 220, 133, 36, 150] to accommodate the joining and leaving of nodes without disrupting the learning progress made before.

Failure Handling. Training in a large-scale network faces a high risk of client or system failures due to issues such as out-of-memory (OOM) errors during training, hardware malfunctions, or network interruptions [69, 54, 71, 208]. As model sizes and the number of participants grow, so do the computational and communication demands, alongside the

probability of failures. The ability to effectively handle these failures is critical for ensuring robust and reliable training in distributed environments. We plan to integrate fault-tolerant mechanisms that gracefully recover from node disconnections or system crashes, reducing stall time due to failure.

Diverse Network Connectivity Structures. The methods presented in this dissertation are based on a star topology, where a central coordinator collects and distributes information to all nodes. In practice, however, there are various network connectivity structures, such as peer-to-peer or hierarchical topologies [222], each having unique communication and synchronization patterns. We plan to investigate these additional network topologies and generalize our aggregation strategies to be more flexible in these conditions.

Privacy and Security Considerations. Deploying machine learning on distributed nodes often requires advanced privacy-preserving and secure aggregation mechanisms [145]. For future research, we plan to combine differential privacy or homomorphic encryption with the proposed techniques to protect sensitive data, and develop adversarially robust models to counter malicious participants or data injections.

In conclusion, this dissertation makes important strides in addressing the core challenges of effective learning in heterogeneous distributed environments. By uniting strategies that tackle statistical and system heterogeneity and introducing auxiliary methods for extracting contextual information, we lay a strong foundation for the development of scalable, reliable, and high-quality distributed machine learning systems. As the field continues to evolve, we anticipate that further exploration of the outlined directions will bring up models capable of more dynamically adapting to real-world constraints and maintaining strong performance in an ever-expanding range of applications.

Appendix A

Theoretical Derivations and Proofs

A.1 Chapter 2: FEDALIGN

Notations. Let $f : \mathscr{X} \to \mathscr{Z}$ be a representation function that maps inputs to features, and $g : \mathscr{X} \to \{0,1\}$ be a ground-truth labeling function that maps input to output space. For the global domain, denote $\mathscr{D}$ as the global distribution over the input space $\mathscr{X}$, and $\tilde{\mathscr{D}}$ as the induced global distribution over the feature latent space $\mathscr{Z}$. For the m-th local domain, denote $\mathscr{D}_m$ as the local distribution and $\tilde{\mathscr{D}}_m$ be the induced image of $\mathscr{D}_m$ over $\mathscr{Z}$. A hypothesis $h : \mathscr{Z} \to \{0,1\}$ is a function that maps features to predicted labels. Let $\tilde{g}$ be the induced image of g over $\mathscr{Z}$. The expected risk of hypothesis h on distribution $\mathscr{D}$ is defined as follows:

$$\mathscr{L}(h) = \mathbb{E}_{z\sim\tilde{\mathscr{D}}}[\mathbb{E}_{y\sim\tilde{g}(z)}[y \neq h(z)]].$$

Let λ_m denote the risk of the optimal hypothesis over hypothesis class $\mathbf{H}$ that has minimum risk on both $\mathscr{D}$ and $\mathscr{D}_m$ distributions, i.e., $\lambda_m = \min_{h\in\mathbf{H}}(\mathscr{L}(h) + \mathscr{L}_m(h))$.

We define distance functions for measuring the divergence between two distributions with respect to the hypothesis class. First, given a feature space $\mathscr{Z}$ and a collection of measurable subsets $\mathscr{A}$ of $\mathscr{Z}$, define $\mathscr{A}$-distance between two distributions $\tilde{\mathscr{D}}$ and $\tilde{\mathscr{D}}'$ on $\mathscr{Z}$ as:

$$d_{\mathscr{A}}(\tilde{\mathscr{D}}, \tilde{\mathscr{D}}') = 2 \sup_{A\in\mathscr{A}} |\Pr_{\tilde{\mathscr{D}}}(A) - \Pr_{\tilde{\mathscr{D}}'}(A)|.$$

Now, fix a particular hypothesis class $\mathbf{H}$, for any given $h \in \mathbf{H}$, define $\mathscr{Z}_h = \{z \in \mathscr{Z} | h(z) = 1\}$ and $\mathscr{A}_{\mathbf{H}} = \{\mathscr{Z}_h | h \in \mathbf{H}\}$. Define the $\mathbf{H}$-divergence between two distributions $\tilde{\mathscr{D}}$ and $\tilde{\mathscr{D}}'$ on $\mathscr{Z}$ as:

$$d_{\mathbf{H}}(\tilde{\mathscr{D}}, \tilde{\mathscr{D}}') = d_{\mathscr{A}_{\mathbf{H}}}(\tilde{\mathscr{D}}, \tilde{\mathscr{D}}').$$

Furthermore, given a particular hypothesis class $\mathbf{H}$, define $\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}} = \{\mathscr{Z}_h \Delta \mathscr{Z}_{h'} | h, h' \in \mathbf{H}\}$, where Δ operation is the symmetric difference in the sense of set operation. Define the $\mathbf{H}\Delta\mathbf{H}$-divergence between two distributions $\tilde{D}$ and $\tilde{D}'$ on $\mathscr{Z}$ as:

$$d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}, \tilde{\mathscr{D}}') = d_{\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}}}(\tilde{\mathscr{D}}, \tilde{\mathscr{D}}').$$

Theorem A.1.1 (Generalization Bound of Federated Learning[1])**.** *Assume there are M clients in a federated learning system. Let $\mathbf{H}$ be the hypothesis class with VC-dimension d. The global hypothesis is the aggregation of h_m, i.e., $h = \frac{1}{M}\sum_{m\in[M]} h_m$. With probability at least $1-\delta$, for $\forall h \in \mathbf{H}$:*

$$\begin{aligned}\mathscr{L}(h) \leq & \frac{1}{M}\sum_{m\in[M]} \hat{\mathscr{L}}_m(h_m) + \frac{1}{M}\sum_{m\in[M]} [d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}}) + \lambda_m] \\ & + \sqrt{\frac{4}{N}(d\,log\,\frac{2eN}{d} + log\,\frac{4M}{\delta})},\end{aligned}$$

where $\hat{\mathscr{L}}_m(h_m)$ is the empirical risk on the m-th client given N observed samples, e is the base of the natural logarithm.

Corollary A.1.1.1 (Generalization Bound of Federated Learning with Mix-up Distributions)**.** *Let $\mathscr{D}'_m$ denote the distribution added for adapting the m-th client. Define the new distribution $\mathscr{D}^*_m$ to be a mixture of the original local distribution and the adaptation distribution, i.e., $\mathscr{D}^*_m = \alpha\mathscr{D}_m + (1-\alpha)\mathscr{D}'_m$, where $\alpha \in [0,1]$ is the weight of the original distribution decided by the number of empirical samples added. Let $\mathbf{H}$ be the hypothesis class with VC-dimension d. The global hypothesis is the aggregation of h_m, i.e., $h = \frac{1}{M}\sum_{m\in[M]} h_m$. With probability at least*

[1]Proof can be found in prior work [152, 251, 120].

$1-\delta$, *for* $\forall h \in \mathbf{H}$*:*

$$\begin{aligned}\mathscr{L}(h) \leq & \frac{1}{M}\sum_{m\in[M]}\hat{\mathscr{L}}_m(h_m) \\ & + \frac{1}{M}\sum_{m\in[M]}[\alpha d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m,\tilde{\mathscr{D}}) + (1-\alpha)d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}'_m,\tilde{\mathscr{D}}) + \lambda_m] \\ & + \sqrt{\frac{4}{N^*}(d\,log\,\frac{2eN^*}{d} + log\,\frac{4M}{\delta})},\end{aligned}$$

where $\hat{\mathscr{L}}_m(h_m)$ *is the empirical risk on the m-th client given* N^* $(N^* > N)$ *observed samples, e is the base of the natural logarithm.*

Proof. Apply Theorem A.1.1 to the mix-up distribution $\mathscr{D}^*_m$, we have that with probability at least $1-\delta$, for $\forall h \in \mathbf{H}$:

$$\begin{aligned}\mathscr{L}(h) \leq & \frac{1}{M}\sum_{m\in[M]}\hat{\mathscr{L}}_m(h_m) + \frac{1}{M}\sum_{m\in[M]}[d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}^*_m,\tilde{\mathscr{D}}) + \lambda_m] \\ & + \sqrt{\frac{4}{N^*_m}(d\,log\,\frac{2eN^*_m}{d} + log\,\frac{4M}{\delta})}.\end{aligned}$$

We derive the upper bound of $\mathbf{H}\Delta\mathbf{H}$-divergence between the mix-up distribution $\mathscr{D}^*_m$ and the global distribution $\mathscr{D}$ as follows:

$$\begin{aligned}d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}^*_m,\tilde{\mathscr{D}}) &= 2\sup_{A\in\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}}}|\mathrm{Pr}_{\tilde{\mathscr{D}}^*_m}(A) - \mathrm{Pr}_{\tilde{\mathscr{D}}}(A)| \\ &= 2\sup_{A\in\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}}}|\mathrm{Pr}_{\alpha\tilde{\mathscr{D}}_m+(1-\alpha)\tilde{\mathscr{D}}'_m}(A) - \mathrm{Pr}_{\tilde{\mathscr{D}}}(A)| \\ &= 2\sup_{A\in\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}}}|\alpha\mathrm{Pr}_{\tilde{\mathscr{D}}_m}(A) + (1-\alpha)\mathrm{Pr}_{\tilde{\mathscr{D}}'_m}(A) - \mathrm{Pr}_{\tilde{\mathscr{D}}}(A)| \\ &\leq 2\alpha\sup_{A\in\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}}}|\mathrm{Pr}_{\tilde{\mathscr{D}}_m}(A) - \mathrm{Pr}_{\tilde{\mathscr{D}}}(A)| \\ &\quad + 2(1-\alpha)\sup_{A\in\mathscr{A}_{\mathbf{H}\Delta\mathbf{H}}}|\mathrm{Pr}_{\tilde{\mathscr{D}}'_m}(A) - \mathrm{Pr}_{\tilde{\mathscr{D}}}(A)| \\ &= \alpha d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m,\tilde{\mathscr{D}}) + (1-\alpha)d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}'_m,\tilde{\mathscr{D}}).\end{aligned}$$

The inequality is derived using the triangle inequality. Replace the $\mathscr{A}$-distance $d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m^*, \tilde{\mathscr{D}})$ with its upper bound derived above, we get the upper bound of $\mathscr{L}(h)$.

If $d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}'_m, \tilde{\mathscr{D}}) < d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}})$, then we have:

$$\begin{aligned} d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m^*, \tilde{\mathscr{D}}) &\leq \alpha d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}}) + (1-\alpha) d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}'_m, \tilde{\mathscr{D}}) \\ &< d_{\mathbf{H}\Delta\mathbf{H}}(\tilde{\mathscr{D}}_m, \tilde{\mathscr{D}}). \end{aligned}$$

Furthermore, with added empirical samples from D'_m, we have $N^* > N$, and:

$$\sqrt{\frac{4}{N^*}(d\,log\,\frac{2eN^*}{d} + log\,\frac{4M}{\delta})} < \sqrt{\frac{4}{N}(d\,log\,\frac{2eN}{d} + log\,\frac{4M}{\delta})}.$$

Therefore, the upper bound of the expected risk with the mix-up distribution is lowered.

