Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (38^{th}cycle)

Distributed and Federated Learning over IoT networks

By

Erich Malan

******

Supervisor(s):

Prof. Andrea Calimera, Supervisor
Prof. Enrico Macii, Co-Supervisor

Doctoral Examination Committee:
Prof. Francesco Regazzoni, Referee, University of Amsterdam
Prof. Flavio Ponzina, Referee, San Diego State University
Dr. Antonio Guerrieri, ICAR–CNR (Istituto di Calcolo e Reti ad Alte Prestazioni)
Prof. Gianvito Urgese, Politecnico di Torino

Politecnico di Torino
2026

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Erich Malan
2026

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Let time flay it

scrape it raw

.

thee I cradle

me taught love.

Acknowledgements

I am deeply grateful to my supervisors, Prof. Andrea Calimera and Prof. Enrico Macii, for granting me the opportunity to conduct this research within the EDA research group. I would like to further thank Professor Calimera for his invaluable advice and insights, which greatly contributed to my work and personal growth.

Lastly, I would like to thank Valentino Peluso for his remarkable dedication and the countless hours he spent mentoring me. His guidance helped shape my research vision and clarify my direction, while his patience and support were crucial to this journey.

Abstract

The growing computational capabilities of low-power devices, paired with advances in wireless networking and deep learning, have accelerated the development of Artificial Intelligence of Things. This paradigm enables on-device data processing, enhancing privacy, efficiency, and supporting real-time inference. However, privacy regulations and infrastructure limitations restrict its full potential, since the prevailing paradigm for training deployed deep learning models still relies on centralizing data. Federated Learning (FL) offers a promising avenue enabling collaborative training across distributed nodes without sharing raw data.

However, the deployment of FL in Internet of Things (IoT) environments introduces several critical challenges. Frequent parameter synchronization leads to communication bandwidth bottlenecks, imposing a significant burden on the constrained resources of distributed nodes, which are often battery-powered and subject to limited data plans. In this scenario, achieving competitive accuracy becomes impractical, particularly when facing severe data heterogeneity across devices. Furthermore, the proliferation of sophisticated attacks poses significant threats to privacy and intellectual property, necessitating continuous investigation of undetected vulnerabilities and development of novel countermeasures, which generally come at the cost of reduced efficiency and accuracy. This thesis tackles these challenges through: i) gradient-driven adaptive strategies that optimize resource utilization without degrading model quality; ii) dynamic, budget-aware policies that enhance performance under strict resource budgets; and iii) security approaches that promote or evaluate the confidentiality of proprietary data and models.

It first investigates communication efficiency, providing an automatic layer-freezing strategy that progressively excludes stable model layers from synchronization, reducing exchanged data by up to 83.9% with minimal to no accuracy degradation. Additionally, a novel two-phase resource management policy overcomes

the accuracy-communication trade-off of static partial client participation, boosting accuracy by up to 12.2% under extreme data heterogeneity. For energy-constrained scenarios, a gradient-driven technique automatically adjusts participation, attaining up to 2.74 times greater energy efficiency. In the security domain, adapting automatic layer freezing to homomorphic encryption-based FL cuts client computation time by up to 35.5% and server computation time by 36.4%, while reducing communication overhead by 37.4%. The thesis also introduces a side-channel attack based on the CPU-frequency traces, discovering a novel threat to model confidentiality. Finally, it discusses the evaluation of FL strategies and applications. By revisiting the role of learning rate schedules and advocating for budget-aware configurations, it uncovers new solutions that reshape the advantages of federated algorithms. Finally, it presents a smart meter case study on sociodemographic classification in smart grids, showing that privacy-preserving FL with homomorphic encryption can closely match centralized accuracy.

Overall, this thesis delivers a set of strategies to address the intertwined challenges of efficiency, accuracy, privacy, and security in federated learning for IoT. By adopting a holistic perspective, it advances the trade-off between efficiency and accuracy, identifies emerging security threats and facilitates the adoption of relative countermeasures, and evaluates the potential of cross-device FL across different contexts and operational constraints. The proposed general yet practical solutions foster efficient, privacy-conscious, and scalable IoT shared intelligence, paving the way for the widespread adoption of ubiquitous intelligent services.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

AIoT Artificial Intelligence of Things

ALF Automatic Layer Freezing

API Application Programming Interface

CE Cross-Entropy Loss

CKKS Cheon-Kim-Kim-Song HE scheme

CNN Convolutional Neural Network

CS Configuration Space

DL Deep Learning

DVFS Dynamic Voltage and Frequency Scaling

ELS Extreme Label Skew

EWMA Exponentially Weighted Moving Average

GAP Gradient Aware Participation

HE Homomorphic Encryption

HPO Hyperparameter Optimization

IaaS Inference-as-a-service

IID Independent and Identically Distributed

IoT Internet-of-Things

IP Intellectual Property

ML Machine Learning

ppFL privacy-preserving Federated Learning

PTF Private Tensor Freezing

SCA Side Channel Attack

SGD Stochastic Gradient Descent

SGDM Stochastic Gradient Descent with Momentum

Symbols

η_{l} Local learning rate

η_{g} Global learning rate

δ^r Pseudo-gradient of round r

δ_s^r Update sent by client s at round r

$\hat{\delta}_{\mathbf{c}}^{\mathbf{r}}$ Encrypted Scaled Update

$\hat{\mathbf{w}}^r$ Encrypted Tensor

$\mathbf{w}^r$ Global model weights at round r

$\mathbf{w}_s^{(r)}$ Final updated model parameters from client s at round r

$\mathbf{w}_s^{(r,k)}$ Local model parameters of client s at local step k of round r

$\mathbf{x}_s$ Local input data of client s

$\mathbf{y}_s$ Target values associated to $\mathbf{x}_s$

$\mathscr{D}_s$ Local dataset of client s

$\mathscr{L}$ Loss function

$\mathscr{S}$ The set of clients (devices)

$\mathscr{S}^r$	Subset of clients selected at round r
$\nabla_s^{(r,k)}$	Gradient of the loss of client s during local step k of round r
B	Batch size
E	Number of local training epochs per client
K	Number of local training steps per client
pk	Public Key
R	Total number of communication rounds
r	Index of the communication round, where $r = 1, \ldots, R$
s	Client (device) belonging to the client pool $\mathscr{S}$
sk	Secure Key

Chapter 1

Introduction

1.1 Context & Motivation

The increasing computational capabilities of Internet-of-Things (IoT) low-power devices, coupled with the advancements in edge computing, Machine Learning (ML) and in particular Deep Learning (DL) , and wireless communication networks, paved the way for a new era of interaction between the physical and digital worlds, referred to as the Artificial Intelligence of Things. The ever-growing capability of sensing valuable and high-dimensional data from the physical world compounded by the hardware and software optimization to support DL models on resource-constrained devices, directly processing raw data at the network edge, improving performance under low-latency and limited bandwidth requirements, while enhancing privacy and security by keeping sensitive data closer to its source, enabled the deployment of edge intelligent services at scale. Yet, a critical obstacle to a full-scale Artificial Intelligence of Things (AIoT) paradigm is the collection and management of sensor data for training these deep learning models, due to privacy concerns and infrastructure scalability issues. One promising solution is the implementation of distributed and Federated Learning (FL) frameworks. These frameworks enable fleets of edge nodes to train a shared model without ever sharing their raw data, avoiding the need to collect and manage privacy-sensitive and high-dimensional data in a centralized fashion. The training process is distributed across the edge nodes, which progressively refine and synchronize the shared model parameters by using

their local datasets, thereby circumventing the need to transfer raw data to a central location.

1.1.1 The Training Bottleneck Between Deep Learning and IoT Data

Deep learning revolutionized the field of machine learning, impacting every sector of our everyday lives, from computer vision to natural language processing and speech recognition. The success of deep learning is due to the ability of deep neural networks to extract and learn complex patterns from large amounts of raw, unstructured, and heterogeneous data, giving actionable insights to improve interaction (and autonomously interact) with both the physical and cyber-world, as well as in social contexts. Since AlexNet [1], DL has evolved across two paths, the first one aims at developing evermore general and massively parametrized models (billions of parameters) in origin handling natural language understanding (e.g., GPT2 [2], LLama [3]) and computer vision [4] tasks, and as of today, expanding to multimodal tasks (generating and processing across modalities such as text, images, video, and audio), such as GPT-4 and Gemini. The second one aims at creating specialized and tailored models, particularly those designed for physical signals, real-world, and IoT data. Unlike foundation models, these architectures are typically much smaller, less parametrized, and tailored for the target application and hardware [5]. Such an approach for processing single or multi-sensory data eventually shows higher generalization capabilities, avoiding overfitting and sample memorization caused by extreme over-parametrization [6], while recent literature is investigating how to integrate both approaches for leveraging the advantages of both [7].

Indeed, the proliferation of IoT has meanwhile seen an explosion of low-power and ultra-low-power devices adoption, whose number today is in the billions world-wide [8, 9]. This massive network of devices comprises embedded sensors, wearable devices, smart appliances, and connected vehicles. These devices are producing an unprecedented volume of data at the network edge, accounting for 79.4 Zettabytes of data this year [10] and forecasted to grow continuously. Indeed, the growing sensing capabilities increased precision across spatial and time-frequency domains, thanks to advancements in both hardware and batteries. On the one hand, the evolution of deep learning architectures has enabled the processing of increasingly complex

and high-dimensional data; on the other hand, IoT devices are now capable of generating data of ever higher quality across spatial and time-frequency domains. Processing high-dimensional raw data is today possible thanks to the hardware [11], software [12–14], and co-design optimizations [15] for providing inference at the edge, by deploying such tiny models directly onto the IoT nodes and ultra-low-power smart sensors. Indeed, inference at the edge is a necessity to bypass the challenges of transmitting such data volumes to centralized cloud storage and computing servers because of latency, bandwidth, and most critically, privacy concerns.

The impressive applications of this paradigm are transforming everyday environments into smarter, more adaptive systems. Examples include industrial automation, where embedded sensors detect anomalies in machinery and predict failures before they occur, thereby enhancing reliability, cost-effectiveness, and safety, thanks to the capability of processing and inferring abnormal conditions from high-frequency vibrational and audio data [5, 16]. The same paradigm also extends to healthcare and environmental monitoring. Wearable devices, such as fitness trackers and smartwatches, can continuously analyze physiological signals to enable personalized health monitoring and facilitate the early detection of medical conditions [17]. Likewise, distributed networks of cameras and microphones can support biodiversity and resource monitoring, while the integration of satellite and drone data enables innovative approaches such as the automatic extraction of remote environmental-DNA samples from wild and poorly accessible ecosystems like the Amazon rainforest [18].

Despite these opportunities, the whole spectrum of real-world IoT data presents unique and persistent challenges. While large, well-curated datasets exist for modalities such as text, audio, and images, thereby enabling powerful generative models and standardized benchmarks, these only cover a subset of the data types and conditions encountered in practice. In many IoT scenarios, the resulting measurements are often noisy, context-specific, and less directly interpretable, which makes the underlying signal distributions harder to validate and correctly synthesize [19]. Centralizing such data is problematic, as it raises significant concerns related to privacy, transmission bandwidth, and data management costs. On the other hand, training models locally on individual devices is rarely effective, since the data available to a single node is often sparse, noisy, and insufficiently representative [20]. As a result, IoT machine learning lies between two extremes: centralization, which is costly and exposes sensitive information, and isolated local training, which presents a poor approach for performing and robust models. In a nutshell, although the technological

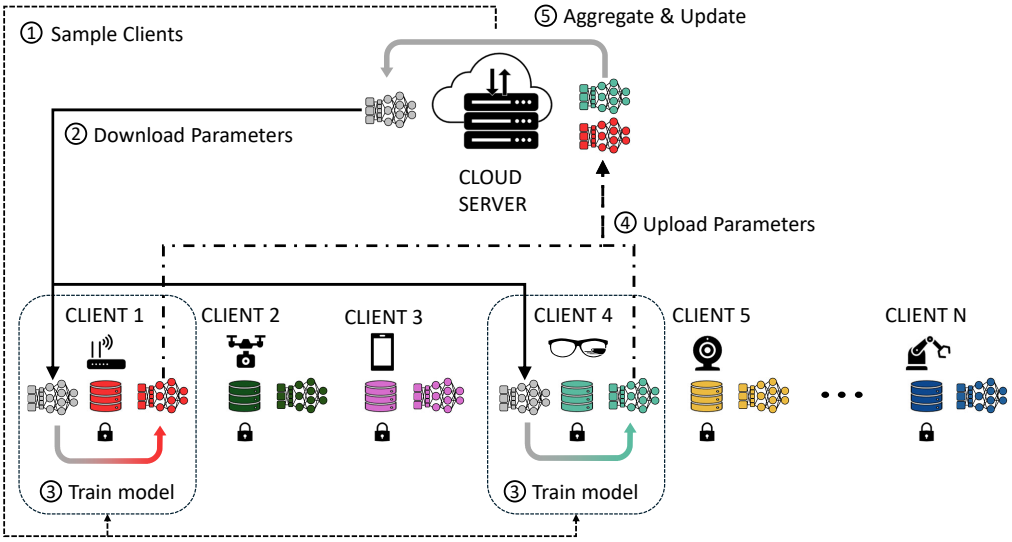


Fig. 1.1 Schematic view of a Federated Learning Workflow.

breakthrough enabled the deployment of deep learning models onto constrained hardware and low-power devices, the proliferation and adoption scale of the intelligent services is still largely limited by training these models, presenting the next major challenge toward a broad-scale AIoT paradigm.

1.1.2 Bridging the Gap: Federated Learning for Privacy-Aware and Efficient Training

The need for private yet general data-driven intelligent services calls for alternative data management strategies that enable collaborative learning without requiring data centralization. A potential avenue is Federated Learning, a distributed learning paradigm that enables decentralized embedded devices to cooperate in training deep learning models without sharing private raw data [21], thereby aligning with the latest data regulations [22].

Cross-device FL envisions a central server coordinating a fleet of low-power devices, referred to as clients, which collaboratively train a shared model through periodic synchronization of updates. Generally, the global learning encompasses multiple iterations, called rounds, usually until reaching convergence or a maximum number of iterations. Fig. 1.1 overviews the workflow of a generic round, which mainly consists of five stages, summarized as follows:

1. *Clients Sampling*. The central server checks the status of all the participating clients and selects a subset of the online devices to contribute to the training (Clients 1 in red and 4 in light blue in the example).

2. *Parameters Download*. The selected clients download the latest version of the global model parameters from the central server.

3. *Local Training*. Each client trains the model with its local dataset. The training iterates over multiple epochs or local steps, whose number is defined as a hyperparameter that regulates the synchronization frequency with the central server.

4. *Parameters Upload*. The selected clients upload the updated version of their local models to the central server. Together with the *Parameters Download* stage, these stages compose the communication cost of the FL framework, impacting the bandwidth and network load.

5. *Parameters Aggregation*. The central server aggregates the local models (or updates) collected from the selected clients and refines the global model parameters, producing a new version.

Although FL's core idea originates from multiple GPUs distributed learning methods, it applies to a different scenario, where the computing nodes are not connected through broadband and high-speed communication channels, and the data is not stored in a centralized location, mainly for privacy concerns and the massive data volumes generated by the sensors [23]. In fact, the participating devices are low-power devices, such as smartphones, tablets, or IoT devices, which are often battery-powered and may not always be online or available for training, and are often connected to wireless networks (e.g., wifi, LTE, 5G, LoRaWAN), which are characterized by limited bandwidth, lower energy efficiency, and poor reliability. By leveraging the local retention and control of data, the pivotal differences between distributed training and cross-device FL are the following:

- *Data Partitioning*: In distributed training, the dataset is centralized and typically partitioned across multiple nodes, preserving the homogeneous distribution of data. In cross-device FL, the data remains at the local sources and is not centralized, thereby strictly hampering the control over the data distribution.

