Technical University of Crete
School of Electrical and Computer Engineering

Machine Learning Model Hyperparameter Fine-Tuning in a Federated Learning Environment

Georgios Valavanis

Thesis submitted in fulfillment of the requirements for the

Diploma of Electrical and Computer Engineering

Thesis Supervisor: Professor *Sotiris Ioannidis*

Chania, June 2025

Πολυτεχνείο Κρήτης
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Βελτιστοποίηση Υπερπαραμέτρων Μοντέλων Μηχανικής Μάθησης σε Περιβάλλον Federated Learning

Γεώργιος Βαλαβάνης

Διπλωματική εργασία που υποβλήθηκε προς εκπλήρωση των απαιτήσεων για την απόκτηση

Πτυχίου Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών

Επιβλέπων Διπλωματικής: Καθηγητής *Σωτήρης Ιωαννίδης*

Χανιά, Ιούνιος 2025

Technical University of Crete
School of Electrical and Computer Engineering

Machine Learning Model Hyperparameter Fine-Tuning in a Federated Learning Environment

Thesis submitted by
Georgios Valavanis
in fulfillment of the requirements for the
Diploma of Electrical and Computer Engineering

THESIS APPROVAL

Author: _____
Georgios Valavanis

Committee approvals: _____
Sotiris Ioannidis
Professor, Thesis Supervisor, Committee Member

_____
Thrasyvoulos Spyropoulos
Professor, Committee Member

_____
Michail G. Lagoudakis
Professor, Committee Member

Chania, June 2025

Abstract

Federated Learning (FL) has emerged as a paradigm for training machine learning models on decentralized data while preserving privacy. Despite its advantages, the process of hyperparameter fine-tuning remains a critical challenge within FL settings, primarily due to data heterogeneity, communication constraints, and the need for secure collaboration. The present diploma addresses the problem of efficient and privacy-preserving hyperparameter fine-tuning in FL environments by providing a framework that utilizes federated hyperparameter fine-tuning. Here, clients collaboratively explore hyperparameter configurations using local data. Then, after the best hyperparameters are found from the predefined hyperparameter space, a series of secure aggregation rounds takes place at the server. Our system leverages stratified k-fold cross-validation on clients to evaluate hyperparameter combinations locally, encrypted communication to protect model updates, and weighted aggregation to harmonize global model performance. Various classifiers are supported, such as Stochastic Gradient Descent and Gaussian Naive Bayes, providing extended implementation capabilities. Additionally, to ensure data privacy, our framework provides symmetric and asymmetric encryption for the client-server communication. Experimental results demonstrate the efficacy of the approach in achieving similar F1 scores to the implemented non-federated approach while maintaining scalability and security. This work contributes a practical methodology for hyperparameter fine-tuning in FL, balancing performance and privacy.

Keywords: Federated Learning, Hyperparameter fine-tuning, Privacy-Preserving Machine Learning, Data Heterogeneity, Secure Aggregation, Differential Privacy, Secure Multi-Party Computation

Περίληψη

Το Federated Learning (FL) έχει αναδειχθεί ως μια μέθοδος εκπαίδευσης Machine Learning (ML) μοντέλων σε αποκεντρωμένα δεδομένα, που διατηρεί την ιδιωτικότητα και την ασφάλεια. Ωστόσο, παρά τα πλεονεκτήματά του, το hyperparameter fine-tuning παραμένει μια κρίσιμη πρόκληση σε περιβάλλοντα FL λόγω της ετερογένειας των δεδομένων, των περιορισμών επικοινωνίας και της ανάγκης για ασφαλή συνεργασία. Στην παρούσα διπλωματική εργασία θα αντιμετωπιστεί το πρόβλημα του αποδοτικού και ασφαλούς hyperparameter fine-tuning σε περιβάλλον FL. Προτείνεται ένα framework στο οποίο οι clients συνεργάζονται για να εξερευνήσουν συνδυασμούς υπερπαραμέτρων χρησιμοποιώντας τα τοπικά τους δεδομένα. Στη συνέχεια, αφού βρεθούν οι βέλτιστες υπερπαράμετροι από το προκαθορισμένο hyperparameter space, πραγματοποιείται μια σειρά από secure aggregation γύρους στο server. Οι clients αξιοποιούν το stratified k-fold cross validation για την τοπική αξιολόγηση των συνδυασμών υπερπαραμέτρων, την κρυπτογραφημένη επικοινωνία για την προστασία των ενημερώσεων του μοντέλου και το weighted aggregation για την δημιουργία του κεντρικού μοντέλου. Το υλοποιημένο framework υποστηρίζει διάφορους classifiers, όπως Stochastic Gradient Descent και Gaussian Naive Bayes, προσφέροντας εκτεταμένες δυνατότητες. Επιπλέον, για να διασφαλιστεί η ιδιωτικότητα των δεδομένων, παρέχεται συμμετρική και ασύμμετρη κρυπτογράφηση στην επικοινωνία μεταξύ clients και server. Τα πειραματικά αποτελέσματα αποδεικνύουν την αποτελεσματικότητα αυτής της προσέγγισης στην επίτευξη παρόμοιων F1 scores με την υλοποιημένη μη federated προσέγγιση, διατηρώντας παράλληλα την επεκτασιμότητα και την ασφάλεια. Η παρούσα εργασία συνεισφέρει μια πρακτική μεθοδολογία για το hyperparameter fine-tuning στο FL, εξισορροπώντας την απόδοση και την ιδιωτικότητα.

Acknowledgments

I would like to express my sincere gratitude to my research supervisor, Professor Sotiris Ioannidis. His expertise, guidance, and invaluable feedback provided me with the tools necessary to advance in this research journey. I am also grateful to the members of my thesis committee, Thrasyvoulos Spyropoulos and Michail Lagoudakis for their thoughtful questions, their suggestions, and their time. Their expertise has helped me enrich my understanding of the field. I would also like to thank Dr. Alexander Sevtsov and Giannis Lamprou for providing assistance in the implementation of the hyperparameter fine-tuning framework. Throughout this thesis, they assisted me in overcoming several obstacles. My heartfelt thanks go to my family, whose love and encouragement have made this achievement possible. Their constant support and understanding, especially during the challenging moments of this journey, have been my source of strength. Special thanks to my friends who have provided moral support and necessary distractions when the work became overwhelming. Their friendship has been invaluable during this process. This work would not have been possible without the assistance of everyone mentioned above.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 2
- 1.2 Outline . . . 3

2 Background **5**
- 2.1 What is Federated Learning? . . . 5
- 2.2 History and Origins . . . 6
- 2.3 Federated Learning vs Centralized Machine Learning . . . 7
- 2.4 Data Locality and Privacy . . . 10
- 2.5 Aggregation Mechanisms in Federated Learning . . . 11
- 2.6 Communication Frameworks in Federated Learning . . . 12
- 2.7 Challenges in Federated Learning . . . 13
- 2.8 Applications of Federated Learning . . . 15
- 2.9 Hyperparameter fine-tuning in Federated Learning . . . 15

3 Related Work **19**
- 3.1 Optimization of the Federated Learning process . . . 19
- 3.2 Traditional Hyperparameter Optimization Methods . . . 22
- 3.3 Hyperparameter Fine-tuning in Federated Settings . . . 24
- 3.4 Federated Learning Adaptations on Different Domains . . . 26
- 3.5 Performance Evaluation Methods . . . 27

4 Methodology **31**
- 4.1 Hyperparameter fine-tuning process . . . 31
- 4.2 Aggregation Modes . . . 32
- 4.3 Security Framework . . . 34
- 4.4 Experimental Tool . . . 35
- 4.5 Performance Monitoring and Visualization . . . 36
- 4.6 Communication Infrastructure Implementation . . . 36

5 Implementation **39**
- 5.1 Initial Encryption Process and Hyperparameter fine-tuning . . . 39
- 5.2 Model Aggregation Strategies . . . 42

5.3 Communication Layer 43

6 Evaluation **45**
6.1 Datasets 45
6.2 Experimental Methodology 47
6.3 Results and Analysis 48

7 Conclusion and Future Work **51**
7.1 Conclusion 51
7.2 Limitations and Future Work 52

8 List of Acronyms **53**

List of Tables

2.1 Comparison between Centralized ML and FL 9
2.2 Applications of FL Across Different Domains 15

6.1 Performance Comparison Across Models with Residual Path(**Best** F1 scores Weighted/Unweighted/NoFed/NoTune) 48
6.2 Performance Comparison Across Models without Residual Path(**Best** F1 scores Weighted/Unweighted/NoFed/NoTune) 49

Chapter 1

Introduction

The rapid development and usage of smart devices, edge computing platforms, and privacy regulations have fundamentally transformed the landscape of ML. This new ecosystem introduces ML with several challenges related to data privacy, regulatory compliance, communication limitations, and computational scalability. FL has emerged as a compelling solution to these challenges by enabling collaborative model training across decentralized devices without requiring direct access to raw data. This paradigm not only enhances data privacy but also mitigates communication overhead and leverages the computational capabilities of edge nodes. To address these challenges, FL enables collaborative model training across decentralized devices without requiring direct access to client raw data. This approach not only provides data privacy but also reduces communication overhead and utilizes the computational resources of edge devices.

Unlike traditional centralized approaches, FL distributes the training process across multiple clients, each training models on their local data. These locally trained models are then aggregated at a central server to produce a global model that is eventually shared back to the clients. This distributed architecture makes FL particularly valuable in domains such as healthcare, finance, and mobile applications, where data privacy concerns are paramount.

Despite its advantages, the effectiveness of FL systems is heavily dependent on the appropriate configuration of hyperparameters that control the model architecture. Hyperparameter fine-tuning in federated environments introduces unique challenges deriving from data heterogeneity, system diversity, and privacy constraints. Each client's local dataset may exhibit different characteristics (aka. features) distributions , computational capabilities vary substantially across clients/devices, and privacy requirements limit the visibility of client-specific information. These factors complicate the process of identifying well performing hyperparameter configurations that generalize well across the federated network.

This thesis addresses this critical research gap by developing a comprehensive framework for hyperparameter fine-tuning in FL environments. This framework enables exploration of hyperparameter spaces in order to maximize model performance. In order to provide reliable feedback on hyperparameter selection, we adopt stratified k-fold cross-validation, which is performed locally at each client. Furthermore, client-specific evaluations are aggregated through a weighted voting mechanism to identify configurations that perform well across heterogeneous data distributions.

To support secure communication, our implementation leverages modern cryptographic techniques. These techniques include incorporating MD5 checksum verification, as well as symmetric and asymmetric encryption protocols, to maintain data integrity and confidentiality throughout the optimization process. Our system architecture features a communication infrastructure with acknowledgment mechanisms designed to ensure reliable model parameter exchange. The framework's modular design supports multiple machine learning models, including Stochastic Gradient Descent (SGD), Logistic Regression (LogReg), Gaussian Naive Bayes (GNB), and Multi-Layer Perceptron (MLP), offering versatility across diverse learning scenarios.

Beyond addressing the fundamental challenges of hyperparameter fine-tuning in federated environments, our work introduces several innovative components that enhance the overall performance of the system. A key innovation in our framework is the integration of a residual learning mechanism with federated hyperparameter fine-tuning. By maintaining a global parameter state across aggregation rounds, our approach enhances model accuracy. This mechanism operates with both weighted and unweighted aggregation strategies, providing flexibility. The combination of these technical innovations results in a framework that not only addresses the theoretical challenges of federated hyperparameter fine-tuning but also offers practical solutions for real-world implementation.

Therefore, our comprehensive framework bridges an important gap in the FL literature. It provides a systematic approach to hyperparameter fine-tuning that maintains privacy guarantees while achieving competitive model performance. The experimental results, detailed in later chapters, demonstrate the efficacy of our approach across various datasets and model architectures. Furthermore, its potential for broad application in privacy-sensitive domains is highlighted, particularly where traditional centralized optimization approaches are unsuitable or infeasible.

1.1 Contributions

The main contributions of this thesis are:

1. Design of a secure communication infrastructure that incorporates MD5 checksum verification, symmetric and asymmetric encryption protocols, and acknowledgment-

based message exchange, ensuring reliable and confidential transfer of model parameters between clients and the central server.

2. Development of a modular framework for privacy-preserving hyperparameter fine-tuning in FL environments, enabling systematic evaluation of model configurations without compromising data locality.
3. Empirical evaluation of the framework across diverse model architectures and datasets, highlighting trade-offs between model performance, communication efficiency, and privacy preservation.
4. Implementation of a residual learning mechanism that enhances model convergence by maintaining a persistent global state across aggregation rounds, applicable in both weighted and unweighted settings.

1.2 Outline

The layout of this thesis is as follows:

- Chapter 2 presents the theoretical background, introducing the core principles of FL and discussing data locality, privacy concerns, aggregation strategies, communication protocols, and the unique challenges of hyperparameter fine-tuning in decentralized environments.
- Chapter 3 presents a comprehensive review of related work examining existing approaches to federated hyperparameter fine-tuning, traditional and federated hyperparameter fine-tuning methods, domain-specific adaptations, and performance evaluation methodologies.
- Chapter 4 outlines the proposed methodology, detailing the framework's design, security mechanisms, aggregation strategies, communication infrastructure, and evaluation logic.
- Chapter 5 describes the implementation details, covering both the client-side and server-side components of our system, along with the communication protocols and encryption mechanisms employed.
- Chapter 6 presents our experimental evaluation, including dataset descriptions, methodology, and results.
- Chapter 7 concludes the thesis by summarizing key findings, discussing limitations, and proposing future directions for extending the framework's capabilities and applicability.