□

A.2 Chapter 3: REACT

We provide convergence analysis of REACT on linear models under the premise of Theorem 3.5.1, that the models h and f admit the form (3.4), the adaptive weights are updated by exactly solving Eq. (3.1), and relevant datasets are sampled at the beginning of the algorithm and fixed throughout the iterations.

Based on Eq. (3.4), we consider the following objective function:

$$\mathscr{L}(f(X;\theta_{\text{meta}},\theta_{\text{adapt}})) = \frac{1}{2}\|f(X;\theta_{\text{meta}},\theta_{\text{adapt}}) - Y\|^2 + \frac{\lambda}{2}\|\theta_{\text{adapt}}\|^2,$$

which consists of a mean squared error and an L2 regularization for the adaptive weights (see Section 3.4.4). Y is the target associated with the loss function. It can have different forms according to the underlying target model. For example, it can be the input data for reconstruction loss, center of samples for methods like DeepSVDD, or labels in cases of supervised or semi-supervised learning.

Without loss of generality we set $\theta \in \mathbb{R}^{d_1}$ and $\phi \in \mathbb{R}^{d_2}$ for some $d_1, d_2 > 0$. Notice that this assumption can be generalized by considering vectorization of the matrix product and hence our results can easily be extended to more generic output spaces. We also note that the assumption that the datasets have uncorrelated constant variance, i.e. $(X^i)^\top (X^i) = \sigma_i I$ is to simplify computations in the proof. The proof can be relaxed to bounded norm, i.e. $\|X^i\|_2 \leq \sigma_i$ where $\|.\|_2$ is L-2 norm on the matrix space.

We restate the theorem below:

Theorem A.2.1. *Consider REACT on the linear model in (3.4) with Eq. (3.1) being solved exactly. Let X_s^i and X_q^i satisfy $(X_s^i)^\top X_s^i = (X_q^i)^\top X_q^i = \sigma_i I$ for each task $i \in \{1, ..., M\}$, where σ_i are the variances and I is the identity matrix. Learning rates are chosen as $\eta_{meta} < \frac{1}{\sum_{i=1}^M \sigma_i \lambda/(\sigma_i+\lambda)}$ and $\eta_h < \frac{1}{\max\left(\sum_{j=1}^{n_h} \sigma_j(\sigma_j+\lambda), \|X_s\|\right)}$ where $X_s = \sum_{j=1}^M \sigma_j (X_s^j)^\top$. Then, for any $\varepsilon > 0$, there exists*

$$K = \mathscr{O}\left(\log_{1/\rho_{meta}}(1/\varepsilon) + \log_{1/\rho_h}(1/\varepsilon)\right)$$

where $\rho_{meta} = 1 - \eta_{meta} \sum_{i=1}^M \sigma_i \lambda/(\sigma_i + \lambda)$ and $\rho_h = 1 - \eta_h \sum_{j=1}^M \sigma_j(\sigma_j + \lambda)$ such that the K-iteration of Algorithm 2 satisfies

$$\|\theta^K - \theta^*\| \leq \varepsilon, \quad \textit{and} \quad \|\phi^K - \phi^*\| \leq \varepsilon,$$

where θ^* and ϕ^* are stationary points of the algorithm.

Proof. We prove the result following the steps in Algorithm 2. Let $\theta_{\text{adapt}}^{i,k}$ be the fine-tuned adaptive weights of task i at the k-th iteration of REACT and similarly, let ϕ^k denote the hypernetwork parameters and θ_{meta}^k be the meta weights at iteration k.

Task fine-tuning. The exact intermediate updates defined in (3.1) can be rewritten as follows:

$$\theta_{\text{adapt}}^{i,k+1} = \arg\min_\theta \frac{1}{2}\|X_s^i(\theta_{\text{meta}}^k + \theta) - Y_s^i\|^2 + \frac{\lambda}{2}\|\theta\|^2.$$

Setting gradient to zero, we have

$$\mathbf{0} = \left(X_s^i\right)^\top X_s^i \left(\theta_{\text{adapt}}^{i,k+1} + \theta_{\text{meta}}^k\right) - \left(X_s^i\right)^\top Y_s^i + \lambda\, \theta_{\text{adapt}}^{i,k+1},$$

where $\mathbf{0}$ is the vector of all zeros. This implies

$$\theta_{\text{adapt}}^{i,k+1} = \frac{1}{\sigma_i + \lambda} \left(X_s^i\right)^\top Y_s^i - \frac{\sigma_i}{\sigma_i + \lambda} \theta_{\text{meta}}^k, \tag{A.1}$$

where we used the fact that $\left(X_s^i\right)^\top X_s^i = \sigma_i I$.

Meta weight update. Next, we consider the gradient update of meta weight in (2). The gradient with respect to θ_{meta} is

$$\begin{aligned}
\nabla_{\theta_{\text{meta}}} \sum_{x \in \mathbf{D}_{\text{query}}^i} \quad & \mathscr{L}\left(f\left(x; \theta_{\text{meta}}, \theta_{\text{adapt}}^{i,k+1}\right)\right)\Big|_{\theta_{\text{meta}}^k} \\
& = \left(X_q^i\right)^\top X_q^i \left(\theta_{\text{meta}}^k + \theta_{\text{adapt}}^{i,k+1}\right) - \left(X_q^i\right)^\top Y_q^i \\
& = \sigma_i \left(\tfrac{\lambda}{\sigma_i+\lambda} \theta_{\text{meta}}^k + \tfrac{1}{\sigma_i+\lambda} \left(X_s^i\right)^\top Y_s^i\right) - \left(X_q^i\right)^\top Y_q^i,
\end{aligned}$$

where the last equality is given by (5) and the assumption in data covariance matrix. Therefore, the gradient update step is

$$\begin{aligned}
\theta_{\text{meta}}^{k+1} =& \theta_{\text{meta}}^k - \eta_{\text{meta}} \sum_{i=1}^M \sum_{x \in \mathbf{D}_{\text{query}}^i} \nabla_{\theta_{\text{meta}}} \mathscr{L}\left(f\left(x; \theta_{\text{meta}}^k, \theta_{\text{adapt}}^{i,k+1}\right)\right) \\
=& \theta_{\text{meta}}^k - \eta_{\text{meta}} \sum_{i=1}^M \sigma_i \left(\frac{\lambda}{\sigma_i + \lambda} \theta_{\text{meta}}^k + \frac{1}{\sigma_i + \lambda} \left(X_s^i\right)^\top Y_s^i\right) - \left(X_q^i\right)^\top Y_q^i \\
=& \left(1 - \eta_{\text{meta}} \sum_{i=1}^M \frac{\sigma_i \lambda}{\sigma_i + \lambda}\right) \theta_{\text{meta}}^k \\
& - \eta_{\text{meta}} \sum_{i=1}^M \frac{\sigma_i}{\sigma_i + \lambda} \left(X_s^i\right)^\top Y_s^i - \left(X_q^i\right)^\top Y_q^i.
\end{aligned}$$

Let us introduce $\rho_{\text{meta}} = 1 - \eta_{\text{meta}} \sum_{i=1}^M \sigma_i \lambda / (\sigma_i + \lambda)$, and choose learning rate $0 <$

$\eta_{\text{meta}} < 1/\sum_{i=1}^{M} \sigma_i\lambda/(\sigma_i+\lambda)$ so that $0 < \rho_{\text{meta}} < 1$.

The stationary point θ^*_{meta} should satisfy

$$\begin{aligned}\theta^*_{\text{meta}} =& \left(1-\eta_{\text{meta}}\sum_{i=1}^{M}\frac{\sigma_i\lambda}{\sigma_i+\lambda}\right)\theta^*_{\text{meta}} \\ &-\eta_{\text{meta}}\sum_{i=1}^{M}\frac{\sigma_i}{\sigma_i+\lambda}\left(X_s^i\right)^\top Y_s^i-\left(X_q^i\right)^\top Y_q^i.\end{aligned}$$

Thus, we obtain

$$\left(\theta^{k+1}_{\text{meta}}-\theta^*_{\text{meta}}\right)=\rho_{\text{meta}}\left(\theta^k_{\text{meta}}-\theta^*_{\text{meta}}\right),$$

yielding,

$$\left\|\theta^{k+1}_{\text{meta}}-\theta^*_{\text{meta}}\right\|\leq\rho_{\text{meta}}\left\|\theta^k_{\text{meta}}-\theta^*_{\text{meta}}\right\|\leq\cdots\leq\rho^{k+1}_{\text{meta}}\left\|\theta^0_{\text{meta}}-\theta^*_{\text{meta}}\right\|. \tag{A.2}$$

Hypernetwork update. Lastly, the gradient of the objective function update with respect to ϕ^k is

$$\begin{aligned}&\nabla_\phi\sum_{x\in\mathbf{D}^j_{\text{query}}}\mathscr{L}\left(f\left(x;\theta^{k+1}_{\text{meta}},h\left(X_s^j;\phi\right)\right)\right)\Bigg|_{\phi^k}\\ &=\left(X_q^jX_s^j\right)^\top\left(X_q^jX_s^j\phi^k+X_q^j\theta^{k+1}_{\text{meta}}\right)-\left(X_q^jX_s^j\right)^\top Y_q^j+\lambda\left(X_s^j\right)^\top X_s^j\phi^k\\ &=\sigma_j\left(\sigma_j+\lambda\right)\phi^k+\sigma_j\left(X_s^j\right)^\top\theta^{k+1}_{\text{meta}}-\left(X_q^jX_s^j\right)^\top Y_q^j\end{aligned}$$

Thus, the update (3.3) can be written as

$$\begin{aligned}\phi^{k+1}=&\phi^k-\eta_h\sum_{j=1}^{n_h}\sum_{x\in\mathbf{D}^j_{\text{query}}}\nabla_\phi\mathscr{L}\left(f\left(x;\theta^{k+1}_{\text{meta}},h\left(X_s^j;\phi\right)\right)\right)\\ =&\left(1-\eta_h\sum_{j=1}^{M}\sigma_j\left(\sigma_j+\lambda\right)\right)\phi^k\\ &-\eta_h\left(\sum_{i=1}^{M}\sigma_j\left(X_s^j\right)^\top\right)\theta^{k+1}_{\text{meta}}+\eta_h\sum_{i=1}^{M}\left(X_q^jX_s^j\right)^\top Y_q^j,\end{aligned}$$

where the last equality follows from $\left(X_q^j\right)^\top X_q^j = \left(X_s^j\right)^\top X_s^j = \sigma_j I$.