The data is deemed to be unlikely Independent and Identically Distributed (IID) , as it is collected from different sources in different geographical places.

- *Multiple Local Steps*: In FL, each participating device performs several local optimization steps on its private dataset before communicating updates to the server. This reduces the need for constant communication, which is crucial in bandwidth-limited or high-latency environments such as mobile networks. The number of local steps acts as a knob to balance the communication cost and convergence.

- *Partial Participation*: Unlike GPU clusters, where all nodes participate in every step, FL operates under the assumption that only a limited subset of devices will be available or selected in each training round. Indeed, the devices involved might lack sufficient resources or the necessary conditions (e.g., idling and connected to stable and fast networks) for being deemed as eligible for the training and synchronization tasks.

FL has found a wide range of applications across various industrial scenarios, leveraging diverse data sources. In text-based settings, notable examples include next-word [24] and emoji [25] prediction. For audio, FL has been applied to keyword spotting [26]. Vision-based applications are extensive, spanning image labeling [27], gesture recognition [28], facial expression classification [29], lifelong person re-identification [30], and video anomaly detection [31]. Other application domains include virtual power plant scheduling [32] and intrusion detection systems [33].

1.1.3 FL Challenges

Despite these successes, the broader applicability of FL for IoT contexts is still challenged by several interconnected issues related to performance, efficiency, and security, stemming from the lack of centralized data management, the limited computational and energy resources of edge devices, and the interaction among mutually untrusted parties.

Statistical Heterogeneity: The standard training paradigm assumes centralized datasets that are often curated to follow an IID distribution. In contrast, data in federated settings is typically non-IID, meaning that clients exhibit different local data distributions, particularly at the label level, where each device may observe

only a subset of the target classes. This heterogeneity between nodes introduces local bias and convergence issues. Specific forms of data heterogeneity include label distribution skew and extreme label skew, both of which can limit the maximum accuracy achievable by a deep model trained in a federated fashion and, more importantly, slow the convergence of the training algorithm, inflating the costs required to achieve competitive accuracy levels. The complexity of this problem is inherently tied to the number of local steps and the concurrency level, which together make it more challenging to meet stringent constraints and accuracy demands.

Communication Bandwidth and Energy Efficiency: the costs endured by the clients and the infrastructure load represent a key challenge in FL. Frequent model updates across the network can overwhelm available bandwidth and quickly drain the batteries of the edge devices. Managing the synchronization frequency may reduce the bandwidth pressure, but it provides trade-offs between local computational effort and initial versus final convergence and accuracy levels. More importantly, the number of participants plays a key role in managing the network pressure, communication bandwidth, and training quality. Achieving better performance under fixed communication constraints, or alleviating the communication burden through selective and efficient synchronization, is key to realizing scalable federated learning systems. A further key cost is the energy efficiency. The participating devices in FL are often battery-powered and have limited computational resources. Thus, optimizing the energy consumption during the training process is essential to prolong the device's battery life and ensure its participation availability for future training rounds.

Privacy and Intellectual Property (IP) : although distributed intelligent services keep data local and avoid raw-data transmission, this only provides limited protection, since clients and servers still exchange raw model parameters or updates that can leak sensitive information and expose valuable model IP. The rapid evolution of increasingly sophisticated attacks calls for continuous research and security evaluation across all stages of the deployment pipeline, considering both training and inference deployment operations. Protecting training data, models, and learned parameters throughout all deployment phases (both during training and inference) is therefore crucial for sensitive applications and for safeguarding IP in distributed AI services. Encryption-based strategies offer a promising solution to secure the privacy advantages of federated learning, but introduce substantial system-wide overhead and complicate resource management on client devices. Additionally, the

trained models are typically deployed as edge inference services, where undetected attacks could jeopardize the effectiveness of the countermeasures introduced at the training stage. Promptly identifying and mitigating these threats is key to retaining the intended security and privacy guarantees of distributed services.

1.2 Objective & Contribution

The core macro objective of this work is to facilitate the adoption of intelligent services in real-world IoT scenarios by focusing on the main challenges that hinder the adoption of distributed and federated learning frameworks. The interconnection of these challenges underscores the need for innovative and holistic approaches tailored to budgeted and privacy-sensitive IoT ecosystems. To this end, this thesis pursues the following specific objectives:

- developing communication-efficient strategies that i) reduce the amount of data exchanged between the clients and the server, targeting label distribution skew; and ii) improving the attainable accuracy within constrained data plans under extreme label skew scenarios.
- designing energy-efficient mechanisms that optimize the edge resources during the training process, reducing the energy requirements of each participant when facing extreme label-skewed distributions.
- investigating threats undermining the privacy and intellectual property, both at the training and inference stages. A special focus is placed on encryption-based privacy-preserving techniques and facilitating their adoption by reducing the introduced overhead without compromising model utility.
- promoting fairer evaluation frameworks that ensure reliable comparisons between different FL algorithms and configurations, emphasizing the research of budget-bounded solutions
- investigating the application advantages of privacy-preserving FL across realistic use cases.

Although the aim is to propose solutions that are as general as possible, different challenges often require tailored approaches. Therefore, this thesis introduces a set

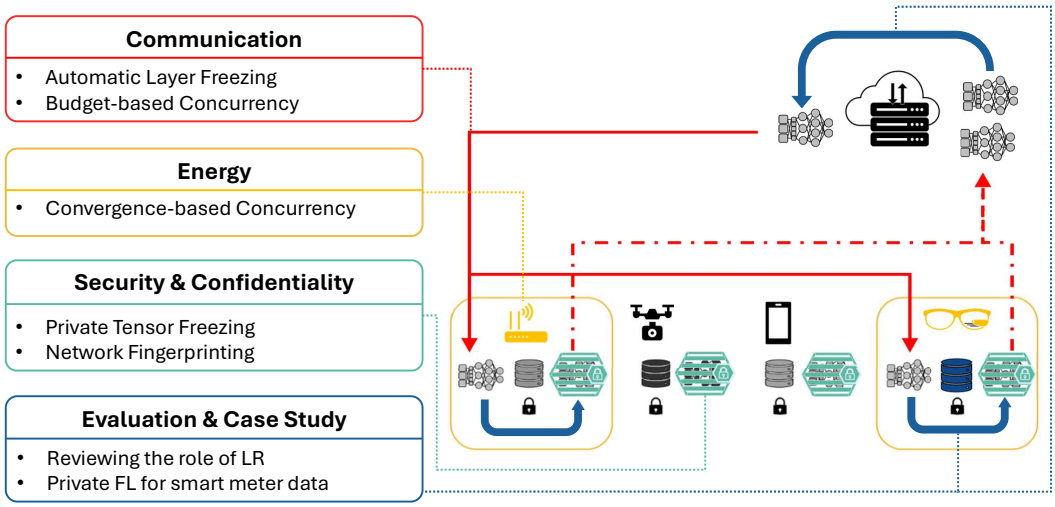


Fig. 1.2 Schematic view of the thesis contributions.

of optimizations carefully designed to minimize negative impacts on other critical metrics, while primarily targeting a single key metric or a specific challenge (e.g., extreme label-skewed data distributions). The overarching goal is to preserve model accuracy, avoid computational, memory, and communication overhead on the client side, and maintain strong privacy guarantees to prevent information leakage (without sharing raw data or sensitive data statistics).

The main contributions of this thesis, represented in the figure 1.2 can be summarized as follows:

- **Communication-efficient strategies for FL**. The work proposes two novel and adaptive strategies to reduce the communication cost in FL. The first one progressively freezes the layers of the neural network during training, reducing the amount of data exchanged between the clients and the server without introducing overhead on the client side or degrading model utility. The second approach dynamically adjusts client participation depending on the remaining budget throughout the training process, comprising two distinct phases with complementary objectives. The first phase aims to quickly approach a good global representation through fast and low-cost aggregations, while the second phase focuses on refining the model with higher participation rates. By adapting the per-round concurrency, this method strikes a balance between rapid global updates and robust class representation, ultimately improving

convergence in extreme label-skew scenarios and enhancing accuracy within limited communication budgets.

- **Adaptive resource management for Energy-efficient FL**. Further work investigates the impact of adaptive concurrency on reducing the energy costs required to achieve competitive accuracy under extreme label-skewed distributions. To mitigate the abundant energy costs, a gradient-based adaptive algorithm is developed to automatically adjust concurrency based on the training status. The core idea is to increase concurrency only when model progress shows signs of stagnation, thereby effectively managing the cost-quality trade-offs and providing a robust approach across varying levels of label skew.

- **Privacy and security in edge intelligence**. This thesis investigates protection countermeasures and attacks against the confidentiality of the AI distributed services. More specifically, it presents a tensor freezing as a means to reduce the computational and communication overhead of privacy-preserving FL with Homomorphic Encryption (HE) , without compromising model accuracy or privacy guarantees. In addition, it introduces a novel side-channel attack that leverages the CPU frequency traces to perform neural network architecture fingerprinting in edge intelligent services, threatening model intellectual property.

- **Evaluation: fair benchmarking and a smart grid case study**. This thesis emphasizes the importance of examining the role of the learning rate, particularly its evolution throughout training, to enhance the fidelity of comparisons across deployment settings. To this end, it defines a hyperparameter search space that includes two-sided learning rate schedules, ensuring fair benchmarking across strategies. It further deepens the analysis with a tighter focus on budget-aware learning solutions, which adapt learning dynamics to the available budget and thereby improve performance under constrained resources. Ultimately, the thesis concludes with a real-world analysis and investigates a practical use case that analyzes the trade-offs of privacy-preserving FL (ppFL) for fine-grained household sociodemographic classification using smart meter data. It introduces a novel benchmark that combines a realistic synthetic data generation pipeline with a ppFL evaluation framework, showing that FL based on encrypted updates can serve as a promising approach whose performance closely approaches that of a centralized baseline.

1.2.1 Thesis Structure

The remainder of this thesis is structured as follows.

Chapter 2 provides the necessary technical background on FL, introducing the main concepts and the algorithm backbones used throughout the thesis. It also discusses in more detail the challenges faced by FL in IoT scenarios and surveys the related work addressing them. For completeness, a brief recap of the most relevant concepts is provided at the beginning of each subsequent chapter.

Chapter 3 discusses communication-efficient strategies for FL. Section 3.1 explores the adoption of layer freezing to reduce communication costs without introducing overhead on the client side and without degrading model utility. Section 3.2 investigates a dynamic client participation mechanism that aims at increasing the model accuracy under fixed communication budgets when training data is characterized by extreme label skew.

Chapter 4 introduces an energy-efficient adaptive concurrency scheme based on the training dynamics that enables greater energy efficiency for cross-device FL under extreme label-skewed data distributions.

Chapter 5 addresses privacy and security challenges in FL. Section 5.1 introduces tensor freezing as a means to reduce the computational and communication overhead of FL based on homomorphic encryption, without compromising model accuracy or privacy guarantees. Section 5.2 presents a novel side-channel attack based on the CPU frequency, capable of performing neural network architecture fingerprinting in edge inference services, and thereby threatening the security guarantees of edge intelligent systems by compromising model confidentiality.

Chapter 6 introduces fair and pre-deployment evaluation frameworks. Section 6.1 emphasizes the importance of including two-sided learning rate schedules in the hyperparameter search space to ensure fair comparisons, further highlighting the benefits of budget-aware scheduling solutions. Section 6.2 provides a real-world analysis and evaluates the advantages of homomorphic-encryption–based FL for fine-grained household sociodemographic classification from smart meter data.

Finally, **Chapter 7** concludes the thesis by summarizing the main contributions and presenting key areas for future work.

1.2.2 Publication List

The research work presented in this thesis, besides the work done during my PhD, has led to several publications in high-quality journals and international conferences, as listed below in chronological order.

Journal Publications

1. E. Malan, V. Peluso, A. Calimera, and E. Macii, "Communication-efficient federated learning with gradual layer freezing," IEEE Embedded System Letters, vol. 15, no. 1, pp. 25-28, 2023.

2. E. Malan, V. Peluso, A. Calimera, E. Macii, and P. Montuschi, "Automatic layer freezing for communication efficiency in cross-device federated learning," IEEE Internet of Things Journal, vol. 11, no. 4, pp. 6072-6083, 2024.

3. E. Malan, V. Peluso, A. Calimera, and E. Macii, "Refined Two-Sided Learning Rate Tuning for Robust Evaluation in Federated Learning," in IEEE Transactions on Artificial Intelligence, vol. 7, no. 2, pp. 906-917, 2026.

4. E. Malan, V. Peluso, A. Calimera, and E. Macii, "FedAGF: Adaptive Concurrency via Gradient Feedback for Mitigating Extreme Label Skew in Budget-Constrained Federated Learning," in IEEE Transactions on Circuits and Systems I: Regular Papers, early access, 2026.

Conference Proceedings

1. E. Malan, V. Peluso, A. Calimera, and E. Macii, "Enabling DVFS Side-Channel Attacks for Neural Network Fingerprinting in Edge Inference Services," in Proc. IEEE ISLPED, Wien, Austria, 2024, pp. 1-6.

2. V. Peluso, E. Malan, A. Calimera, and E. Macii, "Private tensor freezing for an efficient federated learning with homomorphic encryption," in Proc. IEEE ICCD, Milan, Italy, 2025, pp. 308-315.

3. E. Malan, V. Peluso, A. Calimera, and E. Macii, "Draft & refine: Efficient resource management in federated learning under pathological labels skew," in Proc. IEEE ICECS, Nancy, France, 2025, pp. 1-4.

4. E. Malan, V. Peluso, A. Calimera, and E. Macii, “Adaptive Client Participation in Budget-Constrained Federated Learning with Extreme Label Skew," in Proc. IEEE NEWCAS, Paris, France, 2025, pp. 15-19.

5. E. Malan, C. De Vizia, M. Castangia, V. Peluso, A. Calimera, and E. Macii, “Privacy-Preserving Federated Learning for Household Characteristic Identification,” in Proc. IEEE COMPSAC, Toronto, ON, Canada, 2025, pp. 2040-2045

6. E. Malan, V. Peluso, A. Calimera, and E. Macii, “Gradient-Aware Participation for Energy Reduction in Federated Learning with Extreme Label Skew,” in Proc. IJCNN, Rome, Italy, 2025, pp. 1-7.

Chapter 2

Background and Literature Review

This section provides a more technical overview of cross-device federated learning, introducing its core algorithms, key challenges, and the current state-of-the-art approaches. It sets the foundation for understanding the motivations behind the proposed solutions in this work.

2.1 Cross-Device Federated Learning

Cross-device FL is a decentralized machine learning paradigm where data is horizontally partitioned across a fleet of low-power devices, denoted as $\mathscr{S}$. The conventional setup considers a known pool of clients, with cardinality $|\mathscr{S}|$. Each device s possesses a unique, private dataset $\mathscr{D}_s$, consisting of input data $\mathbf{x}_s$ and corresponding targets $\mathbf{y}_s$.

This thesis focuses on the standard synchronous FL architecture with a centralized topology, where a central server orchestrates training over multiple communication rounds, denoted by R, indexed by $r \in \{1,\ldots,R\}$. Devices collaboratively train a single global model by updating shared weights $\mathbf{w}$. Throughout each round, selected devices independently refine $\mathbf{w}$ using their private data $\mathscr{D}_s$ without revealing raw samples or sensitive information, thus enhancing user privacy. Devices periodically transmit local updates to the server, which aggregates them to update and improve the global model. This cycle of distribution, local computation, and aggregation possibly continues until convergence, yielding an accurate global model.