Chapter 2

Background

The evolution of machine learning from centralized to decentralized implementations represents an important shift in how modern data challenges are addressed. This chapter establishes the theoretical foundations of federated learning (FL), tracing its development from conceptual origins to practical implementation frameworks. First, the core principles that differentiate FL from traditional approaches, including data locality, privacy preservation, and distributed optimization, are examined. The discussion then progresses through key architectural components, aggregation mechanisms, and communication protocols that enable collaborative model training across decentralized clients. Establishing this theoretical groundwork provides context for understanding the specific challenges of hyperparameter fine-tuning in federated environments.

2.1 What is Federated Learning?

FL enables training of ML models without the need to access raw data. Operating in a decentralized manner, several devices or entities, called clients, collaborate in order to train a common model while keeping their data localized. In contrast to traditional centralized ML, where data are collected in a single location for processing, in FL, clients and a central server exchange model updates, such as gradients or model parameters. This ensures that raw data remain localized and private, prioritizing privacy and security concerns.

At its core, FL aims to solve a distributed optimization problem. Each client trains on its local data and periodically contributes to a global model by sharing updates. These updates are aggregated at the server to refine the global model, and then the global model is distributed back to the clients. This iterative process continues until the model finally converges. The standard algorithm for facilitating this training process is Federated Averaging (FedAvg) [1], which provides a simple and efficient way of averaging local model updates from clients.

Federated learning is usually implemented using three distinct topologies. **Horizontal Federated Learning (HFL)** represents the most straightforward implementation. Here, every client has the same feature space. Thus, the data used for aggregation from different clients will have the same feature variables. The second FL topology is **Vertical Federated Learning (VFL)**. In VFL, clients may have different business models but share the same customers. In contrast to HFL, the feature space of the data from the clients might be different. This approach usually requires secure alignment techniques to match overlapping samples without exposing raw data. The main focus of VFL is to combine complementary features to improve model performance. The third and final topology of FL is **Federated Transfer Learning (FTL)**. FTL is particularly useful when datasets differ in both sample and feature spaces. This topology enables model training in entirely different domains by transferring learned representations. This method proves particularly valuable when direct collaboration becomes impossible due to fundamental data heterogeneity or regulatory constraints.

In recent years, FL has gained significant attention for its ability to deliver competitive model performance while also providing privacy guarantees. These traits make FL particularly useful in sensitive domains such as healthcare, finance, and personalized applications on mobile devices. This growing interest stems from increasing regulatory pressures and consumer demand for privacy-preserving data utilization methods that do not compromise on analytical capabilities or insights. To better understand the development of this important paradigm, the historical origins of FL will be explored.

2.2 History and Origins

The foundations of FL derive from the intersection of optimization theory, distributed systems, and privacy-related technologies. FL was initially introduced by Google in 2016 [1], with the implementation of the Federated Averaging (FedAvg) algorithm. This algorithm presented a really effective way of combining locally computed updates from multiple clients into a global model without requiring the transfer of raw data, making it the standard approach for FL. Building upon this foundation, Google expanded FL's applications by using it to develop predictive text models for their mobile keyboard Gboard [2]. This practical application demonstrated how FL can enhance the predictive models' performance while also offering privacy to the client's data. By training language models directly on users' devices rather than centralizing all their data, Google opened the path for more research and development.

The conceptual roots of FL can be traced to earlier research in distributed optimization. Distributed optimization has always aimed to solve large computational problems by distributing tasks among a network of devices. Techniques such as the Distributed Mini-Batch Algorithm (DPM) [3] and distributed consensus algorithms

[4] initially addressed issues with decentralized data, establishing the groundwork for FL. A significant milestone was the development of the MapReduce framework by Google in 2004 [5]. Although not directly related to FL, MapReduce inspired further research into more complex distributed systems by proving that decentralized computation was feasible at scale. Furthermore, advancements in cloud and edge computing created the technological infrastructure necessary for FL by enabling the distribution of computational resources among geographically dispersed devices.

Parallel to these computational developments, the emergence of privacy issues with conventional ML significantly influenced FL's trajectory. Two early methods of privacy-preserving computation were Secure Multi-Party Computation (SMPC) [6] and Differential Privacy (DP) [7] showing that collaborative computation could take place without disclosing sensitive information. These methods influenced the design of FL by introducing techniques for secure aggregation and ensuring that client updates could be shared without compromising user privacy. The rise of DP in particular, was a significant turning point. DP ensures that individual data points cannot be deduced from aggregated output by offering a mathematical framework for quantifying privacy guarantees. As evidenced by studies such as [8], the incorporation of DP into FL has emerged as a fundamental component of contemporary FL systems, especially for applications that demand robust privacy guarantees.

The proliferation of data privacy legislation has also played a pivotal role in accelerating FL's development and adoption. Strict guidelines for data handling, storage, and sharing were established by the California Consumer Privacy Act (CCPA) in the US and the General Data Protection Regulation (GDPR) in Europe. Due to these rules, the demand for privacy-preserving technologies such as FL that follow the guidelines of user consent and data minimization has risen [9]. FL offers a feasible solution to regulatory challenges by restricting data transfer and ensuring data remains local. Its distributed design naturally complies with various privacy standards, making it a desirable choice for companies functioning in heavily regulated industries such as banking, healthcare, and telecommunications.

Having established the historical context for FL's development, a direct comparison with traditional centralized ML approaches follows in order to better understand the distinctive advantages and limitations of the federated paradigm.

2.3 Federated Learning vs Centralized Machine Learning

Compared to traditional centralized ML, which has long been the most popular method for model training, FL offers a different solution. In this section, the main differences between these two methods are examined with regard to architecture,

communication, privacy, scalability, and practical applications.

In centralized ML, training data from all users or devices is combined into a single centralized server. The model is trained using this combined dataset, allowing for direct access to all the information. This process assumes the availability of high-bandwidth networks and reliable infrastructure to support data transfer, which can be infeasible in modern distributed systems. On the contrary, FL adopts a decentralized learning architecture, where the raw data remains localized on client devices. Instead of transferring raw data, only the model updates (e.g., gradients or weights) are sent to a central server for aggregation. These updates are then aggregated by the central server to create a global model, which is shared among the clients afterwards. This implementation solves a number of issues related to large-scale data movement by eliminating the need for centralized data collection [1].

Communication efficiency is another important aspect of ML, especially when there is a large amount of available data. In centralized ML, all training data are transferred to the server once. The data exchanged later on may be reduced, but more communication is required in the beginning. Large volumes of raw data can be costly and time-consuming to transfer as datasets get bigger, particularly in environments with limited bandwidth. On the other hand, FL distributes the communication costs over a number of model training cycles. Instead of transferring raw data, model updates are communicated periodically. To further lower the communication overhead, methods such as quantization, gradient compression, and sparsification are frequently used [10]. Even though FL relies on iterative operations, distributed training tasks can still be effectively completed by optimizing the communication overhead.

Since raw user data needs to be moved to a central server, centralized ML presents serious privacy risks. Data breaches, illegal access, and non-compliance with regulations are issues that arise from this data movement in sensitive applications such as healthcare, finance, and personalized services. For instance, compliance with privacy laws such as the GDPR and CCPA can be challenging when large volumes of private data are centralized [9]. In contrast, FL guarantees that sensitive raw data never leaves the client devices, making it a more privacy-preserving option. To lessen the possibility of privacy breaches, only aggregated or encrypted model updates are exchanged. Additional safeguards such as homomorphic encryption, secure multi-party computation [6], and DP ensure that individual data points cannot be inferred from the updates. All of the above make FL a more suitable choice for privacy-sensitive tasks.

Scalability is another dimension where FL demonstrates significant potential. As data volumes grow, the need for high-capacity servers, computational resources, and storage infrastructure increases significantly. Furthermore, stable network connectivity is necessary for centralized training in order to transfer data, which may not be possible for edge devices or geographically scattered data sources. By com-

parison, FL's decentralized architecture makes it more scalable by nature. FL leverages distributed computational resources available on client devices, enabling large-scale model training without overwhelming a central server. Also, it eliminates the need for centralized data storage because client training takes place locally. As a result of its distributed nature, FL is especially well-suited for edge devices such as smartphones, Internet of Things (IoT) gadgets, and self-governing systems.

Nonetheless, centralized ML retains certain advantages, most notably its unrestricted access to the full dataset. This access allows for better model refinement. Centralized models tend to achieve higher accuracy and reliability due to their direct access to the whole dataset, especially when data is non-independent and identically distributed (Non-IID) [10]. That's where FL usually struggles to perform. In Non-IID settings, clients might possess incomplete or highly biased data, complicating model convergence. Researchers have developed several approaches to face data heterogeneity. Some of those approaches include Personalized Federated Learning with Adaptive Feature Aggregation and Knowledge Transfer (FedAFK) [11], and adaptive optimization techniques such as FedProx [12]. Despite these obstacles, FL has demonstrated competitive performance across a variety of applications, especially under conditions with limited resources and privacy considerations.

In summary, while centralized ML excels at producing highly accurate models through direct access to global data, FL emerged specifically to address its limitations in managing distributed, privacy-sensitive datasets. With its decentralized architecture, robust privacy guarantees, and communication-efficient techniques, FL represents a promising paradigm for contemporary ML applications in distributed environments. The following table summarizes the fundamental differences between centralized ML and FL.

Aspect	**Centralized ML**	**FL**
Data Storage	Centralized on a central server	Localized on client devices
Communication Efficiency	One-time transfer of raw data	Iterative exchange of model updates
Privacy	Higher risk of breaches and regulatory challenges	Privacy-preserving through techniques such as DP and Homomorphic Encryption (HE)
Scalability	Limited by central server resources	Scales effectively using distributed computational resources
Performance	High accuracy with balanced data	May face challenges with Non-IID data distribution
Applications	Suitable for centralized, infrastructure-rich scenarios	Ideal for privacy-sensitive and distributed systems

Table 2.1: Comparison between Centralized ML and FL

2.4 Data Locality and Privacy

One of the main focuses of FL is **data locality**, the practice of storing raw data locally rather than moving it to a central server. This approach is essential for safeguarding user privacy, ensuring compliance with data governance regulations, and enabling large-scale distributed ML across diverse environments.

In a centralized configuration, raw data is collected into a single server. While this method grants full dataset access, it simultaneously introduces potential privacy breaches, regulatory obstacles, and high communication expenses, as previously mentioned. FL addresses these issues by keeping data on the originating storage device. This localized approach offers several benefits. Since raw data is never transmitted, the likelihood of exposing sensitive information to unauthorized parties is minimized [9]. Furthermore, data locality allows resource-constrained devices, such as mobile phones and IoT sensors, to participate in the training process without transferring large amounts of data [1].

Data locality, though, cannot ensure data security as a standalone component. To further protect sensitive data and mitigate potential vulnerabilities, FL systems typically incorporate advanced privacy-preserving techniques. Key methods include DP, secure aggregation and trusted execution environments, each providing distinct layers of protection within the federated architecture.

Differential privacy (DP) introduces statistical noise into model updates to mask the impact of individual data points. This ensures that, even if an adversary gains access to the global model, extracting specific details about individual records from the aggregated updates remains unattainable. DP has emerged as a standard technique for maintaining a balance between privacy and utility in FL [13].

Secure aggregation protocols complement data locality by ensuring that the central server can access only the aggregated model updates rather than individual client contributions. Based on SMPC principles, these protocols enable encrypted transmission of model updates prior to aggregation. This encryption mechanism prevents both the server and adversary actors from looking at any individual updates by encrypting model updates prior to transmission [14].

Trusted Execution Environments (TEEs), such as Intel SGX, provide secure hardware-based isolation for computations. By establishing secure enclaves for processing sensitive data and model updates, TEEs ensure that even if the server is compromised, model updates and aggregation operations remain protected within an isolated environment [15].

Despite these substantial advantages, ensuring privacy in FL introduces several challenges. Techniques such as DP and encryption can introduce trade-offs between privacy, model accuracy, and communication efficiency. For example, adding noise for DP may degrade the accuracy of the trained model, particularly in cases with limited data or high heterogeneity [10]. Additionally, privacy-preserving methods

such as HE and secure aggregation often incur significant computational costs, especially on resource-constrained devices. Generally, privacy mechanisms can increase the size of model updates or require additional rounds of communication, reducing overall efficiency.

With data locality and privacy considerations established, our attention will be turned to the mechanisms through which client model updates are combined to create an effective global model in FL systems.

2.5 Aggregation Mechanisms in Federated Learning

FL relies on aggregation mechanisms, which allow local model updates to be aggregated into a global model. These mechanisms are critical for addressing the challenges posed by Non-IID data and for ensuring robust model convergence across heterogeneous client environments. Fundamental aggregation mechanisms include FedAvg [1] and FedProx [12].

Federated averaging represents the foundational algorithm in FL aggregation. The core principle of FedAvg is to compute a weighted average of local model parameters, where the weights are proportional to the number of training samples at each client. The FedAvg algorithm can be mathematically formulated as:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1} \tag{2.1}$$

where w_{t+1} represents the global model at iteration $t+1$, K is the total number of clients, n_k is the number of training samples at client k, n is the total number of training samples across all clients and w_k^{t+1} represents the local model parameters of client k.

While FedAvg offers a fundamental method, it has a number of drawbacks, especially in settings with skewed data distributions and substantial data heterogeneity.