Notice that the learning rate η_h implies $0 < \eta_h < 1/\max\left(\sum_{j=1}^{n_h} \sigma_j\left(\sigma_j + \lambda\right), \|\mathbf{X}_s\|\right)$ so that the rate ρ_h and $\mathbf{X}_s$ satisfy $0 < \rho_h < 1$ and $0 < \eta_h \|\mathbf{X}_s\| < 1$. On the other hand, the stationary points ϕ^* and θ^* satisfy

$$\phi^* = \rho_h \phi^* - \eta_h \mathbf{X}_s \theta^*_{\text{meta}} + \eta_h \sum_{i=1}^{M} \left(X_q^j X_s^j\right)^\top Y_q^j,$$

yielding

$$\phi^{k+1} - \phi^* = \rho_h \left(\phi^k - \phi^*\right) - \eta_h \mathbf{X}_s \left(\theta^{k+1}_{\text{meta}} - \theta^*_{\text{meta}}\right)$$

and

$$\left\|\phi^{k+1} - \phi^*\right\| \leq \rho_h \left\|\phi^k - \phi^*\right\| + \eta_h \|\mathbf{X}_s\| \left\|\theta^{k+1}_{\text{meta}} - \theta^*_{\text{meta}}\right\| \tag{A.3}$$

Convergence. With Eq. (A.2), we can show that for $k \geq K_{\text{meta}} = \log_{1/\rho_m}(1/\varepsilon) + \log_{1/\rho_m}\left(\left\|\theta^0_{\text{meta}} - \theta^*_{\text{meta}}\right\|\right)$, we have

$$\left\|\theta^k_{\text{meta}} - \theta^*_{\text{meta}}\right\| \leq \varepsilon$$

Similarly, from Eq. (A.3), we get

$$\begin{aligned}
\left\|\phi^k - \phi^*\right\| &\leq \rho_h \left\|\phi^{k-1} - \phi^*\right\| + \eta_h \|\mathbf{X}_s\| \left\|\theta^k_{\text{meta}} - \theta^*_{\text{meta}}\right\| \\
&\leq \rho_h \left\|\phi^{k-1} - \phi^*\right\| + \eta_h \|\mathbf{X}_s\| \rho^k_{\text{meta}} \left\|\theta^0_{\text{meta}} - \theta^*_{\text{meta}}\right\| \\
&\leq \rho_h^2 \left\|\phi^{k-2} - \phi^*\right\| + \eta_h \|\mathbf{X}_s\| \left\|\theta^0_{\text{meta}} - \theta^*_{\text{meta}}\right\| \left[\rho^k_{\text{meta}} + \rho_h \rho^{k-1}_{\text{meta}}\right] \\
&\cdots \\
&\leq \rho_h^k \left\|\phi^0 - \phi^*\right\| + \frac{\rho_{\text{meta}}\, \eta_h \|\mathbf{X}_s\| \left\|\theta^0_{\text{meta}} - \theta^*_{\text{meta}}\right\|}{\rho_{\text{meta}} - \rho_h} \left(\rho^k_{\text{meta}} - \rho^k_h\right),
\end{aligned}$$

where we used (6) in the second inequality. Therefore, for any k satisfies

$$
\begin{aligned}
k \geq & K_h \\
= & \log_{1/\rho_h}(2/\varepsilon) + \log_{1/\rho_h}\left(\left\|\phi^0 - \phi^*\right\|\right) \\
& + \log_{1/\rho_h}(4/\varepsilon) + \log_{1/\rho_h}\left(\frac{\rho_{\text{meta}}\,\eta_h\,\|\mathbf{X}_s\|\,\left\|\theta^0_{\text{meta}} - \theta^*_{\text{meta}}\right\|}{|\rho_{\text{meta}} - \rho_h|}\right) \\
& + \log_{1/\rho_{\text{meta}}}(4/\varepsilon) + \log_{1/\rho_{\text{meta}}}\left(\frac{\rho_{\text{meta}}\,\eta_h\,\|\mathbf{X}_s\|\,\left\|\theta^0_{\text{meta}} - \theta^*_{\text{meta}}\right\|}{|\rho_{\text{meta}} - \rho_h|}\right),
\end{aligned}
$$

we have

$$
\left\|\phi^k - \phi^*\right\| \leq \frac{\varepsilon}{2} + \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \varepsilon.
$$

Therefore, we can choose

$$
K = \max\left(K_{\text{meta}}, K_h\right) = O\left(\log_{1/\rho_{\text{meta}}}(1/\varepsilon) + \log_{1/\rho_h}(1/\varepsilon)\right)
$$

This completes the proof.

□

A.3 Chapter 4: RECIPFL

Notations. We will give the proof of Theorem 4.4.1 using the results of Baxter [12]. Let us introduce the notations and definitions before we state a key theorem from Baxter (Theorem 18 and Corollary 19) from which the main results of the paper are derived.

Let $\mathscr{X}$ be the input space and $\mathscr{Y}$ be the output space. Let $\mathscr{P}_1, \ldots, \mathscr{P}_M$ be M probability measures on $\mathscr{X} \times \mathscr{Y}$. For every $m = 1, \ldots, M$, sample $(x^{(m)}, y^{(m)})$ from the distribution $\mathscr{P}_m$, and abbreviate $\mathscr{L}_m(\phi) = \ell\left(f_m\left(x^{(m)}; \text{GHN}(\mathscr{G}_m, a_m; \phi)\right), y^{(m)}\right)$, where ϕ represents the parameters of the graph hypernetwork. Define a metric $d_{\mathscr{P}}$ on $\mathbb{R}^d$:

$$
d_{\mathscr{P}}(\phi, \phi') = \frac{1}{M}\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathscr{P}}\left|\sum_{m=1}^{M}\mathscr{L}_m(\phi) - \sum_{m=1}^{M}\mathscr{L}_m(\phi')\right|, \tag{A.4}
$$

where $\mathscr{P} = \mathscr{P}_1 \times \cdots \times \mathscr{P}_M$ is the product probability measure, and $(\boldsymbol{x}, \boldsymbol{y}) = ((x^{(1)}, y^{(1)}), \ldots, (x^{(M)}, y^{(M)}))$. Define the covering number of a subset E of $\mathbb{R}^d$ by closed ball of radius ε with respect to the metric $d_{\mathscr{P}}$:

$$\mathscr{N}(\varepsilon, E, d_{\mathscr{P}}) = \inf\{n : \exists \phi_1, \cdots, \phi_n, \forall \phi \in E, \exists j, d_{\mathscr{P}}(\phi, \phi_j) \leq \varepsilon\} \tag{A.5}$$

and the capacity of $E \subset \mathbb{R}^d$ by

$$\mathscr{C}(\varepsilon, E) = \sup_{\mathscr{P}} \mathscr{N}(\varepsilon, E, d_{\mathscr{P}}), \tag{A.6}$$

where the supremum is taken over all product probability measures on $(\mathscr{X}, \mathscr{Y})^M$. The capacity measures the complexity of the hypothesis space in much the same way as the VC-dimension measures the complexity of a set of Boolean functions. Here our hypothesis space is indexed by $\phi \in \mathbb{R}^d$. Now we are ready to state the theorem from Baxter applied in our RECIPFL method.

Theorem A.3.1. *Let $\boldsymbol{D} = \{\boldsymbol{D}_m\}_{m=1}^M$ be generated by N independent trials from $(\mathscr{X} \times \mathscr{Y})^M$ according to some product probability measure $\mathscr{P} = \mathscr{P}_1 \times \cdots \times \mathscr{P}_M$. If*

$$N \geq \max\left\{ \frac{4}{M\varepsilon^2} \log \frac{4\mathscr{C}\left(\frac{\varepsilon}{4}, \mathbb{R}^d\right)}{\delta}, \frac{1}{\varepsilon^2} \right\}, \tag{A.7}$$

then

$$\mathbb{P}\left(\boldsymbol{D} : \sup_{\phi} |\mathscr{L}(\phi) - \hat{\mathscr{L}}(\phi, \boldsymbol{D})| > \varepsilon \right) \leq \delta. \tag{A.8}$$

Proof of Theorem 4.4.1. It suffices to bound $\mathscr{C}\left(\frac{\varepsilon}{4}, \mathbb{R}^d\right)$. Notice that by the Lipschitz assumption on the loss function l, $|\mathscr{L}_m(\phi) - \mathscr{L}_m(\phi')| \leq K\|\phi - \phi'\|$ for all $m = 1, \cdots, M$. This implies by (A.4), for all $\phi, \phi' \in \mathbb{R}^d$,

$$d_{\mathscr{P}}(\phi, \phi') \leq \frac{1}{M} \sum_{m=1}^{M} \left| \mathscr{L}_m(\phi) - \mathscr{L}_m(\phi') \right| \leq K\|\phi - \phi'\|. \tag{A.9}$$

So $\|\phi - \phi'\| \leq \frac{\varepsilon}{K}$ implies $d_{\mathscr{P}}(\phi, \phi') \leq \varepsilon$. Now take an integer $p > RK\sqrt{d}/\varepsilon$ and decompose $[-R,R]^d$ as the union of p^d congruent cubes by dividing $[-R,R]$ into p pieces of equal length. The side length of these cubes is $2R/p$ and so each cube is contained in a ball of radius $R\sqrt{d}/p < \frac{\varepsilon}{K}$ centered at the center of the cube. This proves the covering number $\mathscr{N}(\varepsilon, E, d_{\mathscr{P}}) \leq \lceil RK\sqrt{d}/\varepsilon \rceil^d$ for all $\mathscr{P}$. So, $\mathscr{C}\left(\frac{\varepsilon}{4}, \mathbb{R}^d\right) \leq \lceil 4RK\sqrt{d}/\varepsilon \rceil^d$. □

A.4 Chapter 5: ORTHOFL

In this section, we prove that the orthogonalization strategy in ORTHOFL preserves the maximal information from the global weight shift vectors perpendicular to the local update direction.

For $v, w \in \mathbb{R}^d$, let $\langle v, w \rangle := \sum_{i=1}^d v_i w_i$ be the standard inner product on $\mathbb{R}^d$. We restate the Lamma as follows:

Lemma A.4.1. *Let $v \in \mathbb{R}^d$ and $\mathscr{U} = \{u_1, \cdots, u_k\}$ be an orthonormal set for some $k < d$. Then for any $w \in (\operatorname{span} \mathscr{U})^\perp$,*

$$\|v - v^\perp\| \leq \|v - w\|, \tag{A.10}$$

where $v^\perp := v - \sum_{i=1}^k \langle v, u_i \rangle u_i$ denote the component of v orthogonal to $\mathscr{U}$. Moreover, the angle between v and $v^\perp$ is less than any angle between v and w for $w \in (\operatorname{span} \mathscr{U})^\perp$.