2.1.1 General Objective

The participating clients $s \in \mathscr{S}$ collaboratively seek to minimize the federated objective in Equation 2.1. The global model $f(\mathbf{w})$ maps input $\mathbf{x}$ to target $\mathbf{y}$ and is trained to minimize the empirical loss computed over the union of all clients' datasets. Specifically, during each round r, the objective is to minimize the average empirical error across all clients:

$$\min_{\mathbf{w}} \frac{1}{|\mathscr{S}|} \sum_{s \in \mathscr{S}} F_s(\mathbf{w}), \tag{2.1}$$

where $F_s^{(r)}(\mathbf{w})$ denotes the local empirical loss for client s computed using model parameters $\mathbf{w}$ at round r.

2.1.2 Federated Averaging

The most representative work in this context is *Federated Averaging (FedAvg)*, whose algorithm is outlined in Algorithm 1. The process begins on the server, which randomly initializes the global model weights $\mathbf{w}^1$ (line 1). Training proceeds over R communication rounds. As introduced in Chapter 1, each round r consists of the following key steps: *Client Sampling*, *Parameter Download*, *Local Training*, *Parameter Upload*, and *Parameter Aggregation* (lines 3–12).

In each round, a random subset of clients $\mathscr{S}^r$ is selected from the full client set without replacement (line 2), emulating the random availability of devices. The number of participating clients C represents the concurrency level [34]. Concurrency is usually limited by infrastructure constraints, such as maximum bandwidth or server capacity [21], but can be fine-tuned to balance network load, overall training cost, convergence speed, and final model performance.

Then, in parallel, each selected client downloads the latest global model parameters $\mathbf{w}^r$ and overwrites its local model parameters with these global values (line 3). This is followed by the local training loop (lines 4–7), which consists of K local training steps. While the standard implementation uses a fixed number of local steps, previous literature often employed the number of local epochs E , iterating over each batch for the entire dataset.

Algorithm 1: FedAvg workflow.

Input: R Number of rounds, C Concurrency , B Batch size , η_l Local learning rate , K Number of local training steps
Output: Global model parameters $\mathbf{w}^R$

1 $\mathbf{w}^1 \leftarrow$ Random weight initialization
2 **for** $r = 1, \ldots, R$ **do**
  /* Client Sampling */
3 Sample a random subset $\mathscr{S}^r \subset \mathscr{S}$ of C clients
  /* Local Training on selected clients */
4 **for** $s \in \mathscr{S}^r$ **do**
    /* Parameter Download */
5    $\mathbf{w}_s^{(r,0)} \leftarrow \mathbf{w}^{(r)}$
    /* Local Updates */
6   **for** $k = 1, \ldots, K$ **do**
7      $(\mathbf{x}_s^{(r,k)}, \mathbf{y}_s^{(r,k)}) \leftarrow \text{Sample}(B, \mathscr{D}_s)$
8      $\mathbf{g}_s^{(r,k)} \leftarrow \nabla \mathscr{L}(\mathbf{w}_s^{(r,k-1)}; \mathbf{x}_s^{(r,k)}, \mathbf{y}_s^{(r,k)})$
9      $\mathbf{w}_s^{(r,k)} \leftarrow \mathbf{w}_s^{(r,k-1)} - \eta_l \cdot \mathbf{g}_s^{(r,k)}$
    /* Parameter Upload */
10    $\mathbf{w}_s^{(r)} \leftarrow \mathbf{w}_s^{(r,K)}$
11   SENDTOSERVER($\mathbf{w}_s^{(r)}$)
  /* Parameter Aggregation */
12 $\mathbf{w}^{(r+1)} \leftarrow \frac{1}{|\mathscr{S}^r|} \sum_{s \in \mathscr{S}^r} \mathbf{w}_s^{(r)}$

During each local step, the client performs mini-batch Stochastic Gradient Descent (SGD) . First, it samples a batch of data and corresponding targets from its private dataset (line 5). The client then evaluates the loss function $\mathscr{L}$ by forward-propagating the batch through the model and comparing the output with the targets. The gradient of the loss, $\nabla_s^{(r,k)}$, is computed via back-propagation with respect to the local model parameters. The parameters $\mathbf{w}_s^{(r,k)}$ are then updated by scaling the computed gradient with the local learning rate η_l.

After completing K local updates, the client stores the latest model version (line 9) as $\mathbf{w}_s^{(r)}$ and uploads it to the server. The server then aggregates the models received from all selected clients by averaging their locally trained parameters, producing the new global model $\mathbf{w}^{(r+1)}$. To notice that the aggregation at the server is performed over only a fraction of all possible updates, exacerbating the bias originating from

non-IID data. The partial participation effects compound with the biases caused by data partitioning and the drift induced by the multiple local steps.

The analysis proposed in [35, 36] shows that the averaging of client models in federated learning can be reformulated as an approximation of an SGD step. In this interpretation, the average global model parameters $\mathbf{w}^{(r)}$at round r serve as the initial point, the global learning rate η_{g} is set to 1, and the aggregate of client updates δ_s^r acts as a sort of gradient. Specifically, each client modifies the received global model by performing multiple local updates. The cumulative update by the client at the local step k is given by the difference between its locally updated model $\mathbf{w}_s^{(r,k)}$ and the global model which downloaded $\mathbf{w}^{(r)}$, more formally:

$$\delta_s^{(r)} = \mathbf{w}_s^{(r,k)} - \mathbf{w}^{(r)}. \tag{2.2}$$

The server updates the global model with the average of the sampled clients' updates, which is referred to as *pseudo-gradient*.

$$\delta^{(r)} = \frac{1}{\left|\mathscr{S}^{(r)}\right|} \sum_{s \in \mathscr{S}^{(r)}} \delta_s^{(r)}. \tag{2.3}$$

It is straightforward to see that the parameters average or the SGD reformulation with global learning rate 1 provide the same resulting set of weights $\mathbf{w}^{r+1}$, as expressed in Eq. 2.6:

$$\begin{aligned}
\mathbf{w}^{(r+1)} &= \frac{1}{\left|\mathscr{S}^{r}\right|} \sum_{s \in \mathscr{S}^{r}} \mathbf{w}_s^{(r,k)} && (2.4)\\
&= \frac{1}{\left|\mathscr{S}^{r}\right|} \sum_{s \in \mathscr{S}^{r}} \left(\mathbf{w}^{(r)} + \delta_s^{(r,k)}\right) && (2.5)\\
&= \mathbf{w}^{(r)} + \delta^{(r)}. && (2.6)
\end{aligned}$$

As detailed in Algorithm 2, the generalized FL workflow extends the standard training loop by introducing *two-sided optimization*. In addition to the local learning rate η_{l} used during client training, a global learning rate η_{g} is defined for the server-side update. This extension allows both sides of the optimization process to be tuned, offering additional optimization compared to the classical FEDAVG formulation [37]. The main modifications are highlighted in light gray in the algorithm.

Algorithm 2: Generalized Federated Learning Workflow with two-sided optimization.

Input: R Number of rounds, C Concurrency, B Batch size, η_l Local learning rate, η_g Global learning rate, K Number of local training steps
Output: Global model parameters $\mathbf{w}^R$

$\mathbf{w}^1 \leftarrow$ Random weight initialization
for $r = 1, \ldots, R$ **do**
  /* Client Sampling */
  Sample a random subset $\mathscr{S}^r \subset \mathscr{S}$ of clients of size C
  /* Local Training on selected clients */
  for $s \in \mathscr{S}^r$ **do**
    /* Download Global Model */
    $\mathbf{w}_s^{(r,0)} \leftarrow \mathbf{w}^{(r)}$
    /* Perform K Local Steps */
    for $k = 1, \ldots, K$ **do**
      $\mathbf{w}_s^{(r,k)} \leftarrow \textsc{Optimize}(\mathbf{w}_s^{(r,k-1)}, \mathscr{D}_s, B, \eta_\mathrm{l})$
    /* Upload Client Update */
    $\delta_s^{(r)} \leftarrow \mathbf{w}_s^{(r,K)} - \mathbf{w}^{(r)}$
    $\textsc{SendToServer}(\delta_s^{(r)})$
  /* Global Model Update */
  $\delta^{(r)} = \frac{1}{|\mathscr{S}^{(r)}|} \sum_{s \in \mathscr{S}^{(r)}} \delta_s^{(r)}$
  $\mathbf{w}^{(r+1)} \leftarrow \textsc{Optimize}\left(\mathbf{w}^{(r)}, \delta^{(r)}, \eta_\mathrm{g}\right)$

On the client side, optimizers leveraging momentum, such as Stochastic Gradient Descent with Momentum (SGDM) or adaptive optimizers, such as Adam, may replace vanilla SGD, while the server aggregation step can likewise adopt different optimization rules. Formally, the global model update at round r can be expressed as:

$$\mathbf{w}^{(r+1)} = \textsc{Optimize}(\mathbf{w}^{(r)}, \eta_\mathrm{g}, \delta^{(r)}), \tag{2.7}$$

where OPTIMIZE denotes any optimizer that computes the next global weights from the aggregated pseudo-gradients $\delta^{(r)}$. This general formulation enables the deployment of several widely used optimization methods within the same framework. In particular, FEDAVG server-side optimization step can be formulated as:

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} + \eta_\mathrm{g} \delta^{(r)}, \tag{2.8}$$

where $\eta_g^{(r)} = 1$.

2.2 Cross-Device FL: Challenges and Solutions

As introduced earlier, FL can be regarded as an approximation of centralized training distributed across GPU clusters to acquire a unified global model without aggregating raw data in a single location. Although this decentralized approach offers clear privacy advantages, it generally incurs an accuracy gap compared to centralized training due to interdependent factors such as data heterogeneity, partial client participation, limited bandwidth, and resource constraints of client devices (e.g., energy, memory, and communication budgets).

Despite considerable progress, effective FL implementations still face significant challenges. A first challenge concerns training quality, which is affected by structural and environmental factors: clients typically hold statistically heterogeneous datasets, and only a subset of clients per round is deemed eligible, further amplifying heterogeneity and degrading the quality of the global model. A second challenge lies in the training budget, encompassing resource usage on clients, the server, and the overall network. This budget is often defined by the number of training rounds required to reach a target accuracy [38, 39]; as the number of rounds grows, the demands on computation, memory, and communication result in substantial recurring costs.

The relative importance of these costs depends, however, on the application context. In some scenarios, the pressure is primarily on instantaneous infrastructure load (e.g., network bandwidth saturation or server-side resource limits), while in others, the dominant concern is overall energy inefficiency, which not only drives up operational costs but also contributes to a higher carbon emission footprint [34]. Furthermore, since many cross-device clients are battery-powered, intensive FL workloads accelerate battery depletion, shortening device lifespan and indirectly increasing environmental impact through more frequent, costly hardware replacement.

Data heterogeneity

One of the central aspects of FL lies in how data is partitioned across participating clients, as introduced in Chapter 1. The general assumption is that data is horizontally

partitioned; in other words, clients share the same feature space but own unique samples. This reflects realistic scenarios such as smartphones or IoT devices, where each client collects its own user- or device-specific data. Unlike centralized machine learning, in FL, the data splits are often highly heterogeneous (the data is partitioned in a non-IID fashion), with each client potentially holding unique feature and label distributions. A major concern is the label distribution skew, which negatively impacts the model accuracy and training convergence.

LABEL DISTRIBUTION SKEW refers to scenarios when clients have datasets with different proportions of classes, biasing the updates of each client and creating different competing objectives. As a result, FL algorithms converge more slowly and might achieve lower global model accuracy.

EXTREME LABEL SKEW is a more severe form of label distribution skew, where each client's dataset contains only a small subset of the target classes. In some cases, clients may have completely disjoint label subsets, making local training and aggregation particularly challenging. ELS worsens the problems caused by label skew: local updates become more biased, and with partial client participation, even the combined datasets may not adequately represent the overall target label distribution. For example, in intrusion detection systems based on network logs, each device typically records only a limited set of traffic events, and no single device observes all possible types of attacks [33].

Communication-Efficiency under non-IID settings

Enhancing communication efficiency in cross-device FL can be formulated as an optimization problem that can be tackled at various stages of the FL workflow by tuning different system parameters as well as different training strategies, for which a comprehensive view is given in [40].

These strategies can be classified into four main categories: (i) those that alleviate network pressure by modulating the synchronization frequency, (ii) those that reduce communication overhead by compressing the exchanged payloads, (iii) those that accelerate convergence through local training, or (iv) aggregation optimization. Although primarily designed to enhance accuracy rather than communication efficiency, training and aggregation optimization methods also accelerate convergence,

thereby indirectly reducing the number of rounds; thus, consequently reducing the transmission costs required to achieve a target accuracy level

Synchronization Frequency As introduced previously, *FedAvg* [37] addresses this bottleneck through a simple yet effective strategy that leverages the number of local training epochs E (or equivalently K local steps in modern implementations) as a tunable parameter. In this approach, clients upload their local models after training for E epochs, where E is fixed in advance. Lower values of E generally improve final accuracy, as information is exchanged more frequently with the server and other clients, limiting the client drift at the cost of additional bandwidth usage. The trade-off analysis in [37] demonstrates that moderately increasing E (e.g., $E = 5$ as a practical rule of thumb) can substantially reduce communication overhead while incurring little to no loss in accuracy. More advanced approaches have introduced finer-grained control over the synchronization rate by adaptively tuning E at different levels of granularity: (*i*) *spatial granularity*, such as per-layer [41–43] or even per-weight adjustments [44]; (*ii*) *temporal granularity*, where E varies across training rounds [45, 46]; and (*iii*) *device granularity*, where E is adapted according to each client's training progress [47].

Parameters Compression An alternative class of optimization strategies reduces communication overhead through model compression, thereby decreasing the size of exchanged updates. Common techniques include sparsification [48–51], quantization [52, 53], or hybrid approaches combining both [54]. Compression may be applied on the client side to lower upload costs or on the server side to reduce download costs [55]. The rationale is based on the assumption that neural networks are largely overparametrized. Thus, the associated payloads can be approximated by ignoring less important parameters or reducing their bit-width precision. However, in either case, the clients' resource consumption increases significantly due to the additional computation required for compression and decompression on both ends, plus it requires additional computational efforts for similar accuracy levels as the approximations hurt the convergence rate, ultimately diminishing the final accuracy [56].

Local Training Optimization The most adopted training loss function for training deep learning classification models is the Cross-Entropy (CE) Loss. For cross-device FL settings, it is maintained to update the model locally, but, as is widely known in centralized settings, the CE loss is sensitive to imbalanced and skewed data

distributions. This problem is exacerbated in cross-device FL as the datasets are highly skewed and imbalanced across the clients. This naturally led researchers to develop alternative loss functions that try to mitigate the local update divergence from the global model. The most known approach is *FedProx* [57], which introduced a proximal regularization term to penalize local updates diverging from the received global model. Following work [58–60] has shown that this strategy offers benefits only under specific conditions, motivating the design of more sophisticated loss functions and training procedures. However, no consensus has emerged on a universally effective solution, as effectiveness often depends on the task and deployment context. Furthermore, more complex formulations introduce additional memory and computational overhead, which can be problematic for edge devices with constrained resources.