To address the limitations of standard FedAvg, researchers have developed more sophisticated aggregation strategies, such as FedProx. FedProx introduces a proximal term to the local optimization objective that constrains the distance between local and global models [12]. This modification helps mitigate performance degradation in Non-IID settings:

$$\min_w h_k(w) = F_k(w) + \frac{\mu}{2}\|w - w_t\|^2 \tag{2.2}$$

where $h_k(w)$ is the objective function for client k, $F_k(w)$ represents the local loss function for client k, μ is the proximal term coefficient that controls the strength of regularization, w denotes the model parameters, and w_t represents the global

model parameters at round t.

While aggregation mechanisms determine how model updates are combined, the efficiency and reliability of the overall FL process depend heavily on the communication framework that facilitates the exchange of these updates between clients and servers.

2.6 Communication Frameworks in Federated Learning

Communication frameworks form the backbone of FL systems, enabling clients and servers to share model updates. These frameworks must address several challenges, including expensive communication costs and system scalability and synchronization issues, particularly in environments where clients may drop out or join unexpectedly. Additionally, varying device capabilities (system heterogeneity) can lead to stragglers and unbalanced participation, further complicating the communication process.

The predominant architecture in FL involves a central server that coordinates with multiple clients [1]. Within this architecture, several communication patterns can be implemented. The most common is synchronous communication. This is where all clients participate in each round of training. The simplicity of synchronous communication offers consistency in model updates and makes aggregation simpler. However, slow clients increase idle time, making it ineffective in environments where clients have varying capabilities. An alternative approach is asynchronous communication [16], where clients operate autonomously. The server, in this case, continuously updates the global model upon receiving new updates. Unlike synchronous communication, it eliminates idle time, vastly improving resource utilization. However, it raises the risk of stale updates (older model updates conflicting with new ones), potentially harming model convergence. Lastly, a third option is semi-synchronous communication, which combines the previous patterns. Here, the server proceeds after receiving updates from a predetermined fraction of clients. By tolerating partial client participation, it balances efficiency and robustness. But, because it is a more complex approach, it requires careful tuning of the participation threshold to avoid bias or wasted computation.

A key objective of communication frameworks is to reduce communication overhead, prompting the development of several optimization techniques. These techniques might involve the compression of the gradients shared between clients and the server. Compression methods [17] include quantization and sparsification. Quantization reduces the number of bits needed to represent each parameter, and sparsification constrains the transmission of the gradient components to the most significant ones. These techniques effectively reduce the volume of data exchanged between clients and the server, thereby improving communication efficiency without substantially compromising model performance.

Another objective of communication frameworks is to minimize the communication rounds without negatively impacting the global model quality. Adaptive local updates [18] are one way to achieve that, where clients dynamically adjust local computation based on data characteristics. Adaptive local updates optimize convergence while mitigating bias through techniques such as gradient correction. A more straightforward approach to minimize communication rounds is periodic averaging. Here, models are synchronized after multiple local training rounds.

Modern FL communication frameworks can respond to network conditions by implementing adaptive protocols. For example, systems such as FedCS [19] dynamically select clients based on real-time resource availability (e.g., computational capacity, data size, and wireless channel conditions) to maximize participation within predefined deadlines. This approach reduces training time and improves convergence efficiency, with minimal impact on model accuracy. Reducing training time also results in enhanced energy efficiency by avoiding prolonged communication with resource-constrained devices.

Despite these advancements in aggregation and communication frameworks, FL systems face several fundamental challenges that must be addressed to ensure their practical effectiveness in real-world applications.

2.7 Challenges in Federated Learning

Despite its promising advantages, FL faces several challenges that impact its implementation and performance. Main challenges include data heterogeneity, system heterogeneity, communication efficiency, resource constraints, and privacy and security concerns, each presenting unique obstacles that require careful consideration in federated system design.

Non-IID data heterogeneity poses a core challenge within FL [20]. Contrary to conventional centralized learning, which allows for data shuffling and balancing, FL must navigate through naturally imbalanced and client-specific data distributions. According to [9], Non-IID scenarios in FL can be classified into several categories: **Feature Distribution Skew**, **Label Distribution Skew**, **Quantity Skew**, **Same label, different feature** and **Same features, different label**. **Feature Distribution Skew** is the scenario where different clients have different feature distributions for the same labels. **Label Distribution Skew** refers to cases where the distribution of labels varies across clients. **Quantity Skew** arises when the amount of data varies significantly from client to client. When the same label y may correspond to vastly different features x across various clients, then there is the **same label, different features** scenario. For example, images of houses can differ across countries. Finally, when identical feature vectors can be associated with varying labels, mainly due to individual preferences, then the **same features, different label** scenario takes place. This is evident in next-word prediction scenarios, where the same next-word predictors may hold diverse

interpretations across different individuals.

These Non-IID data distributions in FL compromise model convergence rate due to conflicting local updates [10]. Clients owning larger datasets may introduce bias to the global model in their favor, and unseen data distributions may lead to poor model performance.

System heterogeneity introduces additional complexity to FL implementations. Clients typically have varying resource constraints and computational capabilities that affect the system's overall performance. Devices with varying CPU/GPU capabilities affect the system's training speed, and client memory capacities limit model size and batch processing. Also, clients that are battery-powered have limited operating time, potentially leading to intermittent participation.

When having resource-constrained clients, it is important that the communication between the clients and the server is efficient [21]. Providing an efficient communication framework for FL has several challenges. Particularly, clients that are edge devices have limited upload capacity, impacting the communication speed. Varying network quality also impacts communication speed and financial costs associated with data transfer, emphasizing the need for efficient communication.

Client availability patterns introduce yet another layer of complexity [22]. They may join or leave the system unpredictably, slower devices can delay the overall training process, and the global distribution of clients across different time zones and usage patterns can significantly affect participation rates and system performance.

While FL offers a more secure environment for client data, it still faces a variety of threats [23]. Key privacy concerns include inference attacks, where adversaries may infer private information from model updates. Another threat is model inversion, where training data can be reconstructed from model parameters. Additionally, membership inference can reveal the participant's IDs by determining if specific data was used in training. Moreover, the clients themselves can be malicious, sharing corrupted updates to the server (Byzantine attacks), or they may attempt to benefit from the global model without contributing (free-riding).

Various approaches have been proposed to address all the above challenges. Regarding data heterogeneity, FedProx offers a more robust aggregation strategy [12]. Other solutions include adapting global models to local distributions and grouping clients with similar data distributions. To address system heterogeneity, solutions include allowing clients to participate independently, selecting clients based on their resource limitations, and accommodating varying client contributions. Privacy and security in FL can be strengthened through cryptographic approaches. Applying DP to mask individual contributions with noise [13] and secure aggregation protocols to ensure that servers can only access aggregated model updates helps ensure privacy.

2.8 Applications of Federated Learning

FL has found applications across several domains, especially where data privacy and distributed computation are essential. The following table provides a comprehensive overview of FL applications across different fields.

Domain	Key Applications	Notable Examples
Healthcare [24]	• Medical imaging analysis. • Electronic health records. • Drug discovery. • Clinical trials.	• Brain tumor detection. • Patient outcome prediction. • Personalized treatment recommendations. • Disease progression modeling.
Mobile Computing [2]	• Keyboard prediction. • Content recommendation. • Device optimization. • User behavior modeling.	• Next-word prediction. • News feed personalization. • Battery management. • App usage optimization.
Financial Services [25]	• Risk assessment. • Fraud detection. • Market analysis. • Customer modeling.	• Credit scoring. • Anti-money laundering. • Trading strategies. • Portfolio optimization.
Industrial IoT [26]	• Predictive maintenance. • Quality control. • Process optimization. • Communication optimization.	• Equipment failure prediction. • Production line optimization. • Inventory management. • Demand forecasting.
Smart Cities [26]	• Transportation. • Energy management. • Urban planning. • Environmental monitoring.	• Traffic optimization. • Smart grid management. • Waste management. • Air quality prediction.
Edge Computing [26]	• IoT networks. • Sensor systems. • Device coordination. • Network optimization.	• Sensor data analysis. • Network load balancing. • Resource allocation. • Device synchronization.

Table 2.2: Applications of FL Across Different Domains

2.9 Hyperparameter fine-tuning in Federated Learning

Hyperparameter fine-tuning is an essential component of FL. Although, it presents unique challenges and considerations that differentiate it from centralized ML ap-

proaches. This section explores the critical role of hyperparameters in FL and various optimization methods adapted for federated environments. These foundations set the stage for the novel contributions presented in later chapters of this thesis.

Hyperparameters in every ML system play an important role in determining both the model's performance and the system's overall operational efficiency. But, unlike centralized learning, FL systems must consider both model-related hyperparameters and hyperparameters specifically related to the federated environment [27]. These parameters significantly affect various aspects of the FL process. For instance, conventional hyperparameters such as the learning rate require taking into account the dynamics of local and global optimization. If an inappropriate learning rate is used, it will result in poor model convergence, especially in settings with Non-IID data [10]. An additional example is that when using SGD, local batch size affects not only the model's convergence but also the communication efficiency and the resource utilization of client devices. Using larger batch sizes can reduce communication efficiency but may require more memory and resources from clients [1]. Generally, in FL when selecting a model architecture, it is important to take into account the communication overhead regarding updating parameters and the processing power of the client devices.

In addition to traditional hyperparameters, FL introduces parameters unique to its distributed nature. For example, the fraction of clients selected for each round of training affects both the convergence speed and the general robustness of the global model [27]. A smaller fraction of clients can reduce communication overhead but may slow overall model convergence, whereas a larger one can increase model reliability at the cost of extended usage of resources. Another important hyperparameter is the number of local training rounds (epochs) per client before aggregation. More local training rounds might lessen the communication frequency but might cause a divergence in local models, especially when local client data are Non-IID. Thus, careful hyperparameter tuning of those parameters is essential for achieving optimal model performance.

Several methods have been introduced to perform hyperparameter tuning effectively in FL. One baseline approach is Federated Grid Search, which extends traditional Grid Search by evaluating different hyperparameter configurations across multiple federated training runs [27]. Federated Grid Search maintains separate global models for different hyperparameter configurations and evaluates performance using a validation set. While conceptually straightforward, it may require significant computational resources and communication overhead.

More sophisticated approaches include adaptive methods that adjust hyperparameters dynamically during the training process. FedADAM [28] extends the ADAM optimizer to federated settings, automatically adjusting learning rates based on the statistics of model updates. FedEx implements a multi-armed bandit approach to hyperparameter fine-tuning, treating each configuration as an arm and using the

improvement in model performance as the reward signal [29].

Further challenges regarding hyperparameter fine-tuning in FL, include limited server visibility. More specifically, the central server has limited visibility into client-side metrics and performance indicators [27]. Also, the impact of hyperparameter changes may not be immediately apparent due to the asynchronous nature of federation rounds [10]. Additionally, device heterogeneity affects hyperparameter fine-tuning. Different clients have varying computational and memory capabilities, making it difficult to find universally optimal hyperparameters [10]. Fluctuating network conditions affect the feasibility of different hyperparameter configurations, particularly those impacting communication frequency [29]. Non-IID data distributions further complicate hyperparameter fine-tuning. Different clients may require different hyperparameters for optimal performance on their local datasets [27] and hyperparameter configurations that work well for some clients may lead to instability or divergence for others [10]. The resource-intensive nature of hyperparameter tuning poses additional challenges in federated settings. Evaluating multiple hyperparameter configurations significantly increases communication costs [27], and running parallel training runs with different configurations strains both server and client resources [29]. These practical constraints often limit the thoroughness of hyperparameter exploration in federated environments compared to centralized approaches.

Addressing these challenges requires federated hyperparameter fine-tuning techniques that balance effectiveness with efficiency, privacy preservation, and adaptability to heterogeneous environments. This need serves as the primary motivation for the framework presented in this thesis, which aims to provide a comprehensive solution for hyperparameter fine-tuning in FL environments while respecting the unique constraints of the federated setting.

Chapter 3

Related Work

This chapter examines the research landscape surrounding hyperparameter fine-tuning in FL environments. It focuses on approaches and frameworks most relevant to our proposed solution. First, existing FL optimization techniques are explored, particularly those addressing challenges our framework aims to overcome. Then, traditional hyperparameter fine-tuning methods and their limitations when applied to distributed systems are assessed. Next, this chapter examines specialized approaches specifically for FL, evaluating their strengths and shortcomings relative to our proposed framework. Finally, essential performance evaluation methodologies are reviewed. Throughout this analysis, specific research gaps that our framework addresses are identified, showcasing how our work fits into existing research.

3.1 Optimization of the Federated Learning process

Current FL optimization techniques face unique challenges because their distributed nature adds more complexity. Communication constraints and data heterogeneity are some of the main challenges that these techniques face. This section focuses on various approaches that emphasize the optimization of FL systems, more specifically client selection strategies, aggregation methods, and communication efficiency.

Client selection in FL affects both model performance and system efficiency; that's why it is crucial to choose a proper client selection strategy. Various strategies have been proposed to optimize the selection process; one of them is importance-based sampling strategies. Importance-based sampling strategies in FL aim to select clients that contribute the most to model improvement. This approach involves calculating a selection probability p_k for each client k, which is typically proportional to its importance:

$$p_k = \frac{\text{importance}_k}{\sum_{i=1}^{K} \text{importance}_i} \tag{3.1}$$

There are several metrics that can be utilized to determine client importance. For example, loss can be used as a metric, where clients with higher loss values are prioritized because they may benefit more from training. Clients with higher loss values may represent more divergent data, and that's why they are critical for improving the overall model robustness. While this approach can improve model robustness on heterogeneous data, it may overemphasize outlier distributions at the expense of general performance, a limitation our framework addresses through balanced importance weighting.