Proof. Extend $\mathscr{U}$ to an orthonormal basis $\mathscr{B} := \{u_1, \cdots, u_k,$
$u_{k+1}, \cdots, u_d\}$ on $\mathbb{R}^d$. Write $w = \sum_{i=1}^d w_i u_i$ and $v = \sum_{i=1}^d v_i u_i$, where $w_i := \langle w, u_i \rangle$ and $v_i := \langle v, u_i \rangle$ denote the coordinates of w and v with respect to the basis $\mathscr{B}$ respectively. Since $w \in (\operatorname{span} \mathscr{U})^\perp$,

$w_i = 0$ for $1 \leq i \leq k$. It follows from Pythagorean theorem that

$$\begin{aligned} \|v-w\|^2 &= \left\| \sum_{i=1}^{d} (v_i - w_i) u_i \right\|^2 \\ &= \left\| \sum_{i=1}^{k} v_i u_i + \sum_{j=k+1}^{d} (v_j - w_j) u_j \right\|^2 \\ &= \left\| \sum_{i=1}^{k} v_i u_i \right\|^2 + \left\| \sum_{j=k+1}^{d} (v_j - w_j) u_j \right\|^2 \\ &\geq \left\| \sum_{i=1}^{k} v_i u_i \right\|^2 . \end{aligned} \tag{A.11}$$

Taking square root of both sides and noting that by definition $v - v^{\perp} = \sum_{i=1}^{k} v_i u_i$, equation (A.10) follows. As for the second claim, let $\theta(v,w)$ denote the angle between v and w, $\theta(v,v^{\perp})$ the angle between v and $v^{\perp}$. By the Cauchy-Schwarz inequality,

$$\begin{aligned} \langle v, w \rangle &= \left\langle v, \sum_{j=k+1}^{d} w_j u_j \right\rangle \\ &= \sum_{j=k+1}^{d} w_j \langle v, u_j \rangle \\ &\leq \left(\sum_{j=k+1}^{d} w_j^2 \right)^{1/2} \cdot \left(\sum_{j=k+1}^{d} v_j^2 \right)^{1/2} \\ &= \|w\| \cdot \|v^{\perp}\|. \end{aligned} \tag{A.12}$$

It follows that

$$\cos\theta(v,w) := \frac{\langle v, w \rangle}{\|v\| \, \|w\|} \leq \frac{\|v^{\perp}\|}{\|v\|} = \frac{\langle v, v^{\perp} \rangle}{\|v\| \, \|v^{\perp}\|} = \cos\theta(v, v^{\perp}). \tag{A.13}$$

Since the consine function is monotonically decreasing on $[0, \pi]$, $\theta(v,w) \geq \theta(v,v^{\perp})$ as claimed.

□

Appendix B

Notations

B.1 Chapter 2: FEDALIGN

- $\mathbf{C}$, $\mathbf{C}_m$, $\overline{\mathbf{C}_m}$: $\mathbf{C}$ represents the universal set of classes, $\mathbf{C}_m$ is the set of classes identified on client m, and $\overline{\mathbf{C}_m} = \mathbf{C} \setminus \mathbf{C}_m$ is the set of classes not observed by client m.
- $\mathscr{X}$, $\mathscr{Y}$, $\mathscr{Z}$: the input space, the output space, and the latent feature space, respectively.
- $\mathscr{W}$: the set of natural language label names.
- $\mathbf{D}_m$: the local training dataset on client m.
- $\mathbf{D}_m'^{(t)}$: the augmented dataset for alignment at communication round t.
- x_i, y_i: an input sample and its binary label vector.
- $[y_{i,c}]_{c \in \mathbf{C}}$: the labels corresponding to each class c in $\mathbf{C}$.
- $\tilde{y}_{i,c}^{(t)}$: the pseudo-label for class c assigned at round t based on anchor-guided alignment.
- $f(\cdot;\theta)$: the data encoder parameterized by θ, mapping input data to latent space $\mathscr{Z}$.
- $h(\cdot;\zeta)$: the classifier parameterized by ζ, transforming latent features into prediction logits.
- $\gamma(\cdot;\omega)$: the label encoder parameterized by ω, mapping classes into latent space $\mathscr{Z}$.

- $g(\cdot;\theta,\zeta)$: The global classification model combining the data encoder and classifier to predict class probabilities.
- $g(x_i;\theta,\zeta)_c$: predicted probability that the input x_i belongs to class c.
- ℓ, ℓ': loss functions used for training (ℓ) and alignment (ℓ').
- M: the total number of clients.
- T: the total number of communication rounds.
- E: the number of local training epochs per client per round.
- $\mathbf{S}_t$: a subset of clients selected randomly in communication round t.
- $\theta_m^{(t)}$, $\omega_m^{(t)}$: the parameters of local data encoder (θ) and label encoder (ω) parameters for client m at round t.
- $d_{i,c}^{(t)}$: cosine distance between the data representation of sample x_i and class anchor c at round t.
- $\hat{\tau}_c^{(t)}$, $\check{\tau}_c^{(t)}$: thresholds for annotating positive and negative samples for class c based on distance from the anchor.
- η: learning rate for parameter updates during local training.
- $\mathscr{D}$, $\mathscr{D}_m$, $\tilde{\mathscr{D}}$, $\tilde{\mathscr{D}}_m$: $\mathscr{D}$ is the global data distribution, $\mathscr{D}_m$ is the local distribution at client m, and $\tilde{\mathscr{D}}$, $\tilde{\mathscr{D}}_m$ are the corresponding distributions in the latent feature space $\mathscr{Z}$.
- $\mathscr{D}'_m$, $\mathscr{D}^*_m$: $\mathscr{D}'_m$ is the augmented distribution for alignment on client m, and $\mathscr{D}^*_m$ is a mixture of $\mathscr{D}_m$ and $\mathscr{D}'_m$.
- α: weight factor controlling the mixture of original and augmented distributions.

- $\mathbf{H}$, h_m, h: $\mathbf{H}$ is the hypothesis class with VC-dimension d, h_m denotes the local hypothesis on client m, and h is the global hypothesis obtained by aggregating local models.
- $\mathscr{L}(h)$: expected risk (or loss) of the global hypothesis h over the global data distribution.
- $d_{\mathbf{H}\Delta\mathbf{H}}$: $\mathscr{A}$-distance measuring divergence between distributions with respect to the hypothesis class $\mathbf{H}$.
- λ_m: risk of the optimal hypothesis concerning the distributions $\mathscr{D}$ and $\mathscr{D}_m$ for client m.
- N, N^*: N is the number of observed samples on client m, and N^* represents the total number of samples after augmentation.

B.2 Chapter 3: REACT

- $\mathscr{P}(x)$, $\mathscr{P}(y|x)$: marginal feature distribution and conditional distribution of labels given features, respectively.
- x, y: input data sample and its corresponding category or label.
- $f(\cdot;\theta)$: threat detection model parameterized by θ.
- $\mathbf{D}$, $\mathbf{D}'$: original unlabeled dataset drawn from distribution $\mathscr{P}$, and a small dataset from distribution $\mathscr{P}'$.
- θ, θ': model parameters before and after adaptation to the new distribution.
- $\mathscr{L}$: loss function used for model optimization.
- $\mathscr{T}_i$, $\mathscr{P}_i$: a meta-learning task and its associated distribution.
- θ_i^*: optimal model parameters for task $\mathscr{T}_i$.
- $\mathscr{D}_i$: dataset of task $\mathscr{T}_i$.

- $\mathbf{D}^i_{\text{support}}$, $\mathbf{D}^i_{\text{query}}$: support set and query set used of task i in meta-learning.
- $h(\cdot;\phi)$: a hypernetwork parameterized by ϕ.
- $\mathbf{V}_i$: task representation of task i used by the hypernetwork.
- θ_{meta}, θ_{adapt}: meta weights and adaptive weights.
- $\mathbf{S}_t$: a subset of selected clients at round t in meta learning.
- c_i: contextual information associated with task $\mathscr{T}_i$.
- η_{meta}, η_h: learning rates for updating meta weights and hypernetwork parameters.
- λ: the regularization parameter for controlling adaptive weight norms.
- ϕ^*, θ^*: stationary points of the hypernetwork and meta weights after convergence.
- $\mathbf{X}^i$, $\mathbf{X}^i_s$, $\mathbf{X}^i_q$: data matrix for task i, and its corresponding support and query splits.
- σ_i: variance parameter for the dataset $\mathbf{X}^i$.
- ρ_{meta}, ρ_h: convergence rates for meta weights and hypernetwork during training.
- M: number of tasks sampled in each meta-training iteration.
- E: number of fine-tuning epochs on support data.
- k: the size of the support set for a given task.
- $|\mathbf{D}'|$: Number of observed samples from the new distribution for adaptation.
- t_1: run time for hypernetwork forward pass.
- t_2: run time for fine-tuning steps during adaptation.

B.3 Chapter 4: RECIPFL

- M: total number of clients in the federated learning system.
- $\mathscr{G}_m$: the customized model architecture for client m.
- $\mathbf{D}_m = \left\{ \left(x_i^{(m)}, y_i^{(m)} \right) \right\}_{i=1}^{N_m}$: the training dataset on client m, where $x_i^{(m)}$ is the input and $y_i^{(m)}$ is the corresponding label.
- $\mathscr{P}_m$: data distribution for client m.
- ℓ: loss function used for training the client models.
- $f_m(\cdot;\theta_m)$: the personalized model for client m parameterized by θ_m.
- $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_M\}$: the set of model parameters for all clients.
- $\mathbf{S}_t$: the subset of clients selected at communication round t.
- E: number of local training epochs for each client per communication round.
- η_c, η_s: learning rates for local client updates and server updates respectively.
- ϕ: parameters of the graph hypernetwork (GHN) on the server.
- $\tilde{\theta}_m$: model weights generated by the graph hypernetwork for client m.
- $\Delta\theta_m$: difference between updated client weights and initial weights generated by GHN.
- $\mathscr{G}(\mathscr{V}, \mathscr{E})$: the directed acyclic graph (DAG) representation of a neural network, where $\mathscr{V}$ is the set of nodes (operations) and $\mathscr{E}$ is the set of directed edges (computation flows).
- h_v: the feature vector for node v in the computational graph.
- l_v: the one-hot encoding of the operation type performed by node v.
- q_v: the parameters of the operator associated with node v.

- $m_v^{(t)}$: the message passed to node v at step t during graph propagation.
- a_m: the client descriptor for client m (e.g., class distribution or client ID embedding).
- f_m^S, f_m^L: the small and large models on strong device m, respectively.
- θ_m^S, θ_m^L: the parameters of the small and large models on strong device m.
- p_i^S, p_i^L: the softmax probabilities from the small and large models for input x_i.
- $\mathrm{CE}(\cdot)$: cross-entropy loss function.
- $\mathrm{D}_{KL}(\cdot\|\cdot)$: Kullback-Leibler divergence for aligning feature distributions.
- T: total number of communication rounds.
- $\mathscr{L}(\phi)$, $\hat{\mathscr{L}}(\phi, \mathbf{D})$: expected loss and empirical loss of the graph hypernetwork.
- N: number of training samples per client used in the generalization bound analysis.
- R: upper bound on the values of hypernetwork parameters ϕ.
- K: Lipschitz constant of the loss function ℓ with respect to ϕ.
- ε, δ: error margin and probability confidence used in generalization bounds.
- α: parameter for the Dirichlet distribution used to simulate non-IID data distributions.

B.4 Chapter 5: ORTHOFL

- M: the total number of clients participating in federated learning.
- W: the global model weights maintained by the server.
- W_m: the local model weights for client m.
- $\mathbf{D}_m$: the local dataset held by client m.