Aggregation Optimization As previously introduced, the most adopted FL framework *FedAvg* was developed based on the objective outlined in 2.1. Indeed, *FedAvg* updates the global model by averaging the weights collected from the latest participating clients, which follows the weighted average of the objective. However, extensive work has debated that the simple weighted average of the parameters is suboptimal, especially given the heterogeneous data and the partial participation. Consequently, a line of work investigated alternative updating strategies, extending the original idea by updating the global model with more refined algorithms. Optimized aggregation procedures can accelerate convergence, thereby reducing the number of communication rounds required to achieve a target accuracy. As with training-side optimizations, faster convergence directly translates into lower communication overhead.

For instance, FEDNOVA provided a more general formulation through a pre-aggregation scheme that calibrates the local updates by normalizing them depending on the client's status. A line of work proposed instead building upon the two-sided optimization, suggesting the use of optimizers on the server side, achieving state-of-the-art performances. More specifically, by leveraging the concept of pseudo-gradients, FEDAVGM [61] proposed using momentum on the server side to reduce the oscillations caused by heterogeneous data and partial participation. The authors of [36] proposed instead the use of adaptive optimizers to further enhance the model convergence. Although multiple optimizers could be used on the server side (e.g., Adagrad [62], or YOGI [63]), the most adopted algorithm leverages the Adam optimizer [64], establishing FEDOPT (FEDADAM) as a standard communication-efficient

algorithm. Experimental results on label distribution skew benchmarks demonstrate that FEDOPT, outperforms FEDAVG in both accuracy and communication efficiency, underscoring the pivotal role of the aggregation strategy in federated learning.

Enhancing Communication and Energy Efficiency under Extreme Label Skew

Communication and computation optimizations Cross-device FL implementations involve massive devices participating in thousands of rounds to achieve competitive accuracy. This leads to dramatic energy consumption on the client side, featuring limited computational capabilities and high sensitivity to energy consumption, given that the devices are often battery-powered and have limited resources [65].

General FL implementations investigated compression strategies for communication efficiency, initially appointed as the core bottleneck of FL. These strategies aim at reducing payloads through sparsification [49–51] and quantization [53] generally effective for over-parametrized networks. Alternative strategies investigated selective synchronization to limit the updates to specific weights [66] or layers [67, 68]. If, on the one hand, reducing the communication payload might reduce the energy consumption on the client side, it might introduce additional computational cost [66]. More recently, the energy consumption has attained momentum, and these strategies have been investigated from a communication and computational efficiency standpoint. Ad-hoc sparsification strategies [69, 70], fixed-point training and synchronization solutions [56] have been extended to take into account the computational burden, which might drastically influence the client-side energy consumption. However, these strategies reduce the attainable accuracy [69, 56], exacerbating the limits of FL when deployed in scenarios affected by Extreme Label Skew (ELS).

2.2.1 Addressing Extreme Label Skew

The importance of challenging conditions introduced by the extreme label skew garnered attention from the research community. Different tailored optimization strategies have been continuously developed to mitigate the negative effects at different procedural stages, focusing mainly on the local training stage. For clarity, we organized the existing methods into four categories, each representative of a main

operating stage: *local training*, *data augmentation*, and *model aggregation and client orchestration*.

Local Training Local updating strategies and loss functions offer a natural way to mitigate local bias caused by missing classes without altering the underlying system architecture or the participation assumptions. In practice, local training optimization has become the primary means of handling extreme label skew by recognizing the widely adopted CE loss as a root cause for biased and inconsistent updates in this regime [71–75]. The gradient dynamics of local training on a limited subset of classes induce asymmetric pushing and pulling forces [71], overemphasizing the present classes' data and overwriting previous global knowledge of missing classes' data. As a consequence, these updates tend to cancel each other during aggregation. Tackling this problem, the pioneering work of FedRS [71] proposed to scale the output activations of the final softmax layer, reducing the effects of the missing classes. FedLC [72] further extended this idea by calibrating the model updates according to the local data distribution. Despite mitigating the misalignment of the local updates, these methods modify the original objective, resulting in faster convergence but ultimately reducing the final model utility. A complementary thread of work highlights that biased local training leads to catastrophic forgetting, which refers to local updates forgetting the global knowledge about absent classes [76–78]. Thus, these works investigated the comparison between local and global features, trying to improve the preservation of the global knowledge on missing classes. This is referred to as *knowledge distillation* and compares the output activation of the global model with the local one, penalizing divergence [76–78]. Yet, performing forward passes through both the global and local models at each training iteration introduces significant computational overhead. A more refined version was investigated in FedVLS [79], which combined the two approaches as complementary strategies. The reweighted loss function is smoothed depending on the local present classes to avoid misleading updates over the missing classes. The activation outputs are further compared with the global model to enhance their classification capability. This theoretically further preserves the previous knowledge on the missing classes. Ultimately, the authors of [80] identified that different samples contribute differently to the client drift. Therefore, they suggest progressively increasing the data pool of a participating client depending on the measured error. This requires scoring the data

samples each time a client participates, introducing extra forward passes and ranking operations.

Alternative strategies Although local training optimization leads the current state-of-the-art strategy for tackling the extreme label skew, other approaches offer orthogonal alternative solutions by synchronizing/generating additional information (e.g., data) or orchestrating the clients depending on their data or update similarity.

Data and Feature Augmentation The problem of missing class samples on the local devices datasets can be targeted by generating synthetic samples [81] or augmenting the local datasets with publicly accessible databases [82]. The underlying assumption of these methods is either having the capability of generating and storing the additional data, which at first introduces computational overhead due to generation or retrieval, while also requiring representative labelled IoT data, which is generally unavailable. For this reason, recent works additionally share per-class embeddings or feature statistics to compensate for the missing classes [83] and possibly combine this approach with tailored loss functions [84], though the additional overhead and privacy implications remain unresolved.

Model aggregation A few studies have focused on the orchestration and aggregation stage, organizing clients in clusters to concurrently optimize one or multiple models. One example is FedConcat [85], which groups clients based on their update similarity and trains separate global models. These models are later merged in a final training phase to obtain a shared global model. However, this method has two major limitations. First, full client participation is required at each round of the clustering process, which is unrealistic in cross-device FL. Second, the size of the final model increases proportionally with the number of clusters, which could hinder deployment on devices with limited resources.

Clients orchestration An orthogonal approach investigated finer-grained selection methods considering a scheduling of the participants, i.e., focusing on which clients should participate in training instead of the concurrency level. Recently, FedSE [86] proposed an alternative clustered sequential training strategy. Clients are split such that the resulting clusters contain all target classes. At each round, clients within a cluster train a shared model iteratively and then upload it to the server for global synchronization. This approach leads each cluster model to learn from more diverse classes, limiting drifts and improving aggregation. Instead of using fixed clusters,

SAFA [87] implemented a sequential training strategy with dynamic interactions among clients. However, as the authors of [86, 87] also stated, sequential training incurs significant communication and energy overhead due to frequent inter-client synchronization, making it unsuited for cross-device FL.

2.2.2 Securing privacy and intellectual property in FL

Recent work demonstrated that the privacy-preserving promise of FL could be compromised by advanced attacks. Despite raw data never leaving the client devices, it is insufficient for guaranteeing privacy due to the periodic exchange of model updates between the clients and the central server. Although these updates do not directly reveal raw data, a growing body of research has shown that plaintext model updates can still leak sensitive information, eventually allowing an honest-but-curious server to reconstruct the private data [88, 89]. Since gradients and weights encapsulate learned representations of the data, they can provide valuable insights into the model's internal structure and behavior. Granting the server and third-party services unrestricted access to such information risks disclosing proprietary knowledge from clients. Indeed, beyond privacy, sharing updates in plaintext also exposes the IP of the local models themselves. Adversaries may exploit them to reconstruct original samples (gradient inversion attacks), infer membership in the training dataset, or extract attributes of the underlying data.

To mitigate these risks, privacy-preserving techniques have been introduced, with Homomorphic Encryption emerging as one of the most promising approaches, embedded into industrial-grade frameworks, e.g., NVIDIA Flare [90], FATE [91], PySyft [92], IBM Federated Learning [93], and FEDML [94]. HE allows mathematical operations to be performed directly on encrypted values, meaning that clients can transmit their updates in encrypted form and the server can perform the aggregation without ever decrypting. This ensures that the server or other honest-but-curious eavesdropping adversaries gain no access to the actual contents of the updates, thereby protecting both user data privacy and model intellectual property.

The integration of HE into FL (HE-based FL) provides strong theoretical guarantees but also introduces non-negligible challenges. Encrypting high-dimensional model parameters significantly increases the size of transmitted messages, thereby amplifying communication costs. Moreover, homomorphic computations are sub-

stantially more resource-intensive than their plaintext counterparts, raising concerns about computational overhead and energy consumption on resource-constrained devices. These challenges have fostered research into more efficient encryption schemes and hybrid approaches that combine HE with other techniques such as secure aggregation or differential privacy, aiming to balance privacy and IP protection with system efficiency.

HE-based Federated Learning

HE is a form of cryptography that allows computations to be performed directly on encrypted data. Over the past decades, several HE schemes have been proposed, with Paillier [95] and Cheon-Kim-Kim-Song (CKKS) [96] becoming the most widely adopted in federated learning. Independent of the strategy, HE-based FL implementations require a Trusted Key Authority for generating two keys, the secure (private) key distributed to the clients for decryption and a public key to be distributed to both clients and server for encryption and aggregation (from now on, *evaluation*) respectively. In the encryption stage, raw data is transformed into ciphertext using the public key. Decryption, which requires access to the secret key, can then recover the original data, but only by authorized parties. The evaluation stage, typically performed by untrusted entities (e.g., a server rented from a third-party service), enables computations to be carried out directly on the ciphertext without exposing the underlying data. The key authority is responsible for initializing, storing, and distributing the cryptographic keys to all participants in the protocol.

The deployment of HE schemes in FL frameworks is constrained by the specific data formats allowed and the limited set of arithmetic operations they support. For example, the Paillier scheme operates only on integers and supports addition exclusively. Consequently, when clients update their tensors, model weights must be converted from 32-bit floating-point to a fixed-point representation, a process introducing non-negligible accuracy degradation in FL environments. In contrast, the CKKS scheme supports both addition and multiplication on floating-point numbers, albeit with a bounded approximation error, which has been shown to have minimal impact on FL model utility, a finding further corroborated by the provided experimental results (Chapter 5, table 5.1).

The pseudo-code in Algorithm 3 illustrates the workflow of a HE-based FL implementation. The key differences with respect to the baseline strategy in Algorithm

Algorithm 3: Workflow of HE-based Federated Learning.

```
1  /* Trusted Key Authority                                              */
2  Initialize sk, pk, and cryptographic context
3  Distribute sk and pk to the clients
4  Distribute pk to the server
   /* Server initialization                                              */
5  for t ∈ 𝒯^1 do
6  |   w_t^1 ← Random initialization
   /* Training Rounds                                                    */
7  for r = 1,...,R do
8  |   Sample a subset 𝒮^r of clients
   |   /* Clients Optimization                                          */
9  |   for s ∈ 𝒮^r do in parallel
10 |   |   for t ∈ 𝒯^1 do
11 |   |   |   ŵ_{t,s}^{r,1} ← ŵ_t^r
12 |   |   |   w_{t,s}^{r,1} ← DECRYPT(ŵ_{t,s}^{r,1}, sk)
13 |   |   w_{t,s}^{r,S} ← TRAIN(w_{t,s}^{r,1}, 𝒟_c, S), ∀t ∈ 𝒯^1
14 |   |   for t ∈ 𝒯^1 do
15 |   |   |   δ_c^r ← 1/|𝒮^r| (w_{t,s}^{r,S} − w_t^r)
16 |   |   |   δ̂^r ← ENCRYPT(δ_c^r, pk)
17 |   |   |   Upload δ̂_c^r to the server
   |   /* Server Optimization                                           */
18 |   for t ∈ 𝒯^1 do
19 |   |   δ̂^r ← EVAL(∑_{s∈S^r} δ̂_c^r, pk)
20 |   |   ŵ^{r+1} ← EVAL(ŵ^r + δ̂^r, pk)
```

1 are highlighted in light blue. The process begins with the trusted key authority, which generates the secret and public keys together with the cryptographic context, i.e., the configuration parameters required to perform operations in the encrypted domain. The secret key is used exclusively for decryption, whereas the public key enables encryption and encrypted-domain arithmetic. For clarity, the pseudo-code explicitly shows only the key steps generation abstracting away the hyperparameters (line 2). The authority then distributes both the secret key (*sk*) and public key (*pk*) to the clients, while the server receives only the public key *pk* (lines 3 and 4). Once initialized, clients retrieve the global model snapshot from the server by downloading the encrypted tensors $\hat{\mathbf{w}}^r$ (line 11). These are decrypted locally using the secret key (line 12). Then, the clients can proceed with local training on plaintext weights (line 13). Once the training task is completed, they start synchronizing back the tensors (lines 14–17), first by computing the effective update already scaled $\boldsymbol{\delta}_c^r$ to relieve the

server of this expensive operation (line 15); then, they encrypt it $\hat{\delta}_c^r$ (line 16), and transmit the ciphertext to the server (line 17).

Finally, once all encrypted updates are collected, the server performs secure aggregation (line 19), producing the new encrypted model weights $\hat{\mathbf{w}}^{\mathbf{r+1}}$ (line 20), representing the new global model snapshot that will be distributed in the next communication round.

Optimization of HE-based FL

A major source of inefficiency in HE lies in the large size of ciphertexts, which is primarily determined by the encryption key length rather than the size of the underlying plaintext. To mitigate this, the technique of batching is employed, where multiple plaintext values are packed into a single ciphertext before encryption. Batching reduces communication overhead by lowering the number of ciphertexts that need to be transmitted. It also enables faster computation by exploiting parallelism, for instance, through Single Instruction Multiple Data (SIMD) operations. In FL, batching integrates naturally with the vectorized form of model updates, allowing entire tensors of weights to be combined into a single ciphertext. Prior studies [97–99] demonstrated the effectiveness of this approach under the Paillier scheme, reporting training speedups of 23×-93× and communication reductions of 66×-101×. The solutions proposed in this thesis (Chapters 5 and 6) employ the CKKS scheme via the open-source TenSEAL library [100], which natively supports batching. Nonetheless, the computational and communication overheads remain significant.

Alternative approaches focus on simplifying the encryption and decryption processes and limiting the set of supported arithmetic operations on encrypted data. For instance, [101] introduces a modified Paillier scheme that employs a tailored key-generation procedure to enable lightweight encryption. This method reduces the encryption time by up to 100×, making it highly effective for such a stage, though it offers no improvements for decryption or server-side computations. Similarly, [91] presents a custom lightweight HE scheme restricted to addition operations, which is sufficient for aggregation in federated learning. Their results show negligible overhead, only 1%-5% higher than training with plaintext models. However, this scheme is constrained to integer-only operations, thus degrading training quality.

Finally, another line of research explores the trade-off between security and efficiency by exploring a sparse encryption scheme, i.e., clients encrypt only a fraction of the tensors sent to the central server. This idea, first introduced in [94], demonstrated that adversarial attacks are ineffective as long as a sufficient portion of tensors remains protected, implying that full encryption may in fact be overly conservative. The main difficulty, however, is that pinpointing the appropriate encryption rate is challenging as it depends on both the model and the dataset, making it impractical for real-world deployments.

Chapter 3

Communication-efficient Federated Learning

Communication cost in FL primarily stems from the frequent exchange of model updates between clients and the central server, during the upload and download phases (stages *2* and *3* in Chapter 1). The frequent synchronization of model parameters for thousands of rounds generates massive data volumes running above capacity in many contexts. The communication cost depends thus on the number of model parameters, their bit-width precision, the number of participating clients (the concurrency), and ultimately the number of rounds. This overhead is especially problematic in resource-constrained environments such as mobile networks or IoT systems with limited bandwidth and intermittent connectivity. As a concrete deployed example, the measured download and upload byte-rates are on the order of $0.75\,\mathrm{MB/s}$ and $0.25\,\mathrm{MB/s}$, respectively [102], although actual rates may be significantly lower in low-power wide-area networks.