Gradient direction diversity offers another metric for client importance, measuring variation in update directions across clients. Clients that contribute unique gradient direction can prevent the model from overfitting. High gradient diversity can be identified by applying cosine similarity-based weighting. This can lead to faster convergence and better performance, especially on heterogeneous data.

Additionally, considering factors such as data freshness and relevance can help prioritize clients with more valuable data. Fresh data reflects current trends, while relevant data aligns closely with the learning objective. For example, clients with recently updated datasets can be weighted higher during aggregation to reduce bias from outdated or irrelevant data. Metrics that evaluate data freshness and relevance are timestamp differentials or domain-specific relevance scores.

Recent works have explored various importance-based sampling strategies to optimize client selection in FL. For example, [30] proposes an optimal client sampling scheme that selects clients based on the importance of their updates, measured by the norm of the update. This aims to reduce communication overhead while maintaining performance.

Beyond importance metrics, system constraints and client capabilities also influence selection decisions. Resource-aware client selection [10, 19] represents another dimension of optimization, considering factors such as available CPU/GPU resources, bandwidth, latency, and memory capacity. While these approaches effectively manage system heterogeneity, they typically operate independently of hyperparameter fine-tuning processes.

Although client selection strategies play a pivotal role regarding performance and convergence in FL, using an appropriate aggregation method can increase model robustness and adaptability to new data. Aggregation methods determine how local model updates are combined into a global model, directly affecting which hyperparameter configurations perform best in federated environments. Since the introduction of FedAvg, various approaches have been proposed that can address its limitations. For instance, Stochastic Controlled Averaging for Federated Learning (SCAFFOLD) is a promising alternative. It addresses client drift in FL by using control variates to correct local updates, formulated as [31]:

$$
\begin{aligned}
y_i &\leftarrow y_i - \eta_l(g_i(y_i) - c_i + c) \\
c_i^+ &\leftarrow \begin{cases} g_i(x) & \text{(Option I)} \\ c_i - c + \frac{1}{K\eta_l}(x - y_i) & \text{(Option II)} \end{cases} \\
x &\leftarrow x + \frac{\eta_g}{|S|} \sum_{i \in S} (y_i - x) \\
c &\leftarrow c + \frac{1}{N} \sum_{i \in S} (c_i^+ - c_i)
\end{aligned}
\tag{3.2}
$$

where y_i represents the local model for client i, $g_i(y_i)$ is the local gradient, c and c_i are the global and local control variates respectively, η_l and η_g are the local and global learning rates, K is the number of local steps, and S is the set of participating clients.

The algorithm maintains control variates at both server and client levels to track and correct for the divergence between local client objectives and the global objective. Option I requires computing an additional gradient at the server model, while Option II reuses previously computed gradients. This control variate mechanism enables SCAFFOLD to achieve superior convergence rates in heterogeneous federated settings compared to standard federated averaging. However, this approach requires maintaining additional state variables at both the server and client sides, increasing memory overhead and communication complexity, particularly when dealing with large-scale federated networks.

Another approach is FedNova [32], where local updates are normalized based on their optimization trajectories. This normalization helps address bias introduced by varying numbers of local update steps across clients, providing more consistent convergence behavior that benefits hyperparameter search processes. In general, weighted aggregation methods can have many different strategies to assign importance to client updates.

These weighting strategies can significantly impact which hyperparameter configurations perform best, as they alter how different client contributions influence the global model.

Beyond performance optimization, aggregation methods can provide robustness against adversarial clients that contribute noisy updates. Some common techniques include median-based aggregation and trimmed mean. The trimmed mean method discards a predefined fraction of the highest and lowest values before averaging, effectively reducing the influence of outliers [33]. Median-based aggregation computes the median of each parameter across client updates, which minimizes the impact of extreme or corrupted values [34]. These robust aggregation approaches ensure that hyperparameter fine-tuning processes remain effective even in the presence of potentially compromised clients, an important consideration for real-world FL deployments.

Although there are aggregation methods that can outperform FedAvg, they introduce further complexity to the implementation. FedAvg is a straightforward aggregation method and is widely used, allowing direct comparison between frameworks. Its simplicity and ease of use were the main reasons that it was chosen as a main aggregation method for our framework.

After examining optimization techniques that influence FL performance, the discussion now turns to traditional hyperparameter fine-tuning methods and their applicability to federated environments.

3.2 Traditional Hyperparameter Optimization Methods

Traditional hyperparameter fine-tuning methods developed for centralized environments serve as the foundation for federated approaches but require significant adaptation to operate effectively in distributed settings. This section examines these fundamental methods—including Grid Search, Random Search, Bayesian Optimization, and gradient-based approaches—analyzing their strengths and limitations when applied to FL.

Grid Search and Random Search constitute the earliest approaches to federated hyperparameter fine-tuning. While conceptually simple, they established the foundation for more sophisticated techniques and remain relevant as baseline methods. More specifically, in Grid Search, combinations of predefined hyperparameter values are evaluated in a discrete search space. This exhaustive approach ensures complete coverage of the search space by evaluating all combinations systematically. The exploration process can also be parallelized since evaluations are independent, offering some efficiency gains. A major drawback of this approach is that the cost of exploration increases exponentially with the number of hyperparameters, making it expensive for complex models such as neural networks.

A more efficient alternative to Grid search is Random search. Introduced by [35], Random Search samples hyperparameter configurations randomly from predefined distributions, as stated in the following equation:

$$\theta_i \sim p(\theta), i = 1, \ldots, N \tag{3.3}$$

where $p(\theta)$ represents the probability distribution over the hyperparameter space.

Studies have shown that Random Search is more effective than Grid Search. As shown in [35], this approach offers several advantages. Firstly, it is more likely to find optimal configurations when certain hyperparameters significantly affect model performance. Secondly, it maintains statistical validity even when stopped early. And thirdly, it consistently outperforms Grid Search given the same computational resources. These advantages stem from Random Search's ability to explore the hyperparameter space more effectively, rather than exhaustively exploring the

whole space. Those traits make Random Search a superior baseline approach, and that's why it is implemented in our framework.

While Grid Search and Random Search rely on uninformed exploration of parameter spaces, Bayesian Optimization employs probabilistic modeling to guide the search process more efficiently [36]. Bayesian Optimization builds a surrogate model, typically using Gaussian Processes to predict promising configurations, requiring far fewer evaluations than brute-force methods.

Gaussian Process (GP) optimization relies on some particular components in order to perform optimally. One important component is the acquisition function. Some common acquisition functions are Expected Improvement (EI), Probability of Improvement (PI) and Upper Confidence Bound (UCB). These functions balance the exploration and exploitation trade-off by determining the most promising hyperparameters to evaluate. The kernel function represents another critical component, with popular choices including the Matern kernel, the Radial Basis Function, and Automatic Relevance Determination. Each kernel offers different trade-offs in smoothness and sensitivity to input dimensions.

Further developments extend Bayesian Optimization to multi-task scenarios. Transfer learning leverages information from similar tasks to accelerate model convergence [37]. Multi-fidelity optimization uses cheaper approximate evaluations (e.g., low-fidelity simulations or smaller datasets) to guide the search, providing high-fidelity results [38]. Lastly, meta-learning utilizes prior knowledge from similar problems to warm-start or bias the optimization process [39]. These multi-task approaches enable more scalable and data-efficient hyperparameter fine-tuning, particularly in complex, resource-intensive applications.

Recent advances in hyperparameter fine-tuning have led to the utilization of Tree-Structured Parzen Estimator (TPE). TPE [40] models the conditional probability of hyperparameters given their performance. TPEs have several advantages, particularly relevant to federated settings. They handle mixed search spaces much better than GPs. They optimize categorical and continuous hyperparameters seamlessly. Also, TPEs are highly scalable in high-dimensional settings, making them more efficient in large-scale tuning tasks. Another strength of TPEs is their ability to model tree-structured dependencies between hyperparameters. This makes TPE a robust choice for modern AutoML pipelines and resource-intensive optimization problems in FL.

Gradient-based methods offer an alternative approach to hyperparameter fine-tuning. These methods enable direct optimization of hyperparameters through gradient computation. For instance, hypergradient descent calculates gradients using the chain rule [41]. Hypergradient descent leverages both forward and reverse mode. Forward mode tracks the evolution of hyperparameters during the training rounds, while reverse mode computes the final hypergradients using chain rule propagation through the previous rounds. Additionally, it employs implicit

differentiation at convergence, utilizing gradients from the optimizer's fixed-point equation. This way, unrolling the entire optimization path is avoided.

This type of approach can extend to bilevel optimization, where hyperparameter fine-tuning is formulated as a bilevel optimization problem [42]:

$$\begin{aligned} \min_{\theta} \quad & \mathcal{L}_{\text{valid}}(w^*(\theta), \theta) \\ \text{s.t.} \quad & w^*(\theta) = \arg\min_{w} \mathcal{L}_{\text{train}}(w, \theta) \end{aligned} \tag{3.4}$$

where $\mathcal{L}_{\text{valid}}(w^*(\theta), \theta)$ is the validation loss as a function of both optimal model parameters and hyperparameters, $\mathcal{L}_{\text{train}}(w, \theta)$ is the training loss, w represents the model parameters, and $\arg\min_w$ denotes the parameters that minimize the training loss.

This bilevel formulation captures the hierarchical nature of hyperparameter fine-tuning, where the choice of hyperparameters θ at the upper level determines the optimal model parameters $w^*(\theta)$ at the lower level, which in turn affects the validation performance used to evaluate the hyperparameters. Key aspects of bilevel optimization include approximate solutions, where truncated training rounds are utilized. Bilevel optimization also employs checkpointing and reversible computations for memory efficiency.

While these traditional methods provide the foundation for hyperparameter fine-tuning, the unique characteristics of federated environments necessitate specialized approaches, which are explored in the following section.

3.3 Hyperparameter Fine-tuning in Federated Settings

Federated hyperparameter fine-tuning has gained attention in the past years. Modern computing infrastructure is moving towards distributed architectures due to the increasing complexity of ML models and the abundance of available data. That has increased the need for the development of sophisticated approaches for hyperparameter fine-tuning. This section explores key methodologies of hyperparameter fine-tuning developed specifically for federated environments.

Training large-scale ML models across multiple devices requires frameworks that can optimize the training process in distributed settings. [10] addressed fundamental challenges such as communication efficiency and convergence guarantees by providing an overview of a comprehensive framework for distributed optimization. This work established key principles that inform hyperparameter fine-tuning in federated environments.

Several approaches have emerged for hyperparameter fine-tuning. For instance, consensus-based approaches can be adapted for hyperparameter optimization. These

approaches ensure that all clients converge on a common set of hyperparameters. They involve local optimization, hyperparameter updates, and consensus iterations.

Another type of approach is asynchronous optimization methods. These methods address the challenges regarding system heterogeneity. [43] developed an asynchronous federated optimization framework that allows clients to continuously stream local updates to the central server, reducing idle time and improving overall system efficiency. This approach proves particularly valuable for hyperparameter fine-tuning, where evaluation speeds may vary significantly across clients with different computational capabilities.

Federated hyperparameter fine-tuning gets more and more complicated to implement in more complex models such as neural networks. Neural networks can have many different tunable parameters, making them hard to manually tune effectively. That's where Neural Architecture Search (NAS) comes in hand. NAS is a technique that automates the design of neural networks. [44] introduced the Federated NAS (FedNAS) framework that enables collaborative architecture optimization. FedNAS leverages gradient-based NAS techniques such as Differentiable Architecture Search (DARTS) [45] to search for optimal architectures. In order to keep client data safe, clients share only gradients related to the current architecture, not raw data. This approach outperformed traditional FL approaches such as FedAvg, especially in Non-IID settings.

Moving onto meta-learning applications, one effective technique that has emerged is Model-Agnostic Meta-Learning (MAML). MAML was initially introduced by [46], which has since been adapted for FL scenarios. MAML works by learning an initial set of model parameters that can quickly adapt to new tasks with minimal gradient updates. Clients perform local inner-loop adaptations, and the server aggregates these updates in the outer loop to refine the global meta-model, creating an efficient framework for hyperparameter evaluation across diverse client environments.

Transfer learning techniques further enhance federated hyperparameter fine-tuning. These techniques can be effective in FL, especially where data availability may be limited. In [47], a federated transfer learning framework was proposed for healthcare applications that enables efficient data transfer across different organizations while maintaining client privacy. This framework proves especially valuable for hyperparameter fine-tuning in domains with limited or imbalanced data distributions.

Local adaptation techniques represent another important direction in federated hyperparameter fine-tuning. [48] proposed an adaptive local optimization framework that allows clients to adjust hyperparameters based on their specific data characteristics and computational resources. This client-specific approach reduces communication overhead without compromising model performance, enabling more efficient exploration of the hyperparameter space in federated environments. By allowing parameter adaptation at the client level, these approaches acknowledge the

heterogeneous nature of federated systems and provide mechanisms to accommodate varying data distributions and resource constraints. This granularity proves particularly valuable for hyperparameter fine-tuning, where different clients may benefit from different parameter settings depending on their local conditions.