- $\mathscr{P}_m$: the data distribution corresponding to client m's dataset.
- ℓ: loss function used for optimization during training.
- $f(\cdot;W)$: the model parameterized by weights W.
- T: the total number of global communication rounds.
- m_t: index of the client that communicates with the server at round t.
- τ_t: the last global round when client m_t communicated with the server.
- $t-\tau$: number of rounds since the last communication from client m_t, i.e., staleness.
- β_t: delay-adaptive factor for aggregation, calculated as $\beta_t = (t-\tau)^{-a} \cdot \beta$.
- a: the hyperparameter controlling the decay factor of the staleness.
- $s_a(x)$: scaling function for the staleness factor, defined as $s_a(x) = x^{-a}$.
- ΔW: the global weight shift since the last update from client m, i.e., $W^{(t)} - W^{(\tau_+)}$.
- ΔW_m: the change in client m's local model weights from its last communication.
- ΔW^l, ΔW_m^l: the change in weights for layer l of the global model and client model, respectively.
- $\Delta W^{l\perp}$: the orthogonal component of global weight shift for layer l with respect to ΔW_m^l.
- E: number of local training epochs per communication round.
- η_c: the learning rate for local training updates on each client.
- $\mathscr{L}_m(W)$: the local objective function for client m based on its dataset.
- $\text{proj}_{\Delta W_m^l}(\Delta W^l)$: the projection of ΔW^l onto the direction of ΔW_m^l.

- $v^{\perp}$: the orthogonal component of vector v with respect to a subspace.
- $\langle v, w \rangle$: the inner product between vectors v and w.
- α: the parameter for the Dirichlet distribution to control data heterogeneity across clients.
- μ, σ: the mean and standard deviation used in the delay distributions for simulating client latency.
- $B_i^{(m)}$: the mini-batch i from client m's local dataset.

B.5 Chapter 6: STCOLAB

- l: a region defined as a geographic unit represented by a polygon in the coordinates.
- u: a user whose mobility data is being tracked.
- t: a specific time period in a day.
- (u, l, t): a daily mobility record entry representing user u's visit to region $l$during t.
- $\mathscr{S}_u$: the sequence of time-location pairs for user u.
- Q: the total number of records for user u.
- $\mathscr{D}$, $\mathscr{D}_{lb}$, $\mathscr{D}_{ul}$: the entire training dataset, its labeled subset and unlabeled subset, respectively.
- x, y: the input features corresponding to mobility data, and the ground-truth label.
- $\mathscr{C}$: the set of all possible classes for a demographic attribute.
- k: number of labeled samples available per class.
- a: the attribute class assigned to user u.
- $f_S^{(t)}$: the spatial model at iteration t.

- $f_T^{(t)}$: the temporal model at iteration t.
- $\tilde{\mathscr{D}}_S^{(t)}$, $\tilde{\mathscr{D}}_T^{(t)}$: the distillation dataset at iteration t for the spatial and the temporal model, respectively.
- r: the percentile score threshold for selecting confident predictions.
- $\mathbb{T}_a^{(t)}$: the confidence threshold for class $a$at iteration t.
- M: the one-channel image representing the visualized map of mobility data.
- z: the spatial feature vector learned from the map.
- p_i^{spatial}, p_i^{temporal}: the prediction probability for user u_i given by the spatial and the temporal module, respectively.
- $\mathscr{L}_S$, $\mathscr{L}_T$: the loss function for training the spatial and the temporal module, respectively.
- h: the window size for temporal convolution.
- d: the dimension of spatial feature vectors.
- e_j: the temporal feature extracted from a window of size h.
- N_f: number of filters in the temporal convolution.
- N_{est}: number of samples used for estimating class distribution.
- $\mathscr{L}_T^{(t)}$: the distillation loss for the temporal model at iteration t.
- $\tilde{\mathscr{D}}_{T_{ul}}^{(t)}$: the selected unlabeled distillation dataset for the temporal model at iteration t.
- $\tilde{\mathscr{D}}_{T_{lb}}^{(t)}$: the balanced labeled dataset used in distillation at iteration t.
- $\hat{y}_{i,a}^S$, $\hat{y}_{i,a}^T$: the predicted label (1 if true, 0 otherwise) for class a at iteration t given by the spatial and the temporal model, respectively.
- $\hat{N}_a$: mumber of samples in $\tilde{\mathscr{D}}_{T_{ul}}^{(t)}$predicted as class a.

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Charu C Aggarwal and Charu C Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[3] Yuval Alaluf, Omer Tov, Ron Mokady, Rinon Gal, and Amit Bermano. Hyperstyle: Stylegan inversion with hypernetworks for real image editing. In *Proceedings of the IEEE/CVF conference on computer Vision and pattern recognition*, pages 18511–18521, 2022.

[4] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. FedRolex: Model-Heterogeneous Federated Learning with Rolling Sub-Model Extraction. *Advances in Neural Information Processing Systems*, 35:29677–29690, 2022.

[5] David Alvarez-Melis and Nicolo Fusi. Geometric dataset distances via optimal transport. *Advances in Neural Information Processing Systems*, 33:21428–21439, 2020.

[6] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3, 2013.

[7] Eric Arazo, Diego Ortego, Paul Albert, Noel E O'Connor, and Kevin McGuinness. Pseudo-Labeling and Confirmation Bias in Deep Semi-Supervised Learning. In *2020 International Joint Conference on Neural Networks*, pages 1–8, 2020.

[8] Dmitrii Avdiukhin and Shiva Kasiviswanathan. Federated learning under arbitrary communication patterns. In *International Conference on Machine Learning*, pages 425–435. PMLR, 2021.

[9] Rotem Zamir Aviv, Ido Hakimi, Assaf Schuster, and Kfir Y Levy. Learning under delayed feedback: Implicitly adapting to gradient delays. *arXiv preprint arXiv:2106.12261*, 2021.

[10] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances In Neural Information Processing Systems*, 27, 2014.

[11] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

[12] Jonathan Baxter. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[13] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79:151–175, 2010.

[14] Sidahmed Benabderrahmane, Ghita Berrada, James Cheney, and Petko Valtchev. A rule mining-based advanced persistent threats detection system. *arXiv preprint arXiv:2105.10053*, 2021.

[15] Liron Bergman and Yedid Hoshen. Classification-based anomaly detection for general data. *arXiv preprint arXiv:2005.02359*, 2020.

[16] Bin Bi, Milad Shokouhi, Michal Kosinski, and Thore Graepel. Inferring the Demographics of Search Users: Social Data Meets Search Queries. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 131–140, 2013.

[17] David Bonet, Daniel Mas Montserrat, Xavier Giró-i Nieto, and Alexander G Ioannidis. Hyperfast: Instant classification for tabular data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11114–11123, 2024.

[18] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[19] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. *NeurIPS*, 33:1877–1901, 2020.

[20] John A. Bullinaria and Joseph P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526, 2007.

[21] Ana Cardoso-Cachopo. Improving Methods for Single-label Text Categorization. PdD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[22] João Carvalho, Mengtao Zhang, Robin Geyer, Carlos Cotrini, and Joachim M Buhmann. Invariant anomaly detection under distribution shifts: a causal perspective. *Advances in Neural Information Processing Systems*, 36, 2024.

[23] Paola Cascante-Bonilla, Fuwen Tan, Yanjun Qi, and Vicente Ordonez. Curriculum Labeling: Revisiting Pseudo-Labeling for Semi-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6912–6920, 2021.

[24] Sumohana Channappayya, Bheemarjuna Reddy Tamma, et al. Augmented memory replay-based continual learning approaches for network intrusion detection. *Advances in Neural Information Processing Systems*, 36, 2024.

[25] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[26] Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. A brief review of hypernetworks in deep learning. *Artificial Intelligence Review*, 57(9):250, 2024.

[27] Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. Recall and learn: Fine-tuning deep pretrained language models with less forgetting. *arXiv preprint arXiv:2004.12651*, 2020.

[28] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014.

[29] Alon Cohen, Amit Daniely, Yoel Drori, Tomer Koren, and Mariano Schain. Asynchronous stochastic optimization robust to arbitrary delays. *Advances in Neural Information Processing Systems*, 34:9024–9035, 2021.

[30] Aron Culotta, Nirmal Kumar, and Jennifer Cutler. Predicting the Demographics of Twitter Users from Website Traffic Data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[31] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[33] Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients. *arXiv preprint arXiv:2010.01264*, 2020.

[34] Hua Ding, Lixing Chen, Shenghong Li, Yang Bai, Pan Zhou, and Zhe Qu. Divide, conquer, and coalesce: Meta parallel graph neural network for iot intrusion detection at scale. In *Proceedings of the ACM on Web Conference 2024*, pages 1656–1667, 2024.

[35] Kaize Ding, Qinghai Zhou, Hanghang Tong, and Huan Liu. Few-shot network anomaly detection via cross-network meta-learning. In *Proceedings of the Web Conference 2021*, pages 2448–2456, 2021.

[36] Jiahua Dong, Lixu Wang, Zhen Fang, Gan Sun, Shichao Xu, Xiao Wang, and Qi Zhu. Federated class-incremental learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10164–10173, 2022.

[37] Marius Dragoi, Elena Burceanu, Emanuela Haller, Andrei Manolache, and Florin Brad. Anoshift: A distribution shift benchmark for unsupervised anomaly detection. *Advances in Neural Information Processing Systems*, 35:32854–32867, 2022.

[38] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International conference on artificial intelligence and statistics*, pages 803–812. PMLR, 2018.

[39] Xiuwen Fang and Mang Ye. Robust Federated Learning with Noisy and Heterogeneous Clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10072–10081, 2022.

[40] Zhihan Fang, Yu Yang, Guang Yang, Yikuan Xian, Fan Zhang, and Desheng Zhang. CellSense: Human Mobility Recovery via Cellular Network Data Enhancement. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3):1–22, 2021.

[41] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R Arabnia. A brief review of domain adaptation. *Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020*, pages 877–894, 2021.

[42] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR, 2020.

[43] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[44] Ahmed Frikha, Denis Krompaß, Hans-Georg Köpken, and Volker Tresp. Few-shot one-class classification via meta-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7448–7456, 2021.

[45] Itai Gat, Idan Schwartz, Alexander Schwing, and Tamir Hazan. Removing Bias in Multi-modal Classifiers: Regularization by Maximizing Functional Entropies. *Advances in Neural Information Processing Systems*, 33:3197–3208, 2020.

[46] Efstathios D. Gennatas, Jerome H. Friedman, Lyle H. Ungar, Romain Pirracchio, Eric Eaton, Lara G. Reichmann, Yannet Interian, José Marcio Luna, Charles B. Simone, Andrew Auerbach, et al. Expert-augmented machine learning. *PNAS*, 117(9):4571–4577, 2020.

[47] Isaac Gibbs and Emmanuel Candes. Adaptive conformal inference under distribution shift. *Advances in Neural Information Processing Systems*, 34:1660–1672, 2021.

[48] Qingyuan Gong, Yushan Liu, Jiayun Zhang, Yang Chen, Qi Li, Yu Xiao, Xin Wang, and Pan Hui. Detecting malicious accounts in online developer communities using deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(10):10633–10649, 2023.

[49] Adam Goodge, Bryan Hooi, See-Kiong Ng, and Wee Siong Ng. Lunar: Unifying local outlier detection methods via graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6737–6745, 2022.