As reviewed in the background (Chapter 2), different optimization strategies exist either acting on the *synchronization* (for which FEDAVG represents the most notable example), *compression*, or by speeding up the convergence, to grant lower costs for the same accuracy thresholds by improving *local training* (e.g., FEDPROX) and *aggregation* (e.g., FEDOPT) stages. As previously stated, different data distributions intrinsically provide different and unique challenges requiring ad-hoc solutions.

This chapter introduces two strategies: the former uses a gradient-based strategy to assess whether layers have converged to a sufficiently refined form [68]. If the

layers have already converged, their continuous training and synchronization are not only futile but represent a waste of resources. By adopting a freezing logic, it is possible to remove these layers from further training and synchronization, thereby alleviating the communication cost without introducing computational overhead or reducing the model's utility. The latter instead tackles the communication cost under extreme label skew [103]. Under these settings, the compounded effects of partial participation and extreme label skew prevent reaching desirable accuracy levels within the available data plans of the devices. A dynamic concurrency management scheme is thereby proposed to overcome the limitations imposed by static participation schemes.

3.1 Communication-Efficient Federated Learning under Label Distribution Skew

3.1.1 Training optimization with Parameters Freezing

This subsection reviews the use of layer freezing as a strategy for optimizing model training. Existing work has been developed predominantly in centralized settings, with a primary focus on accelerating the training of Convolutional Neural Networks (CNNs) . The discussion then extends to FL-specific approaches, where freezing techniques have been explored to improve communication efficiency.

Parameters Freezing in Centralized Training

In centralized settings, most freezing approaches operate at a layer level. When one layer is frozen, its weights do not need to be updated, skipping thus the writing operations. If no other layer depends on a frozen layer, the back-propagation and gradient computation can be truncated, thereby drastically reducing computation time. Layer freezing methods can be broadly categorized as *switchable* or *irreversible*. Switchable freezing allows layers to be frozen and unfrozen across training iterations. Layer selection may be random [104] or guided by heuristics such as the magnitude of weight updates between epochs [105] or classification accuracy on a calibration set [106]. Irreversible freezing does not foresee any rollback when the parameters are entered in the frozen state, i.e., once the parameters are frozen, they can not

be unfrozen. A notable example of irreversible freezing in centralized training is presented in [107], where the procedure monitors the similarity of intermediate activations over time to determine layer convergence. Once a layer is deemed converged, it is frozen for the remaining epochs. However, this approach is unsuitable for federated settings, as monitoring intermediate activations would require direct access to client data on the server side.

Parameters Freezing in Federated Learning

Only a few works investigated the potential of parameter freezing in federated contexts and for different purposes. Specifically, a series of works [108, 109] applied layers freezing to reduce the computational effort during the local training and to incentivize the clients' participation at each round. In such a case, the freezing is *switchable* because the subset of frozen layers does change along different training rounds and across the clients. Concerning irreversible parameters' freezing, *FedPT* [110] proposed a method to reduce communication costs. The idea is to statically select a subset of layers to train from the beginning to the end of the learning process, leaving the remaining layers set to random values. Besides the substantial accuracy losses one may incur, selecting the trainable layers is a manual process that cannot ensure optimality.

Our preliminary analysis conducted in [67] pioneered the potential of irreversible freezing in the FL context, proposing an incremental freezing gating scheme that follows the topological order of the model's layers according to a manually defined schedule.

3.1.2 Overview of Automatic Layer Freezing

The Automatic Layer Freezing (ALF) mechanism was conceived as a component that can be seamlessly integrated into any synchronous FL framework, regardless of other optimization strategies adopted, with minimal effort. ALF operates entirely on the central server, avoiding introducing overhead for the edge clients, but rather progressively relieving them from full parameter updates. It monitors the convergence of each layer and freezes those that have reached a sufficient stability, thus reducing the communication overhead during the model synchronization phase.

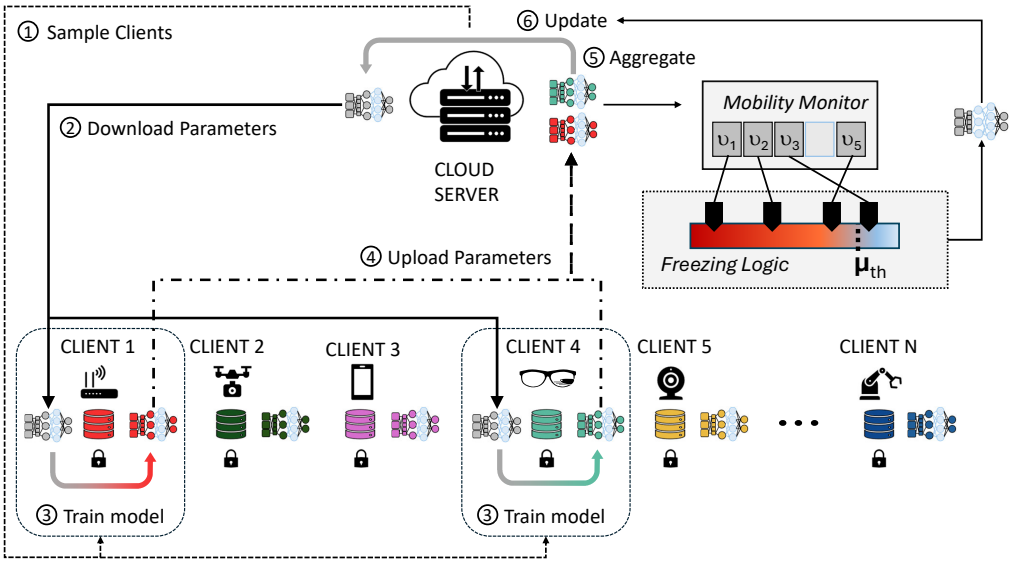


Fig. 3.1 Schematic view of a Federated Learning framework with ALF.

ALF comprises two modules, the *mobility monitor* and the *freezing logic*. The mobility monitor is responsible for assessing the convergence through a proxy metric called mobility. It computes for each layer of the model $l \in \mathscr{L}$ a mobility index s_l. To compute it, the mobility monitor aggregates the weights of the layer l across all clients and compares them with the previous round's weights. The freezing logic is in charge of defining the set of trainable layers $\mathscr{T}^r$ for the next round $r+1$, by comparing the mobility index s_l with a user-defined mobility threshold υ.

Fig. 3.1 illustrates the embedding of ALF in action over a generic FL flow and the main components of the mechanism using a CNN with 5 layers as an example. The figure shows a 5-layer CNN, where only four layers are trainable as the fourth layer was frozen in a previous round (as indicated by the light blue with reduced saturation). As the layer is frozen, it does not participate in the local training phase, and clients can be relieved from downloading and uploading it continuously, once they have downloaded the latest version from the server, thereby reducing the communication overhead.

The remaining layers (indicated by the colored nodes) continue to engage in the learning process. Once the model is uploaded, the mobility monitor computes the values of s_l for the updated layers (1, 2, 3, and 5). As shown in the figure, $s_3 < \upsilon$; consequently, the third layer of the global model is frozen. This information is then broadcast by the server in subsequent rounds, enabling clients to omit the local

training of the newly frozen layer. It is important to note that the model aggregation and the computation of the mobility index operate independently. Therefore, ALF remains compatible with any aggregation strategy.

3.1.3 ALF Workflow

ALF builds upon a synchronous FL framework where a central server coordinates a distributed set of low-power edge nodes (the clients) that are intermittently available. The pseudo-code reported in Algorithm 4 outlines the workflow of an FL implementation embedding ALF. Each client k possesses a private local dataset $\mathscr{D}_s$ with exclusive access. The server initializes the global model $\mathbf{w}^1$ with random weights (line 1) and specifies the initial set of trainable layers $\mathscr{T}^1$ (line 2). At the start, $\mathscr{T}^1$ includes all layers with learnable parameters, namely, the set $\mathscr{L}$.

The training process follows a synchronous FL protocol and encompasses R iterative communication rounds (lines 3–23). At the beginning of each round r, the server randomly selects a subset of clients $\mathscr{S}^r$ (line 4). Every chosen participant retrieves the global update stage flag ($globalUpdateStage_l$) indicating the most recent update for each layer l of the global model $\mathbf{w}^r$ (line 6). Whenever a participant detects that its local update stage ($localUpdateStage_l$) is outdated compared to the global one (line 7), it downloads the latest weights version of that specific layer from the server (line 8). Although this timestep mechanism introduces a minor form of metadata absent in standard FL pipelines, its contribution to the communication load is minimal. Each layer requires only 8 bytes, which is negligible relative to typical model sizes ranging from kBs to MBs. Subsequently, each participating device trains the received model parameters on its private dataset for a fixed number of epochs E, producing an updated local model $\mathbf{w}_s^r$ (line 9). The optimization process modifies only the layers within $\mathscr{T}^r$, i.e., the subset currently designated as trainable, to be uploaded and aggregated (lines 10–12). Upon collecting all client-side contributions, the server aggregates the updates to refine the global model $\mathbf{w}^{r+1}$ for the next communication round (line 12).

The server is further responsible for applying the ALF mechanism (lines 13–23), which comprises two complementary modules: the *mobility monitor* (lines 15–21) and the *freezing logic* (lines 22–23). For each active layer (line 14), the mobility monitor forms an aggregated representation $\boldsymbol{w}_l^r$ by computing a weighted average

Algorithm 4: Workflow of a standard FL framework with ALF.

1 $\mathbf{w}^1 \leftarrow$ Random model initialization
2 $\mathscr{T}^1 \leftarrow \mathscr{L}$
3 **for** r = *1,...,R* **do**
4 Sample a subset $\mathscr{S}^r$ of clients
`/* Clients Optimization */`
5 **for** $s \in \mathscr{S}^r$ **do in parallel**
6 **for** $l \in \mathscr{L}$ **do**
7 **if** $globalUpdateStage_l < localUpdateStage_l$ **then**
8 Download layer l of $\mathbf{w}^r$
9 $\mathbf{w}_s^r \leftarrow \textsc{Train}(\mathbf{w}^r, \mathscr{D}_s, E, \mathscr{T}^r)$
10 **for** $l \in \mathscr{T}^r$ **do**
11 Upload layer l of $\mathbf{w}_s^r$
`/* Server Optimization */`
12 $\mathbf{w}^{r+1} \leftarrow \textsc{Aggregate}(\mathbf{w}_s^r, s \in \mathscr{S}^r)$
`/* ALF */`
13 $\mathscr{T}^{r+1} \leftarrow \mathscr{T}^r$
14 **for** $l \in \mathscr{T}^r$ **do**
`/* Mobility Monitor */`
15 $\boldsymbol{w}_l^r \leftarrow \frac{1}{\sum_{s\in\mathscr{S}^r}|\mathscr{D}_s|} \cdot \sum_{s\in\mathscr{S}^r}|\mathscr{D}_s| \cdot \boldsymbol{w}_{l,k}^r$
16 $\Delta_l \leftarrow \boldsymbol{w}_l^r - \boldsymbol{w}_l^{r-1}$
17 $\Delta_l^{\text{abs}} \leftarrow$ Element-wise absolute values of Δ_l
18 $\boldsymbol{m}_l^r \leftarrow \alpha \cdot \boldsymbol{m}_l^{r-1} + (1-\alpha) \cdot \Delta_l$
19 $\boldsymbol{p}_l^r \leftarrow \alpha \cdot \boldsymbol{p}_l^{r-1} + (1-\alpha) \cdot \Delta_l^{\text{abs}}$
20 $\boldsymbol{n_l^r} \leftarrow$ Element-wise absolute values of $\frac{\boldsymbol{m}_l^r}{\boldsymbol{p}_l^r}$
21 $s_l \leftarrow$ Average all the elements of $\boldsymbol{n_l^r}$
`/* Freezing Logic */`
22 **if** $s_l < \upsilon$ **then**
23 $\mathscr{T}^{r+1} \leftarrow \mathscr{T}^{r+1} \setminus \{l\}$

of the participant-specific updates $\boldsymbol{w}_{l,k}^r$ (line 15). It then evaluates the inter-round deviation Δ_l between successive aggregated weights (line 16). To capture longer-term temporal dynamics, the Exponentially Weighted Moving Average (EWMA) is employed: $\boldsymbol{m}_l^r$ tracks the smoothed trend of the weight differences (line 18), while $\boldsymbol{p}_l^r$ follows the smoothed magnitude of those variations (line 19), using a decay factor $\alpha = 0.95$. The element-wise ratio of their absolute values yields the normalized vector $\boldsymbol{n}_l^r$, whose entries lie within $[0,1]$. Subsequently, the mobility monitor derives the scalar mobility index s_l as the mean of $\boldsymbol{n}_l^r$, obtaining the layer-level indicator $\boldsymbol{s}_l$ (line 21).

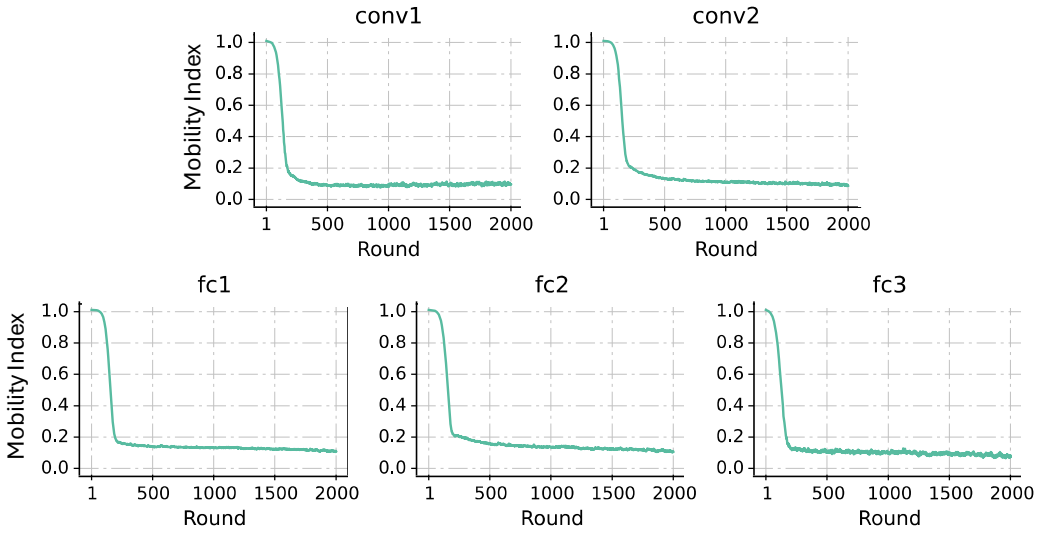


Fig. 3.2 Layer-wise mobility index for a 5-layer CNN (CNN-5) trained on CIFAR-10 with FedAvg (2000 rounds).

In the final stage, the freezing policy enforces a threshold-based gating rule (lines 22–23). Whenever the computed mobility index of a layer $\boldsymbol{s}_l$ falls below the predefined threshold υ (line 22), the corresponding layer l is deemed as converged and appended to the set of frozen modules in $\mathbf{w}^{r+1}$ (line 23).

The presented flow can work with any implementation of the TRAIN and AGGREGATE procedures, i.e., it is compatible with different two-sided optimization strategies, as ALF is implemented as an independent pluggable module. Section 3.1.5 provides an empirical demonstration of this feature for different local and global optimization strategies.