Having examined specialized approaches for federated hyperparameter fine-tuning, the discussion now turns to FL frameworks tailored for specific scientific domains.

3.4 Federated Learning Adaptations on Different Domains

The development of effective federated learning frameworks, including hyperparameter fine-tuning techniques, depends significantly on the application domain and its specific requirements. Different domains present unique challenges that influence hyperparameter selection and tuning strategies. This section explores domain-specific approaches to FL, highlighting how application characteristics shape optimization needs.

The first domain that is going to be explored is computer vision. [49] thoroughly explained how FL can be adapted for real-world computer vision applications, such as image classification. In his work, leveraging distributed data sources without compromising privacy was highlighted as a main advantage of FL. [50] introduced FedPylot, a framework designed for real-time object detection. It addresses the challenges of non-IID data distributions and varying object scales across different clients by utilizing YOLOv7, a state-of-the-art real-time object detector. [51] proposed a federated framework using semantic segmentation. This approach focuses on incremental learning and adapting to new classes or data. His work demonstrated how hyperparameter settings influence a model's ability to incorporate new information while maintaining performance on existing classes.

Natural Language Processing (NLP) presents yet another important application domain for FL. Specifically, mobile keyboard prediction was one of the early applications of FL. [2] presented a FL approach for mobile keyboard prediction that improved personalization by utilizing local adaptation techniques. This approach improved next-word accuracy prediction across diverse user populations without exposing any user data. Moving onto text classification, [52] proposed a federated evaluation and tuning framework for on-device personalization. Their approach is particularly relevant for applications where data is generated and processed locally on devices.

Machine translation further illustrates the challenges of federated learning in NLP. [53] introduced a federated framework for multilingual neural machine translation utilizing adapters to enhance model efficiency. By leveraging adapter-based architectures, this approach efficiently accommodates different languages and dialects while reducing the need for extensive parallel data. The framework enables collab-

oration between multiple organizations with limited data, improving translation quality through federation. Effective hyperparameter fine-tuning proves essential for balancing model capacity, language coverage, and computational efficiency in this context.

Healthcare is another major application domain of FL. Many healthcare applications, such as medical imaging, require strict compliance with privacy regulations and suffer from data heterogeneity. [54] developed a FL framework for medical image analysis. It complies with privacy regulations such as Health Insurance Portability and Accountability Act (HIPAA), enabling secure collaboration between institutions. This is particularly important for tasks such as brain tumor segmentation. By enabling collaboration between institutions, more accurate and robust medical imaging models can be developed. Additionally, FL can also be applied for drug discovery. [55] proposed a FL framework for collaborative analysis in drug discovery, focusing on handling Non-IID data distributions. His framework speeds up the process of finding viable drug candidates by enabling institutions to collaborate. His work focuses on Quantitative Structure-Activity Relationship (QSAR) analysis, which relies heavily on collaboration between research institutions to accelerate development.

These diverse application domains highlight the need for flexible, domain-aware hyperparameter fine-tuning approaches in FL. This increasing need for FL frameworks motivated the frameworks development. The following section examines methodologies for evaluating the performance of these systems, with particular emphasis on hyperparameter fine-tuning assessment.

3.5 Performance Evaluation Methods

Evaluating FL systems can be difficult, due to their distributed nature, privacy constraints, and heterogeneous environments. This section explores the various methodologies and frameworks used to assess the performance of FL systems, with particular emphasis on hyperparameter fine-tuning evaluation.

Thorough evaluation of FL frameworks requires attention to several key considerations that directly impact hyperparameter fine-tuning. It is important to implement methods that provide comparisons across various federated settings. Furthermore, to allow correct replication of the experiments that were carried out by the FL system, it is required to provide extensive documentation. Documenting and sharing experimental configurations includes providing detailed descriptions of datasets, model hyperparameters, and training procedures. By enabling replication of the experiments, the framework's performance can be correctly validated. Another important assessment is a scalability assessment. Testing a framework on different numbers of clients or different scales of data can help understand its limitations on real-world applications and can also identify potential bottlenecks, ensuring overall robustness.

Another tool that can provide performance evaluation of a FL framework is convergence analysis. [56] proposed a framework that analyzes convergence behavior in FL. This framework introduced metrics that capture both local and global optimization trajectories. It also demonstrated how convergence rate can be affected by various client sampling strategies and the number of local update steps. By optimizing convergence rates through appropriate hyperparameter selection, FL systems can achieve better performance with fewer training rounds. Convergence metrics provide objective measures for comparing different hyperparameter configurations, helping identify settings that balance optimization speed with model quality. These metrics prove particularly valuable in communication-constrained environments, where minimizing training rounds delivers significant efficiency benefits.

Performance evaluation metrics can be combined with resource utilization metrics for a deeper understanding of the efficiency of a FL framework. These metrics are responsible for monitoring and analyzing resource consumption across various clients. Memory Utilization Profile (MUP) is one of those metrics. MUP tracks both peak and average memory usage during model training and can help identify potential bottlenecks. Another useful metric is Energy Consumption Rate (ECR). ECR evaluates the energy efficiency of training processes. It shows how much energy is consumed during model updates, particularly useful for resource-constrained environments such as mobile devices. Client Availability Impact (CAI) can also be utilized in conjunction with the other metrics. CAI evaluates how different patterns of client participation affect overall system performance and resource utilization. Understanding client availability helps optimize client selection strategies, ensuring that the most capable clients are utilized effectively to enhance training outcomes. By employing these resource utilization metrics, FL frameworks can achieve a better understanding of how various hyperparameter settings impact overall system efficiency. This ultimately leads to more effective FL, allowing for better resource management while maintaining model performance.

Recent advances in FL have produced various experimental frameworks that can evaluate and test FL systems. These simulation environments play a vital role in advancing research and development in FL. The LEAF framework, presented by [57], provides a platform specifically designed for federated settings. LEAF provides users with several benchmarking tools. It supports diverse data partitioning methods that reflect real-world scenarios, allowing researchers to simulate different data distributions across clients. The framework also includes tools to model various network conditions. These tools can help assess how communication delays and bandwidth limitations affect learning performance. Additionally, LEAF allows for the simulation of client participation patterns, helping to understand how client availability impacts training. Finally, LEAF establishes standardized protocols for performance evaluation, ensuring consistency and comparability across different experiments and optimization approaches.

FedML, introduced by [58], represents another important experimental framework facilitating both simulation and real-world deployment of federated systems. The FedML library supports deployment across various platforms, enabling researchers to test FL algorithms in diverse environments, including edge devices and cloud infrastructures. FedML provides tools for real-time monitoring of the performance of FL systems. These tools help assess model accuracy and resource utilization during actual deployments. It also provides functionalities for automating hyperparameter fine-tuning, streamlining the process of optimizing model performance without extensive manual intervention. Finally, FedML includes standardized benchmarks that facilitate consistent evaluation of different FL algorithms, ensuring comparability across studies and implementations.

Through these evaluation methodologies and experimental frameworks, researchers can systematically assess the performance of federated hyperparameter fine-tuning approaches, identifying their strengths, limitations, and practical applicability in real-world distributed systems.

Chapter 4

Methodology

This chapter presents the theoretical framework and design methodology of our federated hyperparameter fine-tuning system. The system aims to address the critical challenge of fine-tuning model hyperparameters in FL, while preserving data privacy and providing secure communication between participants and the server. The proposed system follows a FL architecture with two main components: a central server that coordinates the optimization process and multiple client nodes that perform local training and hyperparameter evaluation.

4.1 Hyperparameter fine-tuning process

Our framework utilizes a multi-phase hyperparameter fine-tuning process. First, the server randomly generates a diverse set of hyperparameter configurations from predefined parameter spaces stored in a configuration file:

$$C = \{c_1, c_2, \ldots, c_n \mid c_i \in \Theta\} \tag{4.1}$$

where C represents the set of configurations and Θ is the parameter space.

After the creation of the model parameter combinations in the server, they are shared with the clients. Each client then performs stratified k-fold cross-validation on their local data for all received configurations, computing performance metrics for each fold:

$$s_{i,j} = \frac{1}{K} \sum_{k=1}^{K} F1(c_i, D_{j,k}) \tag{4.2}$$

where $s_{i,j}$ is the F1 score for configuration i on client j, and $D_{j,k}$ represents the k-th fold of client j's data.

The F1 score is defined using the following formula:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.3}$$

where:

- $\text{precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$
- $\text{recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$

Once all the model configurations are evaluated, clients share their performance results with the server, which aggregates scores according to a weighted voting mechanism:

$$V_i = \sum_{j=1}^{N} s_{i,j} \cdot \frac{n_j}{n_{total}} \tag{4.4}$$

where V_i is the vote for configuration i, n_j is the number of samples at client j, and n_{total} is the total number of samples across all clients.

Upon retrieval of the performance scores, the server identifies the best overall performing configuration from the predefined set based on the aggregated votes:

$$c_{best} = \arg\max_{c_i \in C} V_i \tag{4.5}$$

Lastly, all clients train their local models using the identified best performing configuration, and a series of aggregation rounds starts.

If the user does not want to use any hyperparameter fine-tuning, our system also supports a baseline mode, where a default configuration is used for all clients. The default configuration uses the model's default parameters for aggregation. The hyperparameter fine-tuning phase is skipped, and the aggregation rounds start immediately. This serves as a control condition to evaluate the effectiveness of the implemented federated hyperparameter fine-tuning approach.

4.2 Aggregation Modes

Our framework supports multiple aggregation strategies to accommodate different FL scenarios. Starting with the weighted aggregation mode, it follows the FedAvg algorithm [1] where client contributions are weighted proportionally to their dataset sizes. This results in giving greater influence to clients with more extensive datasets:

$$\theta_{global} = \frac{\sum_{j=1}^{N} n_j \cdot \theta_j}{\sum_{j=1}^{N} n_j} \tag{4.6}$$

For neural network models, this aggregation is applied layer-wise:

$$\theta_{global,l} = \frac{\sum_{j=1}^{N} n_j \cdot \theta_{j,l}}{\sum_{j=1}^{N} n_j} \tag{4.7}$$

where $\theta_{global,l}$ represents the aggregated parameters for layer l.

The second aggregation mode that is used is unweighted aggregation. Complementing the weighted aggregation mode, in unweighted aggregation all clients contribute equally to the global model regardless of their dataset sizes:

$$\theta_{global} = \frac{1}{N} \sum_{j=1}^{N} \theta_j \tag{4.8}$$

The model parameter aggregation was later coupled with a residual learning mechanism that enhances model convergence and performance. This approach employs a strategy where each aggregation round's parameters are influenced by the aggregated parameters of the previous aggregation rounds. The residual mechanism is mathematically formulated as:

$$\theta_{global}^{t+1} = \alpha \cdot \theta_{aggregated}^{t} + \theta_{global}^{t} \tag{4.9}$$

where θ_{global}^{t+1} represents the updated global parameters at round $t+1$, $\theta_{aggregated}^{t}$ represents the newly aggregated parameters from clients at round t, θ_{global}^{t} represents the accumulated global parameters up to round t, and α is the learning rate that controls the contribution of new parameters.

The residual learning mechanism provides several advantages to the FL process. By considering historical parameter values, the model becomes more resistant to fluctuations caused by client-specific data distributions. Additionally, the model maintains a memory of previous training states, potentially increasing its performance. This residual learning mechanism allows customization of the learning rate parameter to control the balance between stability and adaptability, enabling optimization based on specific deployment requirements.

4.3 Security Framework

Building upon the aggregation mechanisms, the framework implements a comprehensive security architecture that addresses both data privacy and communication integrity concerns. The security framework employs a hybrid encryption approach that combines the strengths of both asymmetric and symmetric cryptography.

At first, RSA keys are exchanged (asymmetric encryption) between clients and the server. The server validates client identities through machine IDs that are embedded in message headers and maintained consistently throughout the session. Then, symmetric AES-256 keys are exchanged that are encrypted with the previously shared RSA keys for security. Every model update and parameter shared between clients and the server afterwards is encrypted with the appropriate symmetric key, leveraging the computational efficiency of symmetric encryption for data transmission. The system further enhances security by implementing MD5 checksums for integrity verification. These checksums allow recipients to confirm that data has not been altered during transmission, providing protection against both accidental corruption and malicious interference.

This hybrid encryption approach is integrated into the communication protocol of the framework. The communication protocol consists of three phases: the setup phase, the training phase, and the termination phase. Firstly, the setup phase establishes secure connections and authenticates participants before any model training begins. This is the phase where the hybrid encryption approach takes place. It concludes with the establishment of persistent socket connections and the initialization of thread-safe buffer structures to facilitate ongoing server-client communications.

After the setup phase, the training phase takes place. During the training phase, the communication protocol manages the secure exchange of model parameters and hyperparameter configurations. All data transmissions use AES-256 encryption with the previously established symmetric keys. Each transmission includes detailed headers containing MD5 checksums, allowing recipients to verify data integrity upon receipt. The protocol implements a synchronization mechanism through thread barriers and acknowledgment messages, ensuring all clients progress through the FL process at a coordinated pace.

Finally, the termination phase handles the conclusion of the FL process. Once the maximum number of aggregation rounds is reached, the server signals all clients to prepare for session termination. The protocol ensures that final model parameters are securely distributed to all participants and comprehensive performance metrics are reported and saved for later analysis. Clean-up procedures release all allocated resources, close socket connections, and securely terminate buffer managers to prevent memory leaks. The protocol includes proper notification of termination to all threads, ensuring they exit gracefully and preventing potential deadlocks or orphaned processes.