[50] Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. Covariate shift by kernel mean matching. 2008.

[51] Xinran Gu, Kaixuan Huang, Jingzhao Zhang, and Longbo Huang. Fast federated learning in the presence of arbitrary device unavailability. *Advances in Neural Information Processing Systems*, 34:12052–12064, 2021.

[52] Shurui Gui, Xiner Li, and Shuiwang Ji. Active test-time adaptation: Theoretical analyses and an algorithm. *arXiv preprint arXiv:2404.05094*, 2024.

[53] Yue Guo, Chenxi Hu, and Yi Yang. Predict the future from the past? on the temporal data distribution shift in financial sentiment classifications. *arXiv preprint arXiv:2310.12620*, 2023.

[54] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2017.

[55] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[56] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[57] Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. news-please: a Generic News Crawler and Extractor. In *International Symposium of Information Science*, pages 218–223, 2017.

[58] Dongqi Han, Zhiliang Wang, Wenqi Chen, Kai Wang, Rui Yu, Su Wang, Han Zhang, Zhihua Wang, Minghui Jin, Jiahai Yang, et al. Anomaly detection in the open world: Normality shift detection, explanation, and adaptation. In *NDSS*, 2023.

[59] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. Adbench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems*, 35:32142–32159, 2022.

[60] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognitio*, pages 770–778, 2016.

[62] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[63] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In *ICML*, pages 2790–2799. PMLR, 2019.

[64] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification. *arXiv preprint arXiv:1909.06335*, 2019.

[65] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[66] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[67] Zhongzhan Huang, Mingfu Liang, Shanshan Zhong, and Liang Lin. Attns: Attention-inspired numerical solving for limited data scenarios. In *Forty-first International Conference on Machine Learning*.

[68] Ngoc Anh Huynh, Wee Keong Ng, and Kanishka Ariyapala. A new adaptive learning algorithm and its application to online malware detection. In *Discovery Science: 20th International Conference, DS 2017, Kyoto, Japan, October 15–17, 2017, Proceedings 20*, pages 18–32. Springer, 2017.

[69] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. Oobleck: Resilient distributed training of large models using pipeline templates. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 382–395, 2023.

[70] Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 186(1007):453–461, 1946.

[71] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, 2019.

[72] Xiayan Ji, Hyonyoung Choi, Oleg Sokolsky, and Insup Lee. Incremental anomaly detection with guarantee in the internet of medical things. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, pages 327–339, 2023.

[73] Peng Jia, Shaofeng Cai, Beng Chin Ooi, Pinghui Wang, and Yiyuan Xiong. Robust and transferable log-based anomaly detection. *Proceedings of the ACM on Management of Data*, 1(1):1–26, 2023.

[74] Zhida Jiang, Yang Xu, Hongli Xu, Zhiyuan Wang, Chunming Qiao, and Yangming Zhao. FedMP: Federated Learning through Adaptive Model Pruning in Heterogeneous Edge Computing. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 767–779. IEEE, 2022.

[75] Alistair EW Johnson, Tom J. Pollard, Lu Shen, Li-wei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(1):1–9, 2016.

[76] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Adrian Garcia Badaracco, Carson Farmer, et al. geopandas/geopandas: v0. 8.1. *Zenodo*, 2020.

[77] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX security symposium (USENIX security 17)*, pages 625–642, 2017.

[78] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[79] Denizhan Kara, Tomoyoshi Kimura, Shengzhong Liu, Jinyang Li, Dongxin Liu, Tianshi Wang, Ruijie Wang, Yizhuo Chen, Yigong Hu, and Tarek Abdelzaher. Freqmae: Frequency-aware masked autoencoder for multi-modal iot sensing. In *Proceedings of the ACM on Web Conference 2024*, pages 2795–2806, 2024.

[80] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *ICML*, pages 5132–5143. PMLR, 2020.

[81] Dongmin Kim, Sunghyun Park, and Jaegul Choo. When model meets new normals: Test-time adaptation for unsupervised time-series anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13113–13121, 2024.

[82] Minjae Kim, Sangyoon Yu, Suhyun Kim, and Soo-Mook Moon. DepthFL: Depthwise Federated Learning for Heterogeneous Clients. In *The Eleventh International Conference on Learning Representations*, 2023.

[83] Raehyun Kim, Hyunjae Kim, Janghyuk Lee, and Jaewoo Kang. Predicting Multiple Demographic Attributes with Task Specific Embedding Transformation and Attention Network. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 765–773, 2019.

[84] Leo Klarner, Tim GJ Rudner, Michael Reutlinger, Torsten Schindler, Garrett M Morris, Charlotte Deane, and Yee Whye Teh. Drug discovery under covariate shift with domain-informed prior distributions over functions. In *International Conference on Machine Learning*, pages 17176–17197. PMLR, 2023.

[85] Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. Parameter Prediction for Unseen Deep Architectures. *Advances in Neural Information Processing Systems*, 34:29433–29448, 2021.

[86] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese Neural Networks For One-Shot Image Recognition. In *ICML deep learning workshop*, volume 2, 2015.

[87] Anastasiia Koloskova, Sebastian U Stich, and Martin Jaggi. Sharper convergence guarantees for asynchronous sgd for distributed and federated learning. *Advances in Neural Information Processing Systems*, 35:17202–17215, 2022.

[88] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated Optimization: Distributed Optimization Beyond the Datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[89] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[90] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. 2009.

[91] Sean Kulinski, Saurabh Bagchi, and David I Inouye. Feature shift detection: Localizing which features have shifted via conditional distribution tests. *Advances in neural information processing systems*, 33:19523–19533, 2020.

[92] Atsutoshi Kumagai, Tomoharu Iwata, Hiroshi Takahashi, and Yasuhiro Fujiwara. Meta-learning for robust anomaly detection. In *International Conference on Artificial Intelligence and Statistics*, pages 675–691. PMLR, 2023.

[93] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient Federated Learning via Guided Participant Selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pages 19–35, 2021.

[94] Ken Lang. Newsweeder: Learning to filter netnews. In *Machine learning proceedings 1995*, pages 331–339. Elsevier, 1995.

[95] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[96] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE, 2010.

[97] Chung-Wei Lee, Wei Fang, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. Multi-Label Zero-Shot Learning with Structured Knowledge Graphs. In *CVPR*, pages 1576–1585, 2018.

[98] Dong-Hyun Lee et al. Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 896, 2013.

[99] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[100] Jimmy Lei Ba, Kevin Swersky, Sanja Fidler, et al. Predicting Deep Zero-Shot Convolutional Neural Networks using Textual Descriptions. In *ICCV*, pages 4247–4255, 2015.

[101] Omer Levy and Yoav Goldberg. Linguistic Regularities in Sparse and Explicit Word Representations. In *CoNLL*, pages 171–180, 2014.

[102] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. Hermes: An Efficient Federated Learning Framework for Heterogeneous Mobile Clients. In *MobiCom*, pages 420–437, 2021.

[103] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. FedMask: Joint Computation and Communication-Efficient Personalized Federated Learning via Heterogeneous Masking. In *SenSys*, pages 42–55, 2021.

[104] Aodong Li, Chen Qiu, Marius Kloft, Padhraic Smyth, Maja Rudolph, and Stephan Mandt. Zero-shot anomaly detection via batch normalization. *Advances in Neural Information Processing Systems*, 36, 2024.

[105] Chenglin Li, Di Niu, Bei Jiang, Xiao Zuo, and Jianming Yang. Meta-HAR: Federated Representation Learning for Human Activity Recognition. In *WWW*, pages 912–922, 2021.

[106] Heng Li, Shiyao Zhou, Wei Yuan, Xiapu Luo, Cuiying Gao, and Shuiyan Chen. Robust android malware detection against adversarial example attacks. In *Proceedings of the Web Conference 2021*, pages 3603–3612, 2021.

[107] Hongyu Li, Liang Ding, Meng Fang, and Dacheng Tao. Revisiting catastrophic forgetting in large language model tuning. *arXiv preprint arXiv:2406.04836*, 2024.

[108] Liang Li, Dian Shi, Ronghui Hou, Hui Li, Miao Pan, and Zhu Han. To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices. In *INFOCOM*, pages 1–10. IEEE, 2021.

[109] Qinbin Li, Bingsheng He, and Dawn Song. Model-Contrastive Federated Learning. In *CVPR*, pages 10713–10722, 2021.

[110] Shuheng Li, Jiayun Zhang, Xiaohan Fu, Xiyuan Zhang, Jingbo Shang, and Rajesh K. Gupta. Matching skeleton-based activity representations with heterogeneous signals for har, 2025.

[111] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated Optimization in Heterogeneous Networks. *MLSys*, 2:429–450, 2020.

[112] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the Convergence of FedAvg on Non-IID Data. In *ICLR*, 2019.

[113] Xin-Chun Li and De-Chuan Zhan. FedRS: Federated Learning with Restricted Softmax for Label Distribution Non-IID Data. In *KDD*, pages 995–1005, 2021.

[114] Xinzhe Li, Qianru Sun, Yaoyao Liu, Qin Zhou, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to Self-Train for Semi-Supervised Few-Shot Classification. *Advances in Neural Information Processing Systems*, 32:10276–10286, 2019.

[115] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[116] Yuxin Li, Wenchao Chen, Bo Chen, Dongsheng Wang, Long Tian, and Mingyuan Zhou. Prototype-oriented unsupervised anomaly detection for multivariate time series. In *International Conference on Machine Learning*, pages 19407–19424. PMLR, 2023.

[117] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. Copod: copula-based outlier detection. In *2020 IEEE international conference on data mining (ICDM)*, pages 1118–1123. IEEE, 2020.

[118] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H Chen. Ecod: Unsupervised outlier detection using empirical cumulative distribution functions. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12181–12193, 2022.

[119] Feng Liang, Weike Pan, and Zhong Ming. FedRec++: Lossless Federated Recommendation with Explicit Feedback. In *AAAI*, volume 35, pages 4224–4231, 2021.

[120] Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Ensemble Distillation for Robust Model Fusion in Federated Learning. *NeurIPS*, 33, 2020.

[121] Xinyang Lin, Hanting Chen, Yixing Xu, Chao Xu, Xiaolin Gui, Yiping Deng, and Yunhe Wang. Federated Learning with Positive and Unlabeled Data. In *ICML*, pages 13344–13355. PMLR, 2022.

[122] Or Litany, Haggai Maron, David Acuna, Jan Kautz, Gal Chechik, and Sanja Fidler. Federated Learning with Heterogeneous Architectures using Graph HyperNetworks. *arXiv preprint arXiv:2201.08459*, 2022.

[123] Boyi Liu, Yiming Ma, Zimu Zhou, Yexuan Shi, Shuyuan Li, and Yongxin Tong. Casa: Clustered federated learning with asynchronous clients. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1851–1862, 2024.

[124] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[125] Ji Liu, Juncheng Jia, Tianshi Che, Chao Huo, Jiaxiang Ren, Yang Zhou, Huaiyu Dai, and Dejing Dou. Fedasmu: Efficient asynchronous federated learning with dynamic staleness-aware model update. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13900–13908, 2024.