For a preliminary validation of the mobility index, we integrated the mobility monitor in a conventional FL scheme based on *FedAvg* without freezing. This is equivalent to running the workflow described in Algorithm 4, dropping lines 22–23. The results are reported in Figure 3.2, which shows the evolution over 2000 rounds of the layer-wise mobility index of a 5-layer CNN trained on the CIFAR-10 dataset (more details about the experimental setup are reported in Section 3.1.4). The plots show a clear trend: despite marginal oscillations, the mobility index decreases as the learning process proceeds, demonstrating that the proposed mobility monitor offers valuable information about the layer's convergence.

Table 3.1 Layer-by-layer overview of the CNN-5 architecture along with its memory requirements. The output size of the final layer depends on the target labels, 10 for CIFAR-10 and 100 for CIFAR-100.

Name	**Layer Type**	**Memory** (MB)
conv1	Convolution (2D)	0.019
conv2	Convolution (2D)	0.391
fc1	Linear	2.406
fc2	Linear	0.289
fc3	Linear	0.007 (0.074)
(Tot) Bias		**0.003 (0.003)**
(Tot) Weights		**3.112 (3.179)**

3.1.4 Experimental Setup

Models & Datasets

Models The experimental setup uses two CNNs with different memory footprints and number of layers, a 5-layer CNN (CNN-5) suited for tiny IoT applications and borrowed from [37], and a more complex 9-layer CNN (VGG-9) based on the VGG architecture [61]. The network comprises two convolutional layers (conv1 and conv2) with 64 5×5 filters, two linear layers (fc1 and fc2) with 394 and 192 neurons, respectively, and a final linear layer (fc3) that returns the prediction logits. The second network can be ported to mobile devices with higher computational resources. Tables 3.1 and 3.2 collect the memory footprint of each layer for the two CNNs. The total memory is a proxy of the communication cost for downloading (and uploading) the model from (to) the central server; the reported numbers refer to a standard 32-bit floating-point precision. We applied the ALF mechanism to the layers' weights only, as the biases require just a few kBs of data exchange.

Datasets The CNNs were trained using two datasets for image classification, CIFAR-10 and CIFAR-100 [111], using a standard split for training and testing. The CIFAR-10 dataset comprises 50,000 training samples and 10,000 testing samples of 32×32 RGB images across 10 classes. CIFAR-100 (similarly to CIFAR-10) consists of 32×32 RGB images, with 50,000 samples for training and 10,000 for testing. However, it provides a more complex classification task, as it includes 100 classes

Table 3.2 Layer-by-layer overview of the VGG-9 architecture along with its memory requirements. The output size of the final layer depends on the target labels, 10 for CIFAR-10 and 100 for CIFAR-100.

Name	**Layer Type**	**Memory** (MB)
conv1	Convolution (2D)	0.003
conv2	Convolution (2D)	0.071
conv3	Convolution (2D)	0.282
conv4	Convolution (2D)	0.563
conv5	Convolution (2D)	1.126
conv6	Convolution (2D)	2.251
fc1	Linear	8.002
fc2	Linear	1.002
fc3	Linear	0.020 (0.196)
(Tot) Bias		**0.007 (0.008)**
(Tot) Weights		**13.319 (13.495)**

instead of 10, featuring thus 600 samples per class (500 for training and 100 for testing).

We applied a standard pre-processing pipeline for data augmentation based on a random crop, followed by a random horizontal flip and cutout. The training datasets were split across 100 clients with non-IID and imbalanced data partitions to emulate a cross-device setting, following a Dirichlet distribution with concentration parameter of 0.3, as done in [112, 110]. This partitioning configuration provides each client with a limited number of data samples in certain classes, or potentially none, depending on the specific parameters.

FL strategies, Training & Evaluation

The experimental setup considered a synchronous standard cross-device setting comprising 100 clients, with a concurrency level of 10 (10% participation ratio). Each training encompassed 2000 rounds, with each single local training session executing $E = 5$ epochs. For local training epoch clients use the SGD optimizer with learning rate $\eta_l^0 = 0.1$, and the exponentially weighted schedule with decay parameter $\lambda = 4$, batch-size 50, and weight decay 1e$-$3.

For a comprehensive evaluation, we performed an extensive comparative study encompassing several strategies, each representing a distinct optimization knob (see Chapter 2). The considered approaches are summarized below:

- *Synchronization*. The standard FedAvg [37] is considered a baseline strategy. We investigated different numbers of local epochs to assess the impact of different synchronization frequencies. We thus ran experiments considering a number of local epochs $E = 5$, $E = 10$ as detailed in the table. As a side note, for $E = 10$, the number of rounds was halved (from 2000 to 1000) to retain the same number of local training steps with halved communication cost. This ensures a fairer comparison as it does not affect the complete training overhead by leaving the total number of training steps unaltered (though the clients should be able to complete the round).

- *Parameters Freezing*. For a thorough assessment of the efficiency of the proposed method, which is the automatic freezing mechanism, FedPT [110] was considered as an alternative strategy to a manual and fixed freezing mechanism. More specifically, a subset of the layers was frozen throughout the entire training process and never synchronized; the subset must be chosen prior to training begins. For a more comprehensive assessment, we considered two different subsets denoted with *PT-C* $_{(1,2)}$ and *PT-L* $_{(1)}$. In the former, all the convolutional layers are frozen; in the latter, only the first linear layer is frozen, which is the layer with the largest memory footprint for the two CNNs under analysis. In both implementations, the training was conducted within the FedAvg framework.

- *Aggregation Optimization*. We considered the FedAdam [112] framework with the Adam solver. Specifically, we set the Adam learning rate to 5e$-$3 with an exponential decay schedule, first momentum 0.9, second momentum 0.99, and adaptivity 1e$-$3.

- *Training Optimization*. To estimate the potential benefits of more efficient local training, we considered the training procedure proposed by FedProx [57], where a regularization term (referred to as the proximal term) was added to the client's loss function. We set the scaling factor of the proximal term to 1e$-$4.

The evaluation campaign monitored the evolution of the global model's classification accuracy on the test set. A moving average over a window of 30 communication

rounds was computed, following the procedure in [112], to smooth out oscillations caused by the inherent immobility of the training process and provide a more reliable assessment. In addition, the communication cost considered the synchronization phases, comprehensive of both upload and download phases. All experiments were conducted within an emulation framework implemented in PyTorch (version 1.9).

3.1.5 Results

Training Evaluation

Tables 3.3, 3.4, 3.5, and 3.6 summarize the accuracy and communication cost at the end of training for the selected networks and datasets. Each table is divided into three sections corresponding to the FL strategies under analysis (FedAvg, FedAdam, and FedProx), reporting results both with and without the ALF mechanism. Additionally, different values of the mobility threshold υ are included to illustrate its effect on training efficiency and accuracy outcomes.

The initial analysis focused on the FedAvg baseline strategy and additionally considers the evaluation of alternative optimization solutions for reducing communication, namely synchronization frequency tuning (FedAvg $E = 10$) and parameter freezing using FedPT (FedAvg + PT-C $_{(1,2)}$ and FedAvg + PT-L $_{(1)}$). Results on CNN-5 (Tables 3.3 and 3.4) show that, while these strategies lower communication costs, they do so at a significant accuracy penalty. For example, setting $E = 10$ reduces accuracy by 0.92% on CIFAR-10 and 3.70% on CIFAR-100. Similarly, FedAvg + PT-L $_{(1)}$ cuts communication by 77.11% (75.50%) for CIFAR-10 (CIFAR-100), but accuracy drops by 5.72% (2.36%). These results reveal the pitfalls of static and manual freezing, as freezing layers from the start of training can severely degrade performance, and poor selection of which layers to freeze leads to sub-optimal outcomes. For instance, PT-C $_{(1,2)}$ achieves modest savings of only 13.12% for CIFAR-10 while causing dramatic accuracy losses of up to 21.12%. Such observations underscore the necessity of an automatic mechanism capable of dynamically identifying which layers to freeze and when, enabling simultaneous improvements in both accuracy and communication efficiency.

As evidenced by the reported results, the proposed mechanism implemented with ALF is specifically designed to address this need. FedAvg with ALF outperforms all

Table 3.3 Training evaluation of different FL strategies, without and with ALF (+ ALF), subdivided into synchronization, local training, and aggregation. Results for CNN-5 on CIFAR-10 (unless otherwise noted, $E = 5$).

	Top-1	**ΔTop-1**	**CP**	**Savings**
FedAvg	74.83	-	121.59	-
FedAvg $_{E=10}$	73.91	-0.92	60.80	50.00%
FedAvg + PT-C $_{(1,2)}$	53.71	-21.12	105.64	13.12%
FedAvg + PT-L $_{(1)}$	69.11	-5.72	27.83	77.11%
FedAvg + ALF $_{\upsilon=0.11}$	77.10	2.27	68.12	43.98%
FedAvg + ALF $_{\upsilon=0.12}$	76.20	1.37	37.67	69.02%
FedAvg + ALF $_{\upsilon=0.13}$	75.87	1.04	27.88	77.07%
FedAdam	76.25	1.43	121.59	0.00%
FedAdam + ALF $_{\upsilon=0.11}$	76.52	1.69	36.17	70.25%
FedAdam + ALF $_{\upsilon=0.12}$	76.25	1.42	30.84	74.64%
FedAdam + ALF $_{\upsilon=0.13}$	75.85	1.02	27.81	77.13%
FedProx	75.05	0.22	121.59	0.00%
FedProx + ALF $_{\upsilon=0.11}$	77.35	2.52	69.89	42.52%

Table 3.4 Training evaluation of different FL strategies, without and with ALF (+ ALF), subdivided into synchronization, local training, and aggregation. Results for CNN-5 on CIFAR-100 (unless otherwise noted, $E = 5$).

	Top-1	**ΔTop-1**	**CP**	**Savings**
FedAvg	41.38	-	124.18	-
FedAvg $_{E=10}$	37.68	-3.70	62.09	50.00%
FedAvg + PT-C $_{(1,2)}$	26.12	-15.26	108.23	12.85%
FedAvg + PT-L $_{(1)}$	39.02	-2.36	30.42	75.50%
FedAvg + ALF $_{\upsilon=0.11}$	41.93	0.55	39.73	68.01%
FedAvg + ALF $_{\upsilon=0.12}$	39.87	-1.51	26.30	78.82%
FedAvg + ALF $_{\upsilon=0.13}$	39.56	-1.82	20.18	83.75%
FedAdam	42.03	0.65	124.18	0.00%
FedAdam + ALF $_{\upsilon=0.11}$	41.10	-0.28	26.20	78.90%
FedAdam + ALF $_{\upsilon=0.12}$	40.96	-0.42	22.78	81.66%
FedAdam + ALF $_{\upsilon=0.13}$	40.37	-1.01	19.98	83.91%
FedProx	41.17	0.21	124.18	0.00%
FedProx + ALF $_{\upsilon=0.11}$	41.44	0.06	38.40	69.08%

competitors on CNN-5 for CIFAR-10 across all the evaluation metrics, achieving higher accuracy than the baseline implementation. The mobility threshold υ serves as a control knob to balance communication cost: higher values of υ lead to earlier freezing of layers and thus lower transmission volumes. For instance, with $\upsilon = 0.13$, FedAvg with ALF incurs a communication cost nearly identical to FedAvg with PT-C $_{(1,2)}$ (27.88 GB vs. 27.83 GB) while achieving substantially higher accuracy (75.87% vs. 69.11%). On CIFAR-100, using larger υ values (0.12 and 0.13) results in slightly lower accuracy compared to baseline FedAvg, with a worst case scenario accounting for -1.82%. Yet, performance remains significantly better than that of alternative optimization strategies. These results demonstrate ALF's ability to flexibly trade off communication cost and accuracy depending on the choice of υ, adapting to both the dataset and the training dynamics.

The experiments based on FedProx and FedAdam further underscore both the efficiency and flexibility of ALF across different settings. For these strategies, the communication cost is initially identical to that of FedAvg, since the only differences lie in local training (FedProx) or server aggregation (FedAdam), with no changes occurring during the transmission phases. Across all strategies, learning proceeds over the same number of rounds, and the communication cost per round remains constant, regardless of the impact of model updates on the final predictive performance. In contrast, ALF dynamically modulates communication volume according to the evolution of clients' updates, gradually reducing the amount of data exchanged as training progresses. This adaptive behavior allows ALF to achieve substantial reductions in communication cost (up to 83.91%) while still maintaining competitive accuracy levels.

The results on the VGG-9 model (Tables 3.5 and 3.6) confirm that ALF remains effective even when applied to a more complex network architecture. Beyond this, the comparative analysis highlights a noteworthy interplay between ALF and the optimization achieved by advanced FL schemes. This effect becomes evident when comparing FedAvg and FedAdam with ALF configured at $\upsilon = 0.11$. While ALF consistently improves efficiency in both cases, the magnitude of the improvement differs substantially. The combination of FedAdam and ALF proves particularly advantageous, delivering simultaneous gains in accuracy and communication cost across all benchmarks. The effect is most pronounced on CIFAR-100: FedAvg with ALF achieves a modest 0.34% accuracy improvement with 25.51% communication savings, whereas FedAdam with ALF secures a 3.81% accuracy gain while reducing

Table 3.5 Training evaluation of different FL strategies, without and with ALF (+ ALF), subdivided into synchronization, local training, and aggregation. Results for VGG-9 on CIFAR-10 (unless otherwise noted, $E = 5$).

	Top-1	**ΔTop-1**	**CP**	**Savings**
FedAvg	81.12	-	520.29	-
FedAvg $_{E=10}$	81.23	0.10	260.15	50.00%
FedAvg + PT-C $_{(1,2)}$	43.05	-38.08	352.92	32.17%
FedAvg + PT-L $_{(1)}$	79.15	-1.97	208.50	59.93%
FedAvg + ALF $_{\upsilon=0.11}$	83.41	2.29	365.15	29.82%
FedAvg + ALF $_{\upsilon=0.12}$	83.10	1.97	242.11	53.47%
FedAvg + ALF $_{\upsilon=0.13}$	82.66	1.53	151.24	70.93%
FedAdam	84.14	3.02	520.29	0.00%
FedAdam + ALF $_{\upsilon=0.11}$	84.20	3.07	158.04	69.63%
FedAdam + ALF $_{\upsilon=0.12}$	84.16	3.04	145.35	72.06%
FedAdam + ALF $_{\upsilon=0.13}$	83.72	2.59	128.88	75.23%
FedProx	81.13	0.01	520.29	0.00%
FedProx + ALF $_{\upsilon=0.11}$	83.04	1.92	347.92	33.13%

Table 3.6 Training evaluation of different FL strategies, without and with ALF (+ ALF), subdivided into synchronization, local training, and aggregation. Results for VGG-9 on CIFAR-100 (unless otherwise noted, $E = 5$).

	Top-1	**ΔTop-1**	**CP**	**Savings**
FedAvg	49.18	-	527.17	-
FedAvg $_{E=10}$	48.78	-0.40	263.59	50.00%
FedAvg + PT-C $_{(1,2)}$	18.43	-30.74	359.80	31.75%
FedAvg + PT-L $_{(1)}$	43.70	-5.48	215.38	59.14%
FedAvg + ALF $_{\upsilon=0.11}$	49.52	0.34	392.71	25.51%
FedAvg + ALF $_{\upsilon=0.12}$	48.56	-0.62	140.50	73.35%
FedAvg + ALF $_{\upsilon=0.13}$	47.99	-1.19	87.46	83.41%
FedAdam	53.81	4.63	527.17	0.00%
FedAdam + ALF $_{\upsilon=0.11}$	52.99	3.81	132.79	74.81%
FedAdam + ALF $_{\upsilon=0.12}$	52.30	3.12	115.44	78.10%
FedAdam + ALF $_{\upsilon=0.13}$	51.55	2.37	104.40	80.20%
FedProx	49.86	0.69	527.17	0.00%
FedProx + ALF $_{\upsilon=0.11}$	49.77	0.60	392.44	25.56%

Table 3.7 Number of communication rounds before freezing for each layer of the CNN-5 model trained with different values of υ (0.11, 0.12, and 0.13). Results on CIFAR-10.