4.4 Experimental Tool

To evaluate the effectiveness of the federated hyperparameter fine-tuning approach, a comprehensive experimental tool is implemented with four distinct phases: the weighted aggregation phase, the unweighted aggregation phase, the no-hyperparameter-tuning phase, and the non-federated learning phase. Each phase executes the framework using different input parameters, such as the choice of aggregation method, enabling or disabling hyperparameter fine-tuning, and switching between centralized and decentralized learning. The purpose of the experimental framework is to provide the user with an appropriate benchmarking tool that facilitates comparative analysis across different FL scenarios. By employing this tool, the performance and efficiency of the proposed federated hyperparameter fine-tuning system were rigorously assessed, ensuring that the evaluation reflects realistic variations in training strategies and settings.

During the weighted aggregation phase, the framework first conducts hyperparameter tuning across all participating clients, allowing each client to evaluate multiple model configurations through k-fold cross-validation on their local data. After the conclusion of the hyperparameter fine-tuning and the decision of the best performing model hyperparameters to be used, the aggregation process starts. Client contributions to the aggregation process are weighted proportionally to their dataset sizes, giving greater influence to clients with more extensive and potentially more representative datasets.

Following this initial phase, the unweighted aggregation phase applies the previously determined best performing hyperparameters, skipping redoing the hyperparameter fine-tuning phase for efficiency. This phase employs an unweighted aggregation strategy, where all clients contribute equally to the global model, regardless of their dataset sizes. This phase serves as a comparative baseline to evaluate whether weighted aggregation can consistently outperform unweighted aggregation.

The subsequent no-hyperparameter-tuning phase establishes a critical baseline by using the default model hyperparameters. During this phase, the framework uses a model with default parameters and proceeds to weighted aggregation. The results provided by this phase can prove whether the implemented hyperparameter fine-tuning is actually effective by comparing the results using the default model against those from the fine-tuned phases.

During the final phase, a centralized client is used that trains on the combined datasets from all clients. This centralized approach represents the theoretical upper bound of performance if data centralization were possible, serving as a reference point.

4.5 Performance Monitoring and Visualization

The framework incorporates a comprehensive performance monitoring system that tracks and visualizes multiple metrics throughout the FL process. This system operates at both the individual client level and the global aggregation level, providing insights into how the learning process evolves across iterations.

For each client, the framework records F1 scores for both training and testing datasets across all aggregation rounds. These client-specific metrics are timestamped and associated with the corresponding model parameters, providing insight into how individual clients progress through the learning process. This tracking allows identification of potential issues with specific clients, such as overfitting or difficulties with certain data distributions. At the global level, the framework calculates and records the F1 score of every aggregation.

The visualization component generates graphical representations of the learning process, including performance curves showing the progression of metrics across aggregation rounds. These visualizations include the performance of weighted and unweighted aggregation, aggregation with default model parameters, and the performance of the non-federated process. It also includes the performance curves of each client's F1 score on its train and test sets, respectively. All performance data is stored in a structured CSV format so that it can be analyzed later on.

4.6 Communication Infrastructure Implementation

The system's communication infrastructure is implemented through a multi-layered architecture that combines C++ for core functionality with Python bindings through Cython [59]. This infrastructure consists of several core network and architecture components.

The network layer consists of The Net_lib helper module. This module manages message headers, with methods for constructing and parsing headers that contain metadata like MD5 checksums for integrity verification, machine identification codes, and file size information. It also provides automated socket cleanup procedures, acting as a safeguard against memory and other resource leaks.

The system also implements a shared buffer mechanism that enables concurrent client connections while maintaining data integrity. The SharedBuffer class provides a robust implementation for storing and managing data with atomic operations for read and write access, preventing race conditions that could otherwise compromise data consistency. This implementation utilizes mutex locks, ensuring thread safety. The buffer dynamically resizes as needed, optimizing memory usage while accommodating variable-sized model parameters.

The BufferManager class coordinates multiple shared buffers. This manager implements efficient buffer allocation strategies that minimize fragmentation and op-

timize memory usage. By centralizing buffer management, the system ensures consistent access patterns and simplified resource cleanup.

Following the network components, architecture components divide into server- and client-related. Starting with the server, the TCPServer class can handle multiple concurrent client connections through a thread-per-client model. It also maintains a central shared buffer for aggregating client updates. TCPServer provides individual buffers for each client that are managed through the BufferManager.

Continuing with the client component, it handles the server connection establishment and maintenance. It implements secure file transfer with integrity verification through MD5 checksums. Additionally, it manages the transmission of model updates and hyperparameter configurations. The client component employs a thread-safe signaling mechanism to coordinate between the Python-based model training process and the C++ communication layer. This design allows the ML component to execute training independently while the communication layer handles parameter exchange with the server. When local training completes, a notification mechanism alerts the communication thread to transmit the updated parameters. Similarly, when aggregated parameters are received from the server, the communication layer signals the Python component to incorporate these updates into the local model. This decoupled architecture improves performance by allowing computation and communication to operate concurrently where possible.

The communication infrastructure implements several fault tolerance mechanisms to handle network disruptions and client failures. Each message exchange includes acknowledgment protocols to confirm successful transmission. In cases where a client becomes unresponsive, the server can detect the failure through timeout mechanisms and adjust the aggregation process accordingly. Additionally, the communication infrastructure employs a barrier synchronization pattern to coordinate client-server interactions across the distributed system.

This methodology chapter has established the theoretical foundation and design principles for our federated hyperparameter fine-tuning framework. The proposed approach addresses the core challenges of distributed optimization through a multi-phase process that combines secure hyperparameter exploration, weighted aggregation mechanisms, and residual learning enhancement. The integration of cryptographic security measures ensures data privacy throughout the optimization process, while the comprehensive experimental tool provides robust benchmarking capabilities. Together, these methodological components form a cohesive framework that balances performance optimization, privacy preservation, and practical implementation requirements. The following implementation chapter will detail how these theoretical constructs are translated into a working system, providing the technical specifications and architectural decisions necessary for real-world deployment.

Chapter 5

Implementation

This chapter presents the detailed algorithms and implementation strategies employed in the federated hyperparameter fine-tuning system. The implementation transforms the theoretical framework presented in the previous chapter into a working system. The chapter follows the logical progression of the FL process, beginning with the establishment of secure communication channels, proceeding through the hyperparameter optimization phases, and concluding with the model aggregation strategies that enable collaborative learning.

5.1 Initial Encryption Process and Hyperparameter fine-tuning

The FL process begins with the establishment of secure communication channels between all participants, a critical prerequisite to ensure client security. The security establishment process must balance strong cryptographic guarantees with computational efficiency, ensuring that the overhead of security measures does not compromise the performance of the machine learning components.

The encryption initialization process implements a hybrid cryptographic approach that leverages the complementary strengths of asymmetric and symmetric encryption systems. This approach can be seen in Algorithm 1.

The secure communication setup process operates through a carefully orchestrated handshake protocol. Initially, both client and server generate RSA key pairs, with each party retaining their private key while preparing to share their public key. The parties then exchange public RSA keys, establishing the foundation for secure communication without revealing sensitive private key material. Using the clients' public keys, the server can securely transmit symmetric encryption keys to the clients. Once symmetric keys are sent, all subsequent information exchange utilizes efficient AES encryption, providing both security and computational efficiency for

Algorithm 1 Client-Server Secure Communication Setup

Require: Security parameters κ, client C, server S
Ensure: Established secure channel $\mathcal{C}$
1: **function** InitEncryption(κ)
2: $(pk_C, sk_C) \leftarrow$ GenRSAKeys(κ) ▷ Generate client key pair. Server key pair is also generated, but on server side
3: $pk_S \leftarrow C \rightarrow S : pk_C$ ▷ Exchange public keys
4: $K_{sym} \leftarrow$ Decrypt($sk_C, E(pk_C, K_{sym})$) ▷ Decrypt the receiving symmetric key using clients private RSA key
5: $\mathcal{C} \leftarrow K_{sym}$ ▷ K_{sym} can now be used to enrypt messages
6: **return** $\mathcal{C}$
7: **end function**

the intensive data exchanges required during federated learning.

With secure communication channels established, the federated hyperparameter fine-tuning takes place. It consists of four main phases. In the first phase, the server defines hyperparameter combinations for a selected ML algorithm. These combinations are configured through a JSON file containing valid hyperparameter search spaces, as shown in Algorithm 2.

Algorithm 2 Hyperparameter Combinations Configuration

Require: Model type $\mathcal{M}$, configuration file $\mathcal{F}$
Ensure: Combinations $\Theta_{\mathcal{M}}$
1: **function** ReadConfigParams($\mathcal{M}, \mathcal{F}$)
2: $\Phi \leftarrow$ LoadJSON($\mathcal{F}$)
3: **if** $\mathcal{M} = \emptyset$ **then**
4: **return** $\emptyset$ ▷ Invalid model type
5: **end if**
6: **return** $\Theta_{\mathcal{M}} \leftarrow \Phi[\mathcal{M}]$
7: **end function**

For example, for a LogReg model, the configuration file includes solvers, penalties, regularization strengths, and other relevant parameters that can produce valid logistic regression models.

Following the generation of the hyperparameter combinations, phase two takes place. As shown in Algorithm 3, each client receives the hyperparameter combinations from the server and performs stratified k-fold cross-validation locally.

After every model initialization during the k-fold loop, one aggregation round takes place. The aggregated parameters are then incorporated into the model, and the model is trained on the specific data split. This aggregation round helps the model to benefit from the collective knowledge of all clients, ensuring that each client

Algorithm 3 Distributed K-Fold Cross-Validation

Require: Local dataset $\mathcal{D}$, splits k, configs Θ
Ensure: Validation scores $\mathbf{s}$

function KFoldLoop($\mathcal{D}, k, \Theta$)
  $\mathcal{S} \leftarrow$ StratifiedKFold($\mathcal{D}, k$)
  for $(X_{tr}, y_{tr}), (X_{val}, y_{val}) \in \mathcal{S}$ **do**
    $X'_{tr} \leftarrow$ StandardScaler(X_{tr}) ▷ Scale data using StandardScaler
    for $j \leftarrow 1$ **to** $|\Theta|$ **do**
      $\theta_j \leftarrow \Theta[j]$
      $\mathcal{M}_j \leftarrow$ InitModel(θ_j)
      $\mathcal{M}_j \leftarrow$ Train(X'_{tr}, y_{tr}) ▷ Fit model and update model params with aggregated ones
      $s_j \leftarrow$ F1Score($\mathcal{M}_j, X_{val}, y_{val}$)
      $\mathbf{s}[j] \leftarrow \mathbf{s}[j] + s_j/k$
    end for
  end for
  return s
end function

model starts from a shared, optimized baseline. Then, the F1 score gets extracted and saved.

Following the k-fold cross-validation phase, phase three takes place. After clients complete local validation, they send their validation scores to the server, and then the server aggregates these scores to select the best performing hyperparameter configuration from the set of the produced hyperparameter configurations. This voting phase is presented in Algorithm 4.

The federated voting mechanism implements weighted voting in order to account for the varying dataset sizes across participating clients. This approach recognizes that clients with larger datasets typically provide more statistically reliable performance estimates, and therefore their vote should have proportionally greater influence. After aggregating all the votes, the index that points to the decided hyperparameter combination is returned. This index is shared back to the clients and is used to find the appropriate combination.

The final phase of the hyperparameter optimization process involves training all client models using the identified best performing hyperparameter combination. This phase marks the transition from hyperparameter exploration to model exploitation and is the point when aggregation rounds start.

Algorithm 4 Federated Voting Mechanism

Require: Client scores $\{\mathbf{s}_i\}_{i=1}^{n}$, data sizes $\{n_i\}_{i=1}^{n}$
Ensure: Best performing parameters j^* ▷ j^* indicates the best performing model configuration in clients stored models array.

function AGGREGATESCORES($\{\mathbf{s}_i\}, \{n_i\}$)
    $N \leftarrow \sum_{i=1}^{n} n_i,\ \mathbf{v} \leftarrow []$ ▷ Calculate total samples N and initialize $\mathbf{v}$ as an empty list
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** $|\mathbf{s}_i|$ **do**
            $\mathbf{v}[j] \leftarrow \mathbf{v}[j] + \mathbf{s}_i[j] \cdot n_i/N$
        end for
    end for
    $j^* \leftarrow \arg\max_j \mathbf{v}[j]$
    return j^*
end function

5.2 Model Aggregation Strategies

The model aggregation component represents the mathematical core of the FL system, where individual client contributions are combined to create improved global models. The implementation supports multiple aggregation strategies to accommodate different FL scenarios and data distribution characteristics.

Currently, the system implements two aggregation strategies: weighted and unweighted aggregation. In weighted aggregation, model parameters are weighted by the proportion of data each client holds, following the FedAvg algorithm [1]. In unweighted aggregation, equal weight is given to each client's model parameters. Both aggregation strategies can be optionally enhanced with a residual learning mechanism that maintains a global parameter state across aggregation rounds. Algorithm 5 shows how model parameters are aggregated at the server.

This residual learning mechanism enhances the aggregation process by incorporating historical parameter values into the current update. This approach is controlled by a learning rate parameter (`LR`, default: 0.10) that determines the contribution of new aggregated parameters to the global model state. The benefits of this approach include improved stability in heterogeneous data environments, smoother convergence trajectories, and reduced susceptibility to local data peculiarities. The mechanism can be selectively enabled or disabled through the learning rate parameter, making it a flexible addition to both weighted and unweighted aggregation modes.