[126] Liu Liu, Olivier De Vel, Chao Chen, Jun Zhang, and Yang Xiang. Anomaly-based insider threat detection using deep autoencoders. In *2018 IEEE international conference on data mining workshops (ICDMW)*, pages 39–48. IEEE, 2018.

[127] Ruixuan Liu, Fangzhao Wu, Chuhan Wu, Yanlin Wang, Lingjuan Lyu, Hong Chen, and Xing Xie. No One Left Behind: Inclusive Federated Learning over Heterogeneous Devices. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3398–3406, 2022.

[128] Shuchang Liu, Shuyuan Xu, Wenhui Yu, Zuohui Fu, Yongfeng Zhang, and Amelie Marian. FedCT: Federated Collaborative Transfer for Recommendation. In *SIGIR*, pages 716–725, 2021.

[129] Xiaofeng Liu, Chaehwa Yoo, Fangxu Xing, Hyejin Oh, Georges El Fakhri, Je-Won Kang, Jonghye Woo, et al. Deep unsupervised domain adaptation: A review of recent advances and perspectives. *APSIPA Transactions on Signal and Information Processing*, 11(1), 2022.

[130] Yanchi Liu, Chuanren Liu, Xinjiang Lu, Mingfei Teng, Hengshu Zhu, and Hui Xiong. Point-of-Interest Demand Modeling with Human Mobility Patterns. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 947–955, 2017.

[131] Feixiong Luo, Guofeng Cao, Kevin Mulligan, and Xiang Li. Explore spatiotemporal and demographic characteristics of human mobility via Twitter: A case study of Chicago. *Applied Geography*, 70:11–25, 2016.

[132] Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No Fear of Heterogeneity: Classifier Calibration for Federated Learning with Non-IID Data. *NeurIPS*, 34, 2021.

[133] Yuhang Ma, Zhongle Xie, Jue Wang, Ke Chen, and Lidan Shou. Continual federated learning based on knowledge distillation. In *IJCAI*, pages 2182–2188, 2022.

[134] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

[135] Yuren Mao, Yaobo Liang, Nan Duan, Haobo Wang, Kai Wang, Lu Chen, and Yunjun Gao. Less-forgetting multi-lingual fine-tuning. *Advances in Neural Information Processing Systems*, 35:14917–14928, 2022.

[136] Moe Matsuki, Paula Lago, and Sozo Inoue. Characterizing Word Embeddings for Zero-Shot Sensor-Based Human Activity Recognition. *Sensors*, 19(22):5043, 2019.

[137] Geoffrey J McLachlan. Iterative Reclassification Procedure for Constructing an Asymptotically Optimal Rule of Allocation in Discriminant Analysis. *Journal of the American Statistical Association*, 70(350):365–369, 1975.

[138] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, pages 1273–1282. PMLR, 2017.

[139] Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. Deep Keyphrase Generation. *arXiv preprint arXiv:1704.06879*, 2017.

[140] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. *NeurIPS*, 26, 2013.

[141] Konstantin Mishchenko, Francis Bach, Mathieu Even, and Blake E Woodworth. Asynchronous sgd beats minibatch sgd under arbitrary delays. *Advances in Neural Information Processing Systems*, 35:420–433, 2022.

[142] Konstantin Mishchenko, Franck Iutzeler, Jérôme Malick, and Massih-Reza Amini. A delay-tolerant proximal-gradient algorithm for distributed learning. In *International conference on machine learning*, pages 3587–3595. PMLR, 2018.

[143] Omar Montasser and Daniel Kifer. Predicting Demographics of High-Resolution Geographies with Geotagged Tweets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[144] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable Prediction of Medical Codes from Clinical Text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 1101–1111, 2018.

[145] Yujin Nam, Abhishek Moitra, Yeshwanth Venkatesha, Xiaofan Yu, Gabrielle De Micheli, Xuan Wang, Minxuan Zhou, Augusto Vega, Priyadarshini Panda, and Tajana Rosing. Rhychee-fl: Robust and efficient hyperdimensional federated learning with homomorphic encryption. 2025.

[146] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pages 3581–3607. PMLR, 2022.

[147] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. Billion-Scale Federated Learning on Mobile Clients: A Submodel Design with Tunable Privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[148] Bethesda (MD): National Library of Medicine. Pmc open access subset [internet], 2003. [accessed 7-June-2022].

[149] Xiaomin Ouyang, Zhiyuan Xie, Heming Fu, Sitong Cheng, Li Pan, Neiwen Ling, Guoliang Xing, Jiayu Zhou, and Jianwei Huang. Harmony: Heterogeneous multi-modal federated learning through disentangled model training. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, pages 530–543, 2023.

[150] Tae Jin Park, Kenichi Kumatani, and Dimitrios Dimitriadis. Tackling dynamics in federated incremental learning with variational embedding rehearsal. *arXiv preprint arXiv:2110.09695*, 2021.

[151] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX security symposium (USENIX Security 19)*, pages 729–746, 2019.

[152] Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. Federated Adversarial Domain Adaptation. *arXiv preprint arXiv:1911.02054*, 2019.

[153] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *EMNLP*, pages 1532–1543, 2014.

[154] Tomáš Pevnỳ. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102:275–304, 2016.

[155] Yinhua Piao, Sangseon Lee, Dohoon Lee, and Sun Kim. Sparse Structure Learning via Graph Neural Networks for Inductive Document Classifcation. In *AAAI*, volume 36, pages 11165–11173, 2022.

[156] Daniele Quercia, Neal Lathia, Francesco Calabrese, Giusy Di Lorenzo, and Jon Crowcroft. Recommending Social Events from Mobile Phone Location Data. In *2010 IEEE International Conference on Data Mining*, pages 971–976, 2010.

[157] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. *Advances in Neural Information Processing Systems*, 32, 2019.

[158] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, pages 8748–8763. PMLR, 2021.

[159] 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*, pages 427–438, 2000.

[160] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[161] Attila Reiss and Didier Stricker. Introducing a New Benchmarked Dataset for Activity Monitoring. In *ISWC*, pages 108–109. IEEE, 2012.

[162] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[163] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018.

[164] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[165] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[166] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[167] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and Metrics for Cold-Start Recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–260, 2002.

[168] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[169] Jessica Schrouff, Natalie Harris, Sanmi Koyejo, Ibrahim M Alabdulmohsin, Eva Schnider, Krista Opsahl-Ong, Alexander Brown, Subhrajit Roy, Diana Mincu, Christina Chen, et al.

Diagnosing failures of fairness transfer across distribution shift in real-world medical settings. *Advances in Neural Information Processing Systems*, 35:19304–19318, 2022.

[170] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. Personalized Federated Learning using Hypernetworks. In *International Conference on Machine Learning*, pages 9489–9502. PMLR, 2021.

[171] Jin Shang, Mingxuan Sun, and Kevyn Collins-Thompson. Demographic Inference Via Knowledge Transfer in Cross-Domain Recommender Systems. In *2018 IEEE International Conference on Data Mining*, pages 1218–1223, 2018.

[172] Kendrick Shen, Robbie M Jones, Ananya Kumar, Sang Michael Xie, Jeff Z HaoChen, Tengyu Ma, and Percy Liang. Connect, not collapse: Explaining contrastive learning for unsupervised domain adaptation. In *International conference on machine learning*, pages 19847–19878. PMLR, 2022.

[173] Ting Shen, Haiquan Chen, and Wei-Shinn Ku. Time-aware Location Sequence Recommendation for Cold-start Mobile Users. In *Proceedings of 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 484–487, 2018.

[174] Weiwei Shi, Yihong Gong, Chris Ding, Zhiheng MaXiaoyu Tao, and Nanning Zheng. Transductive Semi-Supervised Deep Learning using Min-Max Features. In *Proceedings of the European Conference on Computer Vision*, pages 299–315, 2018.

[175] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Advances in Neural Information Processing Systems*, 28, 2015.

[176] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[177] Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication. *arXiv preprint arXiv:1909.05350*, 2019.

[178] Petar Stojanov, Zijian Li, Mingming Gong, Ruichu Cai, Jaime Carbonell, and Kun Zhang. Domain adaptation with invariant representation learning: What transformations to learn? *Advances in Neural Information Processing Systems*, 34:24791–24803, 2021.

[179] Ningxin Su and Baochun Li. How asynchronous can federated learning be? In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, pages 1–11. IEEE, 2022.

[180] Kaixin Sui, Mengyu Zhou, Dapeng Liu, Minghua Ma, Dan Pei, Youjian Zhao, Zimu Li, and Thomas Moscibroda. Characterizing and improving wifi latency in large-scale operational networks. In *MobiSys*, pages 347–360, 2016.

[181] Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. All in one: Multi-task prompting for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2120–2131, 2023.

[182] Xiangguo Sun, Hongzhi Yin, Bo Liu, Hongxu Chen, Jiuxin Cao, Yingxia Shao, and Nguyen Quoc Viet Hung. Heterogeneous hypergraph embedding for graph classification. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 725–733, 2021.

[183] Zachariah Sutton, Peter Willett, and Yaakov Bar-Shalom. Modeling and detection of evolving threats using random finite set statistics. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4319–4323. IEEE, 2018.

[184] Korawat Tanwisuth, Xinjie Fan, Huangjie Zheng, Shujian Zhang, Hao Zhang, Bo Chen, and Mingyuan Zhou. A prototype-oriented framework for unsupervised domain adaptation. *Advances in Neural Information Processing Systems*, 34:17194–17208, 2021.

[185] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee, 2009.

[186] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[187] Chunlin Tian, Li Li, Zhan Shi, Jun Wang, and ChengZhong Xu. HARMONY: Heterogeneity-Aware Hierarchical Management for Federated Learning System. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 631–645. IEEE, 2022.

[188] Cheng-Hao Tu, Hong-You Chen, Zheda Mai, Jike Zhong, Vardaan Pahuja, Tanya Berger-Wolf, Song Gao, Charles Stewart, Yu Su, and Wei-Lun Harry Chao. Holistic transfer: towards non-disruptive fine-tuning with partial target data. *Advances in Neural Information Processing Systems*, 36, 2024.

[189] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. FedDL: Federated Learning via Dynamic Layer Sharing for Human Activity Recognition. In *SenSys*, pages 15–28, 2021.

[190] Yonatan Vaizman, Katherine Ellis, and Gert Lanckriet. Recognizing Detailed Human Context in the Wild from Smartphones and Smartwatches. *IEEE Pervasive Computing*, 16(4):62–74, 2017.

[191] Laurens Van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

[192] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *NeurIPS*, 30, 2017.

[193] Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

[194] Laura Von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, et al. Informed machine learning—a taxonomy and survey of integrating prior knowledge into learning systems. *TKDE*, 35(1):614–633, 2021.

[195] Chen Wang, Ziwei Fan, Liangwei Yang, Mingdai Yang, Xiaolong Liu, Zhiwei Liu, and Philip Yu. Pre-training with transferable attention for addressing market shifts in cross-market sequential recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2970–2979, 2024.

[196] Daheng Wang, Meng Jiang, Munira Syed, Oliver Conway, Vishal Juneja, Sriram Subramanian, and Nitesh V. Chawla. Calendar graph neural networks for modeling time structures in spatiotemporal user behaviors. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2581–2589, 2020.