	FedAvg + ALF			**FedAdam + ALF**		
	0.11	**0.12**	**0.13**	**0.11**	**0.12**	**0.13**
conv1	269	254	238	246	234	209
conv2	617	478	423	407	356	318
fc1	1194	635	456	616	521	469
fc2	1212	658	489	646	562	511
fc3	214	200	189	212	200	187

Table 3.8 Number of communication rounds preceding each layer freezing of the CNN-5 model trained with different values of υ (0.11, 0.12, and 0.13). Results on CIFAR-100.

	FedAvg + ALF			**FedAdam + ALF**		
	0.11	**0.12**	**0.13**	**0.11**	**0.12**	**0.13**
conv1	683	447	348	381	372	325
conv2	716	490	381	499	439	383
fc1	641	420	319	430	371	323
fc2	512	336	259	254	222	205
fc3	451	279	223	168	162	158

communication by 74.81%. These results suggest that the advanced aggregation strategy of FedAdam accelerates the mobility's decreasing trend of network layers, allowing ALF's freezing logic to activate earlier. As a consequence, the integration of ALF with more sophisticated FL strategies not only amplifies communication savings but also enhances accuracy, reinforcing ALF's adaptability to different training dynamics.

The deepen the understanding of how the aggregation method's affects the layer mobility, an additional analysis was conducted to record the number of rounds required for each layer of the CNN-5 model to be frozen (Tables 3.7 and 3.8). Two main trends emerge from the analysis. First, higher values of the mobility threshold υ accelerate the freezing process across all layers and strategies. Second, and more importantly, for the same υ, layers consistently freeze earlier under FedAdam than under FedAvg. For example, with $\upsilon = 0.11$ on CIFAR-10 (Table 3.7), layer fc1 is frozen after 1194 rounds in FedAvg but after only 616 rounds in FedAdam.

This outcome underscores a central strength of ALF: rather than relying on rigid schedules or fixed heuristics, the mechanism adapts to the intrinsic learning dynamics of the chosen FL framework. When a more efficient scheme accelerates layer convergence, ALF detects the increased mobility and advances the freezing process accordingly. A comparison across datasets further reinforces this adaptability. On CIFAR-10, conv1 freezes before fc2, while on CIFAR-100 the order is reversed. Such differences demonstrate that the freezing order and timing are influenced not only by the underlying strategy but also by dataset characteristics. These findings confirm the necessity of an automatic mechanism such as ALF, capable of adjusting the freezing process to both algorithmic and data-driven conditions without manual intervention.

Accuracy vs. Payload Trade-offs

The overall communication volume can also be reduced by limiting the number of training rounds while still maintaining the target accuracy. Such an approach is simple and orthogonal to both the chosen framework and the optimization method, and thus broadly applicable. Yet, the achievable outcomes depend on the specific FL optimization strategy in use. To assess these benefits, we tracked the evolution of the global model's test accuracy over time and recorded the communication cost required to reach predefined accuracy thresholds A_{th}. The analysis was carried out for all three strategies under study, considering both their standard implementations and their integration with ALF.

Table 3.9 reports the results obtained for the CNN-5 model on the CIFAR-10 dataset. Dashes in the table denote cases where the operating strategy failed to reach the target accuracy (within the 2000 training rounds). For ALF, the values correspond to a configuration with $\upsilon = 0.11$. Consistent with prior findings [57, 112], advanced FL strategies such as FedProx and FedAdam converge faster than FedAvg, thereby reducing the communication required to attain a given accuracy. FedProx yields modest savings of only a few GBs (e.g., from 100.31 GB down to 97.70 GB when $A_{th} \geq 75.0\%$). FedAdam, by contrast, achieves notable gains in both accuracy and efficiency. When ALF is integrated, all strategies benefit from additional savings across accuracy levels. The effect is most evident for FedAdam, where the communication cost for achieving 76.5% accuracy is nearly halved, decreasing from 71.56 GB to 36.10 GB.

Table 3.9 Communication Payload (GB) of different FL strategies, without (Default) and with ALF (+ ALF) with $\upsilon = 0.11$. Results on CNN-5 for the CIFAR-10 dataset.

	FedAvg		**FedAdam**		**FedProx**	
A_{th}	**Default**	**+ ALF**	**Default**	**+ ALF**	**Default**	**+ ALF**
74.0	58.55	50.42	28.70	27.91	52.59	50.13
74.5	64.81	60.01	32.71	28.12	63.59	59.87
75.0	100.31	67.97	33.13	31.91	97.70	69.61
75.5	-	68.03	55.02	36.09	-	69.70
76.0	-	68.05	58.85	36.09	-	69.84
76.5	-	68.06	71.56	36.10	-	69.84
77.0	-	68.09	-	-	-	69.85

Table 3.10 Communication Payload (GB) of different FL strategies, without (Default) and with ALF (+ ALF) with $\upsilon = 0.11$. Results on CNN-5 for the CIFAR-100 dataset.

	FedAvg		**FedAdam**		**FedProx**	
A_{th}	**Default**	**+ ALF**	**Default**	**+ ALF**	**Default**	**+ ALF**
39.0	39.86	32.94	24.96	21.95	39.55	32.16
39.5	44.46	36.19	27.69	25.02	49.49	37.70
40.0	58.18	39.10	30.36	25.66	58.55	37.90
40.5	79.35	39.24	32.97	25.87	80.35	38.13
41.0	103.88	39.43	40.42	26.10	111.20	38.31
41.5	-	39.64	50.04	-	-	-
42.0	-	39.72	88.35	-	-	-

In line with common implementation practices, FL training features diminishing returns, i.e., conventional FL strategies incur substantial overhead for relatively modest accuracy improvements, typically amounting to only a few percentage points. For instance, in FedAvg the communication payload increases by 64.02 GB (from 39.86 GB to 103.88 GB) to raise the accuracy from 39.0% to 41.0%. Similarly, FedAdam requires an additional 15.46 GB (from 24.96 GB to 40.42 GB) to achieve a comparable improvement. In contrast, applying ALF on top of FedAdam reduces this cost dramatically: FedAdam with ALF reaches 41.0% accuracy with only 4.15 GB more data (from 21.95 GB to 26.10 GB). These results highlight that, as training progresses, only a limited subset of the model layers requires continued updating and synchronization with the global model. This finding supports the hypothesis that restricting updates to layers still contributing to learning progress yields a more efficient balance between accuracy and communication cost.

Table 3.11 Communication Payload (GB) of different FL strategies, without (Default) and with ALF (+ ALF) with $\upsilon = 0.11$. Results on VGG-9 for the CIFAR-10 dataset.

	FedAvg		**FedAdam**		**FedProx**	
A_{th}	**Default**	**+ ALF**	**Default**	**+ ALF**	**Default**	**+ ALF**
81.0	255.98	271.23	85.59	94.91	272.37	271.32
81.5	430.28	354.10	98.60	99.84	-	346.98
82.0	-	363.97	101.20	101.40	-	347.16
82.5	-	364.21	122.01	121.44	-	347.32
83.0	-	364.38	140.74	138.01	-	347.88
83.5	-	-	202.13	157.82	-	-
84.0	-	-	283.56	157.85	-	-

Table 3.12 Communication Payload (GB) of different FL strategies, without (Default) and with ALF (+ ALF) with $\upsilon = 0.11$. Results on VGG-9 for the CIFAR-100 dataset.

	FedAvg		**FedAdam**		**FedProx**	
A_{th}	**Default**	**+ ALF**	**Default**	**+ ALF**	**Default**	**+ ALF**
49.0	489.74	392.34	78.55	77.85	297.59	283.57
49.5	-	392.61	83.56	86.17	430.44	386.08
50.0	-	-	89.36	90.06	-	-
50.5	-	-	105.17	105.50	-	-
51.0	-	-	110.71	110.33	-	-
51.5	-	-	133.11	131.90	-	-
52.0	-	-	140.76	132.29	-	-
52.5	-	-	153.14	132.55	-	-
53.0	-	-	181.35	132.59	-	-
53.5	-	-	294.43	-	-	-

As previously discussed, layer freezing may slightly limit the achievable accuracy. In our experiments, FedAdam with ALF attained an accuracy about 1% lower than FedAdam without ALF. Nevertheless, ALF achieves 41% accuracy with a communication cost of only 26.10 GB, whereas the conventional FedAdam reaches just 39.5% accuracy with a comparable cost of 27.69 GB. An additional advantage of ALF lies in its flexibility: by adjusting the mobility threshold υ, it is possible to fine-tune the balance between accuracy and communication efficiency, as further elaborated in Section 3.1.5.

The results of the VGG-9 experiments (Tables 3.11 and 3.12) further reinforce the trends observed with the CNN-5 model. At the beginning of training, only a

Table 3.13 Communication Payload (GB) of FedAdam Default and + ALF for different values of the mobility threshold υ. Results on CNN-5 for the CIFAR-10 dataset.

		FedAdam + ALF									
A_{th}	**FedAdam**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.10**	**0.11**	**0.12**	**0.13**	**0.14**	**0.15**
74.0	28.70	28.70	28.63	28.49	28.45	23.79	28.34	27.91	27.51	27.29	24.50	22.46
74.5	32.71	32.71	32.37	32.41	32.37	32.11	31.80	28.12	30.52	27.71	24.51	22.46
75.0	33.13	33.13	32.92	32.96	32.91	32.65	32.32	31.91	30.60	27.71	24.51	22.48
75.5	55.02	52.65	47.24	46.40	45.78	45.02	40.77	36.09	30.63	27.72	24.52	-
76.0	58.85	58.63	54.43	53.30	51.97	45.72	40.84	36.09	30.75	-	-	-
76.5	71.56	65.89	62.23	60.96	52.02	45.95	40.93	36.10	-	-	-	-
77.0	-	87.11	74.94	61.24	52.36	-	-	-	-	-	-	-
77.5	-	96.87	-	-	-	-	-	-	-	-	-	-

small number of layers are frozen, which means that the effect of ALF on reducing communication is still limited. In this phase, achieving lower accuracy levels may even entail a slightly higher communication cost, although the difference amounts to only a few GBs.

As training advances, more layers stabilize and remain frozen, and the benefits of ALF gradually emerge. The transmission cost per round steadily decreases, leading to substantial savings at higher accuracy levels. This dynamic is particularly visible in CIFAR-100, where less than 1 GB of additional data is sufficient for FedAdam with ALF to improve accuracy from 51.5% to 53%.

Sensitivity Analysis

As discussed in Section 3.1.5, the mobility threshold υ serves as a control parameter for regulating the freezing mechanism. To test ALF sensitivity to the thresholding hyperparameter υ, a sensitivity analysis was conducted sweeping υ varying from 0.05 to 0.15 (with 0.01 increments). This analysis was conducted on the CNN-5 for both CIFAR-10 and CIFAR-100 benchmarks, summarized in Tables 3.13 and 3.14, respectively.

Although no closed-form rule exists to determine the optimal value of υ for a specific accuracy or communication budget, the analysis highlights that several configurations of υ can jointly achieve high accuracy and significant cost reductions. For CIFAR-10, eight ALF configurations attain 76.5% accuracy, with communication savings ranging from 7.92% ($\upsilon = 0.05$) to 49.54% ($\upsilon = 0.11$) compared to the baseline FedAdam. For CIFAR-100, four configurations achieve accuracy above 42.0% ($\upsilon \leq 0.08$). Overall, lower values of υ tend to deliver superior accuracy

Table 3.14 Communication Payload (GB) of FedAdam Default and + ALF for different values of the mobility threshold υ. Results on CNN-5 for the CIFAR-100 dataset.

		FedAdam + ALF									
A_{th}	**FedAdam**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.10**	**0.11**	**0.12**	**0.13**	**0.14**	**0.15**
39.0	24.96	24.96	24.96	24.80	21.94	21.83	21.58	21.95	21.82	19.51	17.25	15.81
39.5	27.69	27.69	27.69	26.56	24.97	24.88	24.06	25.02	22.18	19.70	17.43	-
40.0	30.36	30.36	30.36	28.62	27.58	27.20	26.37	25.66	22.34	19.87	17.47	-
40.5	32.97	32.97	32.97	33.05	31.61	31.99	30.06	25.87	22.66	-	-	-
41.0	40.42	40.42	39.47	38.63	38.01	36.82	30.23	26.10	22.68	-	-	-
41.5	50.04	49.84	47.05	42.76	41.05	-	30.30	-	-	-	-	-
42.0	88.35	63.19	60.00	53.32	44.66	-	-	-	-	-	-	-
42.5	-	82.36	66.90	-	-	-	-	-	-	-	-	-

relative to FedAdam without ALF, while still ensuring competitive (or even reduced) communication costs (as observed for CIFAR-100 with $\upsilon \leq 0.06$).

3.1.6 Discussion and Final Remarks

This chapter introduced ALF, an automated mechanism for layer freezing designed to reduce communication overhead in cross-device FL. ALF continuously monitors the evolution of clients' model updates to identify layers that have achieved stability, i.e., those whose updates provide little to no additional contribution to the predictive performance of the global model. Once detected, ALF applies a freezing strategy that locks the weights of these layers in both local and global models, eliminating the need for their synchronization, thereby relieving the associated communication costs. Extensive experiments under diverse settings confirm the effectiveness of ALF, demonstrating simultaneous gains in both model accuracy and communication efficiency. Furthermore, ALF can be seamlessly integrated into existing FL strategies with only minimal adjustments to the standard workflow. Importantly, all ALF operations are executed at the central server, ensuring that resource-constrained clients incur no extra computational or communication burden. Finally, when combined with more advanced FL strategies, ALF can synergize with complementary techniques at both the local training and central aggregation levels, thereby accelerating layer freezing and further driving accuracy improvements as well as communication savings through joint optimization.

3.2 Communication-Efficient Federated Learning under Extreme Label Skew

3.2.1 Introduction and Motivation

Despite the successful application of FL in multiple tasks [27, 26, 113] its applicability is still challenged by multiple factors, such as the limited and pay-per-use connection rates. Applying FL under limited communication might be overly expensive when clients have access to data representative of a limited subset of classes, a phenomenon introduced in Chapter 2 known as *Extreme Label Skew*. Within this scenario, the communication overhead required to achieve competitive accuracy levels may exceed the data bundles of the end devices.

In a typical FL implementation, the number of participating clients is set beforehand and remains unchanged across all rounds (for example, 5 clients in the experiments shown in Table 4.1). This creates a fixed trade-off between how frequently the global model is updated and the effectiveness of each update. Having a larger number of clients involved in each round can help mitigate the impact of biases from individual models, but it also forces each client to participate more often, quickly depleting their available data budget. Conversely, lowering the concurrency level enables more global updates, but can hurt accuracy because the aggregated updates become more skewed, leading to insufficiently representative updates. Therefore, using a static client participation rate can result in inefficient resource usage and degraded model accuracy.