Algorithm 5 Model Parameter Aggregation

Require: Client parameters $\{\theta_i\}_{i=1}^{n}$, aggregation mode m, learning rate η
Ensure: Global parameters θ_G

function AGGREGATEPARAMETERS($\{\theta_i\}, m, \eta$)
  $N \leftarrow \sum_{i=1}^{n} |\mathcal{D}_i|$ ▷ Total samples
  $\theta_G \leftarrow \mathbf{0}$
  for $\theta^{(k)} \in \theta_1$ **do** ▷ Iterate over parameter indices
    if $m =$ "weighted" **then**
      $\theta^{(k)} \leftarrow \frac{1}{N} \sum_{i=1}^{n} |\mathcal{D}_i| \theta_i^{(k)}$
    else
      $\theta^{(k)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \theta_i^{(k)}$
    end if
    if $\eta > 0$ **then**
      $\theta_G^{(k)} \leftarrow \theta_G^{(k)} + \eta \theta^{(k)}$
    end if
  end for
  return θ_G
end function

5.3 Communication Layer

The communication layer represents the technical foundation that enables all distributed operations in the FL system. The main goal of this layer is to coordinate multiple distributed processes. The implementation leverages C++ with TCP sockets to establish robust connections between server and client nodes, providing the low-level networking capabilities necessary for intensive data exchange while maintaining compatibility with the Python-based ML components.

The communication architecture employs multithreading to handle multiple client connections simultaneously, with carefully designed synchronization mechanisms to manage shared resources without compromising data integrity. This concurrent design enables the server to coordinate with numerous clients efficiently while ensuring that the federated learning process proceeds in a coordinated manner across all participants.

The communication layer consists of two primary components: the client communication component and the server communication component. Each component has distinct responsibilities and functionalities.

The client component is responsible for local model training and parameter exchange with the server, serving as an interface. It maintains two buffer structures: one dedicated to parameters that need to be transmitted to the server and another for aggregated parameters received from the server. This dual-buffer architecture follows a simple communication protocol with the server. In every communication

round, before sending a message, a header message is sent to verify the identity of the client. After receiving an ACK message from the server, the actual message is sent. After receiving an ACK from the server, the client waits for the server to send a message.

Complementing the client component, the server component coordinates the FL process by managing multiple client connections, collecting model parameters, and distributing aggregated results. It employs a multithreaded architecture to handle client connections concurrently. Firstly, it accepts connections from all clients and creates threads for every new connection. Then, after global parameters are initialized, client-server communication can begin. The server waits for the client to send the header and main message and replies with ACK messages after successfully receiving these messages. Then it ends the communication cycle by replying to the clients with the respective header and main messages.

Critical to the server's coordination capabilities is its implementation of a barrier synchronization mechanism that ensures all clients progress through the FL process together. This synchronization prevents faster clients from advancing too far ahead of slower participants, ensuring consistent convergence behavior.

The complete implementation presented in this chapter demonstrates a comprehensive and robust framework that successfully addresses the complex challenges of distributed hyperparameter fine-tuning. The system's key innovations work together to create a platform with practical deployment considerations. The communication infrastructure, with its hybrid encryption approach and coordinated synchronization mechanisms, ensures privacy preservation while enabling effective collaborative learning. These implementation contributions provide the technical foundation for the experimental validation presented in the subsequent chapter, where the effectiveness of these approaches is demonstrated across various datasets.

Chapter 6

Evaluation

This chapter presents a comprehensive evaluation of the proposed FL framework, with a particular focus on the effectiveness of federated hyperparameter fine-tuning mechanisms and aggregation strategies. The evaluation is designed to assess how different components of the FL system perform across diverse datasets and model architectures, providing insights into the practical applicability and performance characteristics of the framework. First, the five diverse datasets selected for experimentation are described, covering different domains including demographic analysis, financial services, consumer product reviews, and medical diagnostics. These datasets present varying challenges in terms of class balance, feature complexity, and sample size, providing a robust test bed for evaluating the FL framework. Next, the experimental methodology is analyzed, including the federated system architecture, the machine learning models evaluated, and the specific experimental conditions. The results section presents a detailed analysis of performance across all experimental configurations, measured using the F1 score metric. Through this analysis, patterns in performance across different models, datasets, and FL aggregation strategies are identified.

6.1 Datasets

The evaluation of the proposed federated hyperparameter fine-tuning approach required careful selection of datasets. Several factors decided which datasets were finally used, such as scientific domain, dataset size, and data heterogeneity. After considering these factors, five diverse datasets were selected.

The first one was the **UCI Adult Census Income (AD)** dataset. It is a widely used dataset from the UCI Machine Learning Repository [60], containing demographic information extracted from the 1994 Census database. It consists of approximately 48,842 instances with 14 attributes, including both categorical and numerical features such as age, education, occupation, and marital status. The

classification task in this case is to predict whether an individual's income exceeds $50,000 per year.

The second one was the **UCI Bank Marketing (B)** dataset. This dataset relates to direct marketing campaigns of a Portuguese banking institution [61]. It contains 45,211 instances with 16 features, including client data (age, job, marital status), campaign information, and economic indicators. Here, the classification task involves predicting if a client will subscribe to a term deposit.

The third one was the **UCI Credit Card Fraud (C)** dataset. This is a highly imbalanced dataset containing credit card transactions, with the task of identifying bad credit risks [62]. It contains 1000 transactions with 20 features. This dataset was specifically chosen because of its high class imbalance and its small size for easy testing.

The fourth was the **Amazon Software Reviews (AM)** dataset. This is a subset of the Amazon review dataset [63], focusing on software product reviews. It contains 67,700 samples with 10 features, including review score, review time, helpfulness, and others. The classification task is to predict a positive or negative score. Some features in this dataset that included text data were omitted to avoid skewing results and overcomplicating the classification process. Containing data from real users, this dataset reflects real-world behaviors, preferences, and interactions, making it a strong candidate for our experiments.

The final dataset that was used was the **UCI Breast Cancer Wisconsin (BC)** dataset. A medical dataset for diagnosing breast cancer [64], containing 569 instances with 30 features computed from digitized images of fine needle aspirate of breast mass. Its features describe characteristics of cell nuclei present in the images. In this case, the binary classification task is to distinguish between benign and malignant tumors.

The framework was evaluated using three clients. Each dataset that was used was first divided into three non-overlapping subsets and distributed among the three client nodes. Each of these subsets was of a different size to simulate real scenarios where each client has a different chunk size of data. Next, each client split was further partitioned into training (80%) and test (20%) sets using stratified sampling to maintain the original class distribution. A consistent random seed was also used (seed=42) for reproducibility.

After loading and splitting the dataset for the clients, the following data preprocessing pipeline was applied. First, data rows were sorted by index, and rows with missing values were dropped entirely. Next, the target column was extracted and removed from the feature set. Then, after sorting the unique values, the target values were encoded using the sklearn class LabelEncoder. Following the preprocessing of the target values, the categorical features were also encoded using the LabelEncoder for each categorical column. Any unknown value found in the test data was mapped to a new class. During model training/evaluation phases, nu-

merical features were standardized using the StandardScaler class from sklearn. Finally, class imbalance was addressed by computing balanced sample weights that were used during model training, utilizing the compute_sample_weight function from sklearn. To ensure consistent evaluation across all experimental conditions, model performance was evaluated on the centralized test set. The centralized test set is a concatenation of the predictions and the true values of the client models that clients continuously send to the server.

6.2 Experimental Methodology

The experimental methodology was designed to compare different aggregation strategies and assess the impact of the residual learning mechanism across various ML models and datasets. The experimental framework consisted of a central server and three client nodes.

The evaluation encompassed four distinct ML models, each representing different algorithmic approaches and optimization characteristics relevant to FL applications. **Gaussian Naive Bayes (GNB)** serves as a probabilistic baseline, implementing Bayes' theorem with strong independence assumptions between features. **Logistic Regression (LogReg)** represents linear classification approaches, using a logistic function to model the probability of binary outcomes. With its convex formulation and well-understood convergence guarantees, LogReg provides a stable baseline for evaluating aggregation strategies. **Stochastic Gradient Descent (SGD)** implements linear classification optimized through stochastic gradient descent, introducing the complexities of iterative optimization in distributed settings. **Multi-Layer Perceptron (MLP)** represents neural network approaches with configurable hidden layer architectures and activation functions, providing the most complex optimization landscape and testing the framework's ability to handle high-dimensional parameter spaces.

For each model and dataset combination, four distinct experimental conditions were evaluated. At first, weighted aggregation was tested after performing hyperparameter fine-tuning. Then, using the best performing model configuration that was chosen during the first phase, unweighted aggregation was tested. In order to compare our FL approach with traditional centralized ML, those best performing model parameters were also applied to a centralized setting, using the entire dataset. Finally, weighted aggregation was performed using default model parameters to determine whether our hyperparameter fine-tuning approach provides meaningful improvements.

To assess the standalone contribution of residual learning, all experimental conditions were evaluated both with and without it. The residual learning rate parameters were carefully selected based on preliminary experiments, with specific values indicated in the results tables to ensure reproducibility.

Performance evaluation employed the F1 score as the primary metric, calculated from the combined predictions of all three client models on their respective test sets.

6.3 Results and Analysis

The following experimental results provide comprehensive evidence for the effectiveness of the proposed federated hyperparameter fine-tuning framework. Tables 6.1 and 6.2 present the complete experimental results, showing F1 scores across all experimental conditions for five datasets and four machine learning models, with and without the residual learning mechanism. The best-performing approaches for each model-dataset combination are highlighted in bold, and the second-best are underlined. LR stands for learning rate.

Table 6.1: Performance Comparison Across Models with Residual Path(**Best** F1 scores Weighted/Unweighted/NoFed/NoTune)

Model	Dataset	F1(Weighted/Unweighted/NoFed/NoTune)	Uses Default Model
GNB(LR=0.05)	AD	**0.513**/**0.513**/0.406/0.511	
	B	**0.614**/0.611/0.599/0.580	
	C	**0.64**/0.623/0.589/0.602	
	AM	0.637/**0.651**/0.577/0.581	
	BC	**0.945**/**0.945**/0.923/**0.945**	
LogReg(LR=0.10)	AD	0.479/0.479/**0.482**/0.261	
	B	0.638/0.638/**0.645**/0.598	
	C	0.577/0.577/**0.618**/0.469	
	AM	0.613/0.613/**0.616**/0.613	✓
	BC	0.964/0.964/**0.965**/0.964	✓
SGD(LR=0.10)	AD	**0.507**/**0.507**/0.026/0.473	
	B	**0.651**/0.648/0.608/0.633	
	C	0.604/0.611/0.576/**0.620**	
	AM	0.621/0.621/**0.629**/0.607	
	BC	**0.976**/**0.976**/0.943/0.965	
MLP(LR=0.10)	AD	**0.475**/0.467/0.450/0.445	✓
	B	0.627/**0.630**/0.617/**0.630**	
	C	0.539/0.517/0.533/**0.590**	
	AM	0.663/0.663/0.666/**0.669**	
	BC	**0.976**/**0.976**/**0.976**/**0.976**	

Dataset codes - AD: UCI Adult Census Income, B: UCI Bank Marketing, AM: Amazon Software Reviews, C: UCI Credit Card Fraud, BC: UCI Breast Cancer Wisconsin

The above results demonstrate several notable patterns. First, for GNB, the residual learning mechanism significantly improved performance, particularly on complex datasets like C, where F1 scores increased from 0.597 to 0.64 with weighted

Table 6.2: Performance Comparison Across Models without Residual Path(**Best** F1 scores Weighted/Unweighted/NoFed/NoTune)

Model	Dataset	F1(Weighted/Unweighted/NoFed/NoTune)	Uses Default Model
GNB	AD	**0.511**/0.510/0.406/**0.511**	
	B	0.582/0.588/**0.599**/0.580	
	C	0.597/0.596/0.589/**0.602**	
	AM	**0.581**/**0.581**/0.577/**0.581**	
	BC	**0.945**/**0.945**/0.923/**0.945**	
LogReg	AD	0.479/0.479/**0.482**/0.261	
	B	0.638/0.638/**0.645**/0.598	
	C	0.582/0.582/**0.618**/0.469	
	AM	0.613/0.613/**0.616**/0.613	✓
	BC	0.964/0.964/**0.965**/0.964	✓
SGD	AD	**0.482**/0.481/0.026/0.473	
	B	0.566/0.575/0.608/**0.633**	
	C	**0.661**/0.640/0.576/0.620	
	AM	0.612/0.612/**0.629**/0.607	
	BC	0.953/0.954/0.943/**0.965**	
MLP	AD	0.449/**0.455**/0.454/0.450	
	B	**0.630**/0.629/0.617/**0.630**	✓
	C	0.585/**0.615**/0.533/0.590	
	AM	**0.669**/**0.669**/0.666/**0.669**	
	BC	**0.976**/**0.976**/**0.976**/**0.976**	

Dataset codes - AD: UCI Adult Census Income, B: UCI Bank Marketing, AM: Amazon Software Reviews, C: UCI Credit Card Fraud, BC: UCI Breast Cancer Wisconsin

aggregation. This suggests that residual learning helps probabilistic models maintain more stable updates during aggregation. However, on simpler tasks like BC, both approaches achieved identical performance (0.945), indicating that residual learning benefits are most pronounced on challenging classification problems.