[197] Hao Wang. Improving neural network generalization on data-limited regression with doubly-robust boosting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20821–20829, 2024.

[198] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated Learning with Matched Averaging. In *ICLR*, 2020.

[199] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. *NeurIPS*, 33:7611–7623, 2020.

[200] Kaibin Wang, Qiang He, Feifei Chen, Chunyang Chen, Faliang Huang, Hai Jin, and Yun Yang. FlexiFed: Personalized Federated Learning for Edge Clients with Heterogeneous Model Architectures. In *Proceedings of the ACM Web Conference 2023*, pages 2979–2990, 2023.

[201] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. Your Cart tells You: Inferring Demographic Attributes from Purchase Data. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 173–182, 2016.

[202] Pinghui Wang, Feiyang Sun, Di Wang, Jing Tao, Xiaohong Guan, and Albert Bifet. Inferring Demographics and Social Networks of Mobile Device Users on Campus From AP-Trajectories. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 139–147, 2017.

[203] Qinyong Wang, Hongzhi Yin, Tong Chen, Zi Huang, Hao Wang, Yanchang Zhao, and Nguyen Quoc Viet Hung. Next Point-of-Interest Recommendation on Resource-Constrained Mobile Devices. In *Proceedings of the Web Conference*, pages 906–916, 2020.

[204] Yujia Wang, Yuanpu Cao, Jingcheng Wu, Ruoyu Chen, and Jinghui Chen. Tackling the data heterogeneity in asynchronous federated learning with cached update calibration. In *International Conference on Learning Representations*, 2024.

[205] Ze Wang, Yipin Zhou, Rui Wang, Tsung-Yu Lin, Ashish Shah, and Ser Nam Lim. Few-shot fast-adaptive anomaly detection. *Advances in Neural Information Processing Systems*, 35:4957–4970, 2022.

[206] Zijian Wang, Scott Hale, David Ifeoluwa Adelani, Przemyslaw Grabowicz, Timo Hartman, Fabian Flöck, and David Jurgens. Demographic Inference and Representative Population Estimates from Multilingual Social Media Data. In *Proceedings of the 2019 World Wide Web Conference*, pages 2056–2067, 2019.

[207] A Waswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A Gomez, L Kaiser, and I Polosukhin. Attention is all you need. In *NIPS*, 2017.

[208] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, 2022.

[209] Adina Williams, Nikita Nangia, and Samuel R Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *arXiv preprint arXiv:1704.05426*, 2017.

[210] Garrett Wilson and Diane J Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(5):1–46, 2020.

[211] Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning*, pages 11217–11227. PMLR, 2021.

[212] Tong Wu, Yiqiang Chen, Yang Gu, Jiwei Wang, Siyu Zhang, and Zhanghu Zhechen. Multi-Layer Cross Loss Model for Zero-Shot Human Activity Recognition. In *PAKDD*, pages 210–221, 2020.

[213] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[214] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Zeno++: Robust fully asynchronous sgd. In *International Conference on Machine Learning*, pages 10495–10503. PMLR, 2020.

[215] Fengli Xu, Zongyu Lin, Tong Xia, Diansheng Guo, and Yong Li. SUME: Semantic-enhanced Urban Mobility Network Embedding for User Demographic Inference. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(3):1–25, 2020.

[216] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing Impacts of Heterogeneity in Federated Learning upon Large-Scale Smartphone Data. In *WWW*, pages 935–946, 2021.

[217] Shuo Yang, Xinran Zheng, Jinze Li, Jinfeng Xu, Xingjun Wang, and Edith CH Ngai. Recda: Concept drift adaptation with representation enhancement for network intrusion detection. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3818–3828, 2024.

[218] Xin Yang, Hao Yu, Xin Gao, Hao Wang, Junbo Zhang, and Tianrui Li. Federated continual learning via knowledge fusion: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[219] Jingwei Yi, Fangzhao Wu, Chuhan Wu, Ruixuan Liu, Guangzhong Sun, and Xing Xie. Efficient-FedRec: Efficient Federated Learning Framework for Privacy-Preserving News Recommendation. In *EMNLP*, pages 2814–2824, 2021.

[220] Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *International Conference on Machine Learning*, pages 12073–12086. PMLR, 2021.

[221] Stella X. Yu and Jianbo Shi. Multiclass Spectral Clustering. In *ICCV*, pages 313–319. IEEE, 2003.

[222] Xiaofan Yu, Lucy Cherkasova, Harsh Vardhan, Quanling Zhao, Emily Ekaireb, Xiyuan Zhang, Arya Mazumdar, and Tajana Rosing. Async-hfl: Efficient and robust asynchronous federated learning in hierarchical iot networks. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, pages 236–248, 2023.

[223] Xiaofan Yu, Lanxiang Hu, Benjamin Reichman, Rushil Chandrupatla, Dylan Chu, Xiyuan Zhang, Larry Heck, and Tajana S Rosing. A real time question answering system for multimodal sensors using llms. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 836–837, 2024.

[224] Shuhan Yuan, Panpan Zheng, Xintao Wu, and Hanghang Tong. Few-shot insider threat detection. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2289–2292, 2020.

[225] Xingdi Yuan, Tong Wang, Rui Meng, Khushboo Thaker, Peter Brusilovsky, Daqing He, and Adam Trischler. One Size Does Not Fit All: Generating and Evaluating Variable Number of Keyphrases. *arXiv preprint arXiv:1810.05241*, 2018.

[226] Yu Zang, Zhe Xue, Shilong Ou, Lingyang Chu, Junping Du, and Yunfei Long. Efficient asynchronous federated learning with prospective momentum aggregation and fine-grained correction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16642–16650, 2024.

[227] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph HyperNetworks for Neural Architecture Search. *arXiv preprint arXiv:1810.05749*, 2018.

[228] Feilong Zhang, Xianming Liu, Shiyi Lin, Gang Wu, Xiong Zhou, Junjun Jiang, and Xiangyang Ji. No one idles: Efficient heterogeneous federated learning with parallel edge and server computation. In *International Conference on Machine Learning*, pages 41399–41413. PMLR, 2023.

[229] Hai Zhang, Chunwei Wu, Guitao Cao, Hailing Wang, and Wenming Cao. Hypereditor: Achieving both authenticity and cross-domain capability in image editing via hypernetworks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7051–7059, 2024.

[230] Jiayun Zhang, Petr Byvshev, and Yu Xiao. A video dataset of a wooden box assembly process: dataset. In *Proceedings of the Third Workshop on Data: Acquisition To Analysis*, pages 35–39, 2020.

[231] Jiayun Zhang, Shuheng Li, Haiyu Huang, Zihan Wang, Xiaohan Fu, Dezhi Hong, Rajesh K Gupta, and Jingbo Shang. How few davids improve one goliath: Federated learning in resource-skewed edge computing environments. In *Proceedings of the ACM on Web Conference 2024*, pages 2976–2985, 2024.

[232] Jiayun Zhang, Shuheng Li, Haiyu Huang, Xiaofan Yu, Rajesh K Gupta, and Jingbo Shang. Orthogonal calibration for asynchronous federated learning. *arXiv preprint arXiv:2502.15940*, 2025.

[233] Jiayun Zhang, Junshen Xu, Bugra Can, and Yi Fan. React: Residual-adaptive contextual tuning for fast model adaptation in threat detection. In *Proceedings of the ACM Web Conference 2025*, 2025.

[234] Jiayun Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K Gupta, and Jingbo Shang. Minimally supervised contextual inference from human mobility: an iterative collaborative distillation framework. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 2450–2458, 2023.

[235] Jiayun Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K Gupta, and Jingbo Shang. Contextual inference from sparse shopping transactions based on motif patterns. *IEEE Transactions on Knowledge and Data Engineering*, 37(2):572–583, 2025.

[236] Jiayun Zhang, Xiyuan Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang. Navigating Alignment for Non-identical Client Class Sets: A Label Name-Anchored Federated Learning Framework. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

[237] Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wenchao Xu, and Feijie Wu. Parameterized Knowledge Transfer for Personalized Federated Learning. *Advances in Neural Information Processing Systems*, 34:10092–10104, 2021.

[238] Ke Zhang, Yu-Ru Lin, and Konstantinos Pelechrinis. EigenTransitions with Hypothesis Testing: The Anatomy of Urban Mobility. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 10, 2016.

[239] Ke Zhang, Ganyu Wang, Han Li, Yulong Wang, Hong Chen, and Bin Gu. Asynchronous vertical federated learning for kernelized auc maximization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4244–4255, 2024.

[240] Qiming Zhang, Jing Zhang, Wei Liu, and Dacheng Tao. Category Anchor-Guided Unsupervised Domain Adaptation for Semantic Segmentation. *NeurIPS*, 32, 2019.

[241] Xinyang Zhang, Chenwei Zhang, Xin Luna Dong, Jingbo Shang, and Jiawei Han. Minimally-Supervised Structure-Rich Text Categorization via Learning on Text-Rich Networks. In *Proceedings of the Web Conference*, pages 3258–3268, 2021.

[242] Xiyuan Zhang, Ranak Roy Chowdhury, Jiayun Zhang, Dezhi Hong, Rajesh K Gupta, and Jingbo Shang. Unleashing the power of shared label structures for human activity recognition. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 3340–3350, 2023.

[243] Xiyuan Zhang, Xiaohan Fu, Diyan Teng, Chengyu Dong, Keerthivasan Vijayakumar, Jiayun Zhang, Ranak Roy Chowdhury, Junsheng Han, Dezhi Hong, Rashmi Kulkarni, et al. Physics-informed data denoising for real-life sensing systems. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*, pages 83–96, 2023.

[244] Xiyuan Zhang, Diyan Teng, Ranak Roy Chowdhury, Shuheng Li, Dezhi Hong, Rajesh Gupta, and Jingbo Shang. Unimts: Unified pre-training for motion time series. *Advances in Neural Information Processing Systems*, 37:107469–107493, 2024.

[245] Yu Zhang, Yu Meng, Jiaxin Huang, Frank F Xu, Xuan Wang, and Jiawei Han. Minimally Supervised Categorization of Text with Metadata. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1231–1240, 2020.

[246] Quanling Zhao, Xiaofan Yu, Shengfan Hu, and Tajana Rosing. Multimodalhd: Federated learning over heterogeneous sensor modalities using hyperdimensional computing. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[247] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.

[248] Yong Zhong, Hongtao Liu, Xiaodong Liu, Fan Bao, Weiran Shen, and Chongxuan Li. Deep generative modeling on limited data with regularization by nontransferable pre-trained models. *arXiv preprint arXiv:2208.14133*, 2022.

[249] Yuan Zhong, Nicholas Jing Yuan, Wen Zhong, Fuzheng Zhang, and Xing Xie. You Are Where You Go: Inferring Demographic Attributes from Location Check-ins. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 295–304, 2015.

[250] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. In *ICCV*, pages 19–27, 2015.

[251] Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-Free Knowledge Distillation for Heterogeneous Federated Learning. In *ICML*, pages 12878–12889. PMLR, 2021.

[252] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*, 2018.