Table 3.15 reports the results of our quantitative analysis on a five-layer CNN trained on CIFAR-10 under varying data distributions and communication budgets. The experiments involved 100 clients, each holding up to L classes of samples, while fixing the concurrency level C to 5 clients per round. The ideal case corresponds to $L = 10$, where every client has samples from all target classes. Extreme label skew ($L \leq 4$ and $L \leq 2$) significantly impairs the attainable accuracy, accounting for a worst-case degradation of 16.77% under the most stringent communication constraint (0.6 GB). This degradation arises because locally trained models become biased toward the available classes, thereby slowing down the convergence of the global model. These preliminary findings suggest that larger communication budgets are necessary to achieve higher accuracy, and that when bandwidth is limited,

Table 3.15 FL Top-1 Accuracy (%) on CIFAR-10 under different data distributions and communication constraints with L.

L	0.6 GB	1.2 GB	2.4 GB	4.8 GB
=10	82.47	82.63	82.48	82.81
≤4	73.76	77.38	79.01	80.22
≤2	65.70	73.50	76.08	78.75

complementary optimization strategies become essential. To this end, this work advocates for a more efficient resource management scheme leveraging a dynamic control of client participation throughout the learning process.

Addressing this inefficiency demands a dynamic participation scheme. A preliminary attempt was introduced in AdaFL [114], which developed a dynamic scheme increasing concurrency as training proceeds following a round-based single step policy. However, it lacks budget awareness, thereby resulting in sub-optimal resource utilization. The underlying intuition is that in the initial rounds, the model is far from convergence, and involving many concurrent clients for a single model release is just a waste of resources. Instead, higher participation is beneficial once the global model reaches a more mature representation. It is only at this stage that increasing the number of participants for a few final but highly representative releases enables an effective model refinement and mitigation of the label skew in just a few rounds, and hence with little communication cost. To effectively implement this dynamic scheme, the participation level should be refined according to the actual traffic consumption and the remaining data budget to ensure clients have adequate resources. This thesis introduces a practical solution based on a new resource management protocol named *Draft & Refine* (FedDR), which implements two distinct learning phases: drafting and refinement. In the drafting phase, the server enforces a limited concurrency level to generate many draft releases; in the refinement phase, the server enforces larger concurrency values to perform a few releases to refine the global model. A novelty introduced by FedDR lies in using the clients' available data traffic as a direct control variable to determine the duration of the two phases. This key feature ensures flexibility in adapting to different communication constraints.

3.2.2 Budget-driven Concurrency Management

The objective of FedDR is to maximize the Top-1 classification accuracy without exceeding the communication constraint P_{MAX}. P_{MAX} represents the bounded cumulative payload endured by each client, constrained by the data plan subscription. As a consequence, P_{MAX} limits the number of times a device s can participate throughout the FL process, hence, affecting the total iterations. Although the method is adaptable to uneven budgets, this work considered the case of a uniform participation budget with P_{MAX} equal for all clients.

FedDR is based on the synchronous FL scheme introduced in Chapter 2, which extends the most widely adopted FL algorithm, FedAvg [21]. As in FedAvg, the proposed budget-driven concurrency policy relies on the random selection of clients emulating stochastic eligibility criteria. However, FedDR introduces a key modification concerning the participation ratio, namely, the concurrency level in each communication round evolves dynamically as training progresses. To regulate this participation process, FedDR adopts a two-phase dynamic policy comprising two distinct phases: DRAFTING and REFINEMENT.

The DRAFTING phase employs a low concurrency level, denoted by C_{d}, with the goal of performing multiple rapid and low-cost global updates. This stage allows the system to produce a progressive refinement of the global model, guiding it toward a mature parameter representation. As the budget is close to depletion, the policy transitions to the REFINEMENT phase, during which the concurrency level increases to C_{r}, with $C_{\text{r}} > C_{\text{d}}$. This phase aims to refine the global model through broader client participation, thereby increasing data representation and class coverage at the aggregation stage.

As in FedAvg, the selected clients download the latest global model from the server, perform local training on their datasets, and then upload the updated model parameters. The server aggregates these updates through weighted averaging to produce the new global model. A distinctive feature of FedDR is its adaptive control of the participation rate based on each client's remaining communication budget. Specifically, during the upload phase, the server monitors clients by polling their available communication capacity. Clients remain eligible to participate in the drafting phase as long as their remaining budget satisfies $\geq (1-\rho) \times P_{\text{MAX}}$, where ρ is a predefined hyperparameter with $\rho \in (0,1)$. Once the remaining budgets of all

clients fall below this threshold, the system transitions to the refinement phase, in which C_r clients take part in subsequent rounds until all communication budgets are exhausted. The influence of the hyperparameters C_d, C_r, and ρ on performance is analyzed in Section 3.2.3-c.

3.2.3 Results

Experimental Setup

The experimental analysis used the CIFAR-10 and CIFAR-100 datasets as benchmarks, adopting the standard training and test splits. To emulate a distributed FL environment, the 50,000 training images were divided into 100 equal partitions, corresponding to the number of clients. Following the partitioning strategy in [21], each client was assigned at most L classes, and experiments were conducted with varying L to investigate different degrees of label skew. Specifically, $100 \times L$ data shards were created, each containing images from a single class, and assigned randomly to clients until each received 500 images.

All benchmarks were trained on a five-layer CNN from [21]. During each round, participating clients performed 20 steps of SGD with momentum 0.9, weight decay 1e$-$3, learning rate 0.01, and batch size 50. Model weights were stored as 32-bit floats, requiring 3.1 MB per client, resulting in 6.2 MB of communication per round (3.1 MB for pull and 3.1 MB for push).

FedDR was compared against three state-of-the-art FL baselines: FedAvg [21], AdaFL [114], and FedRS [71]. FedAvg uses a fixed number of participants C and the CE-Loss for local training; we varied $P \in \{5, 10, 20\}$ to study the accuracy-communication trade-off. AdaFL dynamically refines participation, increasing the cohort by one every ΔR rounds up to a maximum of 30 participants, starting from 5, with $\Delta R \in \{50, 100, 200, 400, 800\}$. FedRS maintains a fixed number of participants like FedAvg, but addresses label skew via a modified local loss that limits updates for missing classes; a grid-searched was conducted as follows: $\alpha \in \{0.5, 0.9\}$ and $P \in \{5, 10, 20\}$.

FedDR used ρ=0.93, C_d=5, and C_r=20 unless otherwise specified. All methods shared the same per-client communication budget, tested at 0.6, 1.2, 2.4, and 4.8 GB. For instance, with FedAvg and C=5, these budgets allow 2,000, 4,000, 8,000, and

Table 3.16 Results on CIFAR-10. The best results are in bold.

L	Method	0.6 GB	1.2 GB	2.4 GB	4.8 GB
=10	**FedAvg** C=5	82.47	82.63	82.48	82.81
	FedAvg C=5	73.76	77.38	79.01	80.22
	FedAvg C=10	71.77	76.30	78.87	79.97
≤4	**FedAvg** C=20	66.79	74.39	76.90	78.92
	FedRS	77.76	78.81	79.41	79.73
	AdaFL	75.01	78.33	79.75	80.93
	FedDR (Our)	**78.85**	**80.81**	**81.77**	**82.31**
	FedAvg C=5	65.70	73.50	76.08	78.75
	FedAvg C=10	65.19	72.94	76.27	78.78
≤2	**FedAvg** C=20	56.49	69.59	75.24	77.30
	FedRS	73.77	74.91	75.37	76.09
	AdaFL	68.13	75.30	78.82	79.36
	FedDR (Our)	**74.59**	**79.61**	**81.27**	**82.54**

16,000 training rounds, respectively. Final model performance was evaluated on the test set once the communication budget was exhausted, averaging accuracy over the last 30 rounds to smooth inherent fluctuations, as in [36].

Comparisons

Tables 3.16 and 3.17 report the achieved test accuracy for CIFAR-10 and CIFAR-100, respectively[1]. In each table, the row $L = 10$ represents an ideal scenario where each client has samples from all classes, serving as the baseline. The rows $L \leq 4$ and $L \leq 2$ for CIFAR-10, and $L \leq 5$ and $L \leq 2$ for CIFAR-100, correspond to extreme label skew distributions.

FedAvg consistently underperforms the baseline due to its static participation strategy, which enforces a skew-unaware trade-off between the number of participants and communication rounds. The resulting accuracy losses are especially pronounced under tight communication constraints. For example, in CIFAR-10 with $L \leq 2$ and 0.6 GB, the accuracy drops range from -25.98% (C=20) to -16.77% (C=5).

AdaFL, with its dynamic participation policy, marginally improves accuracy compared to FedAvg, yielding gains from 0.58% to 2.55% for CIFAR-10 and 0.33%

[1]For AdaFL and FedRS, the results correspond to their best-performing configurations.

Table 3.17 Results on CIFAR-100. The best results are in bold.

L	Method	0.6 GB	1.2 GB	2.4 GB	4.8 GB
=100	**FedAvg** C=5	45.51	45.82	46.12	46.38
$\leq$**5**	**FedAvg** C=5	32.77	38.15	40.66	42.51
	FedAvg C=10	31.89	39.71	43.14	44.89
	FedAvg C=20	24.68	35.62	41.30	45.29
	FedRS	39.15	41.66	42.57	43.14
	AdaFL	35.56	42.23	45.34	47.58
	FedDR (Our)	**44.02**	**47.89**	**48.94**	**49.57**
$\leq$**2**	**FedAvg** C=5	14.74	28.85	35.38	40.41
	FedAvg C=10	15.91	28.82	36.85	42.24
	FedAvg C=20	10.17	19.83	31.47	39.38
	FedRS	20.38	25.71	26.96	28.94
	AdaFL	16.24	30.26	39.57	44.26
	FedDR (Our)	**25.43**	**41.05**	**47.62**	**49.72**

to 2.79% for CIFAR-100. A closer inspection of experimental logs highlights two limitations of AdaFL: (*i*) the incremental increase of participants occurs prematurely, while the global model is still far from convergence, wasting communication resources; (*ii*) using a fixed ΔR to trigger participant growth is less effective than a budget-aware approach that leverages the actual communication constraints, as proposed in FedDR.

FedDR consistently outperforms both FedAvg and AdaFL across all scenarios. For CIFAR-10, FedDR achieves accuracy gains of 2.09% to 8.89% over FedAvg and 1.38% to 6.46% over AdaFL; for CIFAR-100, the gains range from 4.28% to 12.20% over FedAvg and 1.98% to 10.79% over AdaFL. From a resource-efficiency perspective, FedDR reaches comparable accuracy with significantly lower communication costs. For instance, in CIFAR-10 with $L \leq 2$, FedDR achieves 79.61% accuracy using only 1.2 GB, whereas AdaFL requires 4.8 GB to reach a similar level (79.36%).

FedRS demonstrates mixed performance, as it can surpass FedAvg under strict communication budgets but often fails to maintain these gains as the budget increases. This is particularly evident in CIFAR-100 with $L \leq 2$ at 4.8 GB, where FedRS achieves only 28.94% accuracy compared to 45.29% for FedAvg (C=20). Overall,

Table 3.18 Sensitivity analysis on FedDR hyper-parameters. Results on CIFAR-100 with $L \leq 5$ and communication constraint of 1.2 GB.

		ρ		
C_d	C_r	**0.91**	**0.93**	**0.95**
3	**20**	47.81	47.93	48.09
5	**10**	44.51	44.99	45.10
5	**20**	47.49	47.89	47.48
5	**30**	48.62	48.56	48.61
10	**20**	43.80	44.00	43.97

FedRS never matches the performance of FedDR, which remains the most accurate and resource-efficient method across all tested scenarios.

FedDR consistently delivers top performance across all communication constraints due to its dynamic, budget-aware concurrency management. Interestingly, it can even surpass the baseline accuracy achieved under fully balanced data distributions. For example, in CIFAR-100 at 4.8 GB, training under extreme label skew with $L \leq 2$ yields 49.72% accuracy, which is 3.34% higher than the 46.38% obtained with $L = 100$. This improvement arises because, during the refinement phase, the larger concurrency value (20 instead of 5) refines the model with a broader set of samples, providing a better approximation of the global data distribution despite label skew, thereby improving stability and accelerating convergence. These results highlight the critical role of adaptive client participation and underscore the effectiveness of FedDR's control strategy.

FedDR Sensitivity Analysis

In the previous experiments, the hyperparameters of FedDR were fixed to $\rho = 0.93$, $C_d = 5$, and $C_r = 20$. In this subsection, we perform a sensitivity analysis to assess the impact of these parameters on performance. Specifically, we explored various combinations of C_d and C_r while sweeping ρ. For brevity, Table 3.18 reports a representative subset of results, focusing on CIFAR-100 with $L \leq 5$ and a communication budget of 1.2 GB.

The results confirm that FedDR is robust under varying settings. The parameter ρ exhibits minimal influence on accuracy, whereas C_d and C_r have a more pronounced effect. For instance, with $\rho = 0.95$, accuracy ranges from 43.97% ($C_d = 10$, $C_r = 20$)

to 48.61% ($C_d = 5$, $C_r = 30$), yet all configurations remain competitive. Notably, every tested setting surpasses the best results of FedAvg and AdaFL (39.71% and 42.23%, respectively, as shown in Table 3.17).

Overall, the analysis suggests that selecting lower C_d values (e.g., 3 or 5) in combination with higher C_r values (e.g., 20 or 30) is preferable to maximize accuracy gains.

3.2.4 Discussion and Final Remarks

This work presented FedDR, a novel resource-aware protocol designed to optimize the accuracy of classifiers trained under cross-device FL. FedDR addresses a key limitation of conventional FL algorithms: when clients' datasets belong to a subset of the target classes, thereby poorly representing the global distribution, standard approaches often fail to achieve competitive accuracy within limited communication budgets. By dynamically refining clients' participation based on the available communication resources, FedDR provides an effective solution, achieving up to 12.20% improvement in accuracy under the same budget in the best-case scenario.

Chapter 4

Energy-Efficient Federated Learning under Extreme Label Skew

To address the high energy consumption of edge clients under extreme label skew, this chapter introduces FedGAP (Federated Learning with Gradient-Aware Participation) [115]. FedGAP promotes an energy-aware participation strategy that adapts the client concurrency on a round-by-round basis according to the evolution of the global model's pseudo-gradient. By identifying stagnant phases where convergence slows and temporarily enlarging the cohort, FedGAP enables the system to escape suboptimal regions, accelerate training, and reduce unnecessary computational and communication overhead.

4.1 Introduction and Motivation

Despite its successful application across multiple IoT use cases [116, 27, 117, 33], FL applicability is still limited by several significant challenges, among which is the energy consumption on the client side. Energy consumption is a critical factor in FL as it involves battery-powered clients such as smartphones or edge gateways to sustain the repeated training steps and synchronization phases required across multiple rounds [65]. These costs depend on several factors, including model complexity, number of local updates, data exchange volume, and network conditions. Consequently, optimizing energy efficiency in FL requires carefully

Table 4.1 Energy cost of FL on CIFAR-10 under homogeneous label distribution (L=10) and extreme label skew ($L\leq$4).

L	C	$A_{\text{th}} > 79.0\%$	$A_{\text{th}} > 79.5\%$	$A_{\text{th}} > 80.0\%$
=10	5	28.3	31.0	35.9
	5	227.2	-	-
≤4	10	298.7	317.3	455.2
	20	398.2	514.4	599.0

balancing computation and communication to maintain model performance without overburdening client devices.

The energy consumption becomes particularly critical when data poorly represent the collective distribution, with clients having a limited subset of the target classes. Under such an extreme label skew condition, convergence is particularly slow, leading to massive energy consumption to attain competitive accuracy levels. Moreover, the limited concurrency levels further compound this effect, exacerbating unstable and slow convergence. Indeed, under severe skew, even the union of the participating clients' datasets may lack samples from specific classes. This additionally slows convergence and could notably degrade the achievable accuracy.