For LogReg, performance remained consistent regardless of residual learning, with nearly identical F1 scores across all experimental conditions. This stability suggests that LogReg's linear decision boundary is less sensitive to the residual mechanism, with the non-federated baseline consistently outperforming federated approaches by a small margin.

The SGD exhibited only marginal improvements with residual learning. Unlike GNB—where the two tables are directly comparable (since the same GNB models are used across both)—the SGD results cannot be compared in the same way. The observed differences between the tables arise primarily from variations in model

parameter selection, rather than the influence of residual learning. Thus, the true effect of residual learning on SGD is negligible.

The MLP exhibited performance gains from residual learning in certain scenarios. On the AD dataset, both the NoTune approach and the hyperparameter fine-tuned approach used the default model parameters. However, the tuned model in the hyperparameter fine-tuned approach achieved higher accuracy—demonstrating the benefit of residual learning. Nevertheless, on other datasets, hyperparameter tuning failed to consistently outperform the default model. This suggests that the parameter space was not sufficiently optimized to allow the tuning mechanism to discover a better solution.

The comparison between weighted and unweighted aggregation revealed only minor performance differences across most models. The most notable gap occurred in GNB with residual learning enabled, where accuracy improved from 0.637 to 0.651 on the AM dataset. These results indicate that—given the relatively balanced client distributions in our studied datasets (except the C dataset)—the choice of weighting scheme had minimal influence on overall model performance.

The "Uses Default Model" indicator reveals that several high-performing configurations did not require hyperparameter fine-tuning, particularly for Logistic Regression. This suggests that in certain scenarios, default parameters may suffice. However, for more complex models like the MLP, hyperparameter fine-tuning rarely improved upon default configurations, highlighting the challenges in optimizing such architectures. In our implementation, the hyperparameter fine-tuning mechanism proved insufficient to identify superior MLP configurations. Furthermore, in most cases, hyperparameter fine-tuned models outperformed the default models, proving that hyperparameter fine-tuning is an important component of FL.

These comprehensive results demonstrate that the proposed federated hyperparameter fine-tuning framework provides meaningful performance improvements across diverse models and datasets. The evidence supports the framework's practical utility while identifying specific areas where future research can further enhance its effectiveness, particularly in the optimization of complex neural networks and the refinement of aggregation strategies for different data distribution scenarios.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presents a comprehensive framework for federated hyperparameter fine-tuning. Our framework addresses the challenge of optimizing ML models in federated settings. By employing a multi-phase approach, our framework effectively performs hyperparameter fine-tuning. Alongside the hyperparameter fine-tuning, an optional residual learning mechanism is implemented. This residual mechanism proved particularly beneficial for simpler models on complex datasets, enhancing convergence stability in heterogeneous environments. Additionally, to ensure privacy, our framework implements a hybrid encryption approach (RSA for key exchange, AES for data transmission) with MD5 checksum verification that ensures both security and efficiency.

Through systematic evaluation across five diverse datasets and four machine learning algorithms, our approach demonstrates that federated hyperparameter fine-tuning can achieve performance competitive with centralized methods while preserving secure data transmission and client data privacy. Performance improvements of up to 45.5% in F1 score compared to default configurations demonstrate the substantial benefits of systematic hyperparameter exploration in federated settings. Notably, our federated approach occasionally outperformed the implemented centralized baseline. Although, to better understand the specific conditions and contexts where federated learning outperforms centralized learning (and vice versa), as well as to uncover the key factors driving these performance differences, further research and experimentation are needed.

In summary, our framework proves that hyperparameter fine-tuning can be effectively implemented in federated environments without compromising performance or privacy. As regulatory frameworks emphasize data protection and computing moves toward edge devices, our framework provides a practical foundation for privacy-preserving ML optimization. The systematic methodology and consis-

tent improvements across different models validate the approach's robustness and practical applicability, contributing both theoretical insights and implementation guidance for future FL systems.

7.2 Limitations and Future Work

The current implementation of hyperparameter fine-tuning has several limitations that guide our future work directions. Currently, the framework supports only traditional ML models (GNB, SGD, LogReg, MLP), limiting its applicability in scenarios requiring deep learning. We plan to extend support to transformers and deep neural networks to broaden the framework's use cases. Furthermore, the existing hyperparameter fine-tuning mechanism uses a static config file in order to define the parameter space that is later used to generate model configurations. This limitation will be addressed by implementing dynamic JSON configuration generation, enabling more flexible and personalized settings. Another area for improvement involves the current implementation of residual learning, which utilizes a static learning rate, limiting its application to more complex models. This will be resolved by introducing dynamic learning rates that self-adjust based on model performance. Finally, while FedAvg serves as a reliable baseline aggregation strategy, it may not perform optimally in heterogeneous or challenging federated scenarios. To improve robustness, we will incorporate more sophisticated aggregation strategies such as FedProx [12] for better convergence under system heterogeneity and SCAFFOLD [31] for reducing client drift.

Chapter 8

List of Acronyms

Acronyms used throughout this work:

GNB Gaussian Naive Bayes

LogReg Logistic Regression

SGD Stochastic Gradient Descent

MLP Multi-Layer Perceptron

HFL Horizontal Federated Learning

VFL Vertical Federated Learning

FTL Federated Transfer Learning

SMPC Secure Multi-Party Computation

DPM Distributed Mini-Batch Algorithm

FL Federated Learning

DP Differential Privacy

ML Machine Learning

HE Homomorphic Encryption

FedAvg Federated Averaging

GP Gaussian Process

TPE Tree-Structured Parzen Estimator

NAS Neural Architecture Search

MAML Model-Agnostic Meta-Learning

CCPA California Consumer Privacy Act

GDPR General Data Protection Regulation

Non-IID non-independent and identically distributed

FedAFK Personalized Federated Learning with Adaptive Feature Aggregation and Knowledge Transfer

IoT Internet of Things

TEEs Trusted Execution Environments

SCAFFOLD Stochastic Controlled Averaging for Federated Learning

EI Expected Improvement

PI Probability of Improvement

UCB Upper Confidence Bound

FedNAS Federated NAS

DARTS Differentiable Architecture Search

NLP Natural Language Processing

HIPAA Health Insurance Portability and Accountability Act

QSAR Quantitative Structure-Activity Relationship

MUP Memory Utilization Profile

ECR Energy Consumption Rate

CAI Client Availability Impact

AD UCI Adult Census Income

B UCI Bank Marketing

C UCI Credit Card Fraud

AM Amazon Software Reviews

BC UCI Breast Cancer Wisconsin

Bibliography

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

[2] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *CoRR*, vol. abs/1811.03604, 2018.

[3] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *J. Mach. Learn. Res.*, vol. 13, p. 165–202, Jan. 2012.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989.

[5] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, (USA), p. 10, USENIX Association, 2004.

[6] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982.

[7] C. Dwork, "Differential privacy," *International Colloquium on Automata, Languages, and Programming*, 2006.

[8] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *CoRR*, vol. abs/1712.07557, 2017.

[9] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawit, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar,

M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. Theertha Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, *Advances and Open Problems in Federated Learning*. Now Foundations and Trends, 2021.

[10] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, p. 50–60, May 2020.

[11] K. Yin and J. Mao, "Personalized federated learning with adaptive feature aggregation and knowledge transfer," 2024.

[12] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," *CoRR*, vol. abs/1812.06127, 2018.

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

[14] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), p. 1175–1191, Association for Computing Machinery, 2017.

[15] Y. Zhang, D. Zeng, J. Luo, Z. Xu, and I. King, "A survey of trustworthy federated learning with perspectives on security, robustness and privacy," in *Companion Proceedings of the ACM Web Conference 2023*, WWW '23 Companion, (New York, NY, USA), p. 1167–1176, Association for Computing Machinery, 2023.

[16] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *CoRR*, vol. abs/1903.03934, 2019.

[17] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *CoRR*, vol. abs/1712.01887, 2017.

[18] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[19] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, IEEE, May 2019.

[20] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *CoRR*, vol. abs/1806.00582, 2018.

[21] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[22] H. H. Yang, Z. Liu, T. Q. S. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 317–333, 2020.

[23] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *CoRR*, vol. abs/2003.02133, 2020.

[24] N. Rieke, J. Hancox, W. Li, F. Milletarì, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, S. Ourselin, M. Sheller, R. M. Summers, A. Trask, D. Xu, M. Baust, and M. J. Cardoso, "The future of digital health with federated learning," *npj Digital Medicine*, vol. 3, Sept. 2020.

[25] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, pp. 513–535, Feb 2023.

[26] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, p. 1622–1658, 2021.

[27] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, B. A. y Arcas, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data, S. N. Diggavi, H. Eichner, A. Gadhikar, Z. Garrett, A. M. Girgis, F. Hanzely, A. Hard, C. He, S. Horváth, Z. Huo, A. Ingerman, M. Jaggi, T. Javidi, P. Kairouz, S. Kale, S. P. Karimireddy, J. Konečný, S. Koyejo, T. Li, L. Liu, M. Mohri, H. Qi, S. J. Reddi, P. Richtárik, K. Singhal, V. Smith, M. Soltanolkotabi, W. Song, A. T. Suresh, S. U. Stich, A. Talwalkar, H. Wang, B. E. Woodworth, S. Wu, F. X. Yu, H. Yuan, M. Zaheer, M. Zhang, T. Zhang, C. Zheng, C. Zhu, and W. Zhu, "A field guide to federated optimization," *CoRR*, vol. abs/2107.06917, 2021.

[28] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *CoRR*, vol. abs/2003.00295, 2020.

[29] M. Khodak, R. Tu, T. Li, L. Li, M.-F. Balcan, V. Smith, and A. Talwalkar, "Federated hyperparameter tuning: challenges, baselines, and connections to weight-sharing," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, (Red Hook, NY, USA), Curran Associates Inc., 2021.

[30] W. Chen, S. Horváth, and P. Richtárik, "Optimal client sampling for federated learning," *CoRR*, vol. abs/2010.13723, 2020.

[31] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: stochastic controlled averaging for federated learning," in *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, JMLR.org, 2020.

[32] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," 2020.

[33] T. Wang, Z. Zheng, and F. Lin, "Federated learning framework based on trimmed mean aggregation rules," *Expert Systems with Applications*, vol. 270, p. 126354, 2025.

[34] X. Jiang, T. Wen, Z. Yang, L. Wu, Y. Chen, S. Sun, Y. Wang, and M. Liu, "Robust federated learning against noisy clients via masked optimization," 2025.

[35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 2, 2012.

[36] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[37] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau, "Scalable hyperparameter transfer learning," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[38] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," 2017.

[39] M. Feurer, J. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, Feb. 2015.

[40] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[41] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," 2015.

[42] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," 2018.

[43] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," 2020.

[44] C. He, M. Annavaram, and S. Avestimehr, "Towards non-i.i.d. and invisible data with fednas: Federated deep learning via neural architecture search," 2021.

[45] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," *CoRR*, vol. abs/1806.09055, 2018.

[46] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017.

[47] Y. Chen, J. Wang, C. Yu, W. Gao, and X. Qin, "Fedhealth: A federated transfer learning framework for wearable healthcare," 2021.

[48] S. H. Lee, S. Sharma, M. Zaheer, and T. Li, "Efficient adaptive federated optimization," 2024.

[49] Y. Himeur, I. Varlamis, H. Kheddar, A. Amira, S. Atalla, Y. Singh, F. Bensaali, and W. Mansoor, "Federated learning for computer vision," 2023.

[50] C. Quéméneur and S. Cherkaoui, "Fedpylot: Navigating federated learning for real-time object detection in internet of vehicles," 2024.

[51] J. Dong, D. Zhang, Y. Cong, W. Cong, H. Ding, and D. Dai, "Federated incremental semantic segmentation," 2023.

[52] M. Paulik, M. Seigel, H. Mason, D. Telaar, J. Kluivers, R. van Dalen, C. W. Lau, L. Carlson, F. Granqvist, C. Vandevelde, S. Agarwal, J. Freudiger, A. Byde, A. Bhowmick, G. Kapoor, S. Beaumont, Áine Cahill, D. Hughes, O. Javidbakht, F. Dong, R. Rishi, and S. Hung, "Federated evaluation and tuning for on-device personalization: System design & applications," 2021.

[53] Y. Liu, X. Bi, L. Li, S. Chen, W. Yang, and X. Sun, "Communication efficient federated learning for multilingual neural machine translation with adapter," 2023.

[54] M. Sheller, B. Edwards, G. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. Colen, and S. Bakas, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, 07 2020.

[55] D. Huang, X. Ye, Y. Zhang, and T. Sakurai, "Collaborative analysis for drug discovery by federated learning on non-iid data," *Methods*, vol. 219, pp. 1–7, 2023.

[56] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," 2020.

[57] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019.

[58] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, "Fedml: A research library and benchmark for federated machine learning," 2020.

[59] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011.

[60] Kohavi and Ron, "Census Income." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5GP7S.

[61] M. S., R. P., and C. P., "Bank Marketing." UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5K306.

[62] Hofmann and Hans, "Statlog (German Credit Data)." UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C5NC77.

[63] M. Monteiro, "Amazon reviews polarity dataset." Accessed via Hugging Face Datasets: https://huggingface.co/datasets/contemmcm/amazon_reviews_polarity_2013.

[64] W. William, M. Olvi, S. Nick, and S. W., "Breast Cancer Wisconsin (Diagnostic)." UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5DW2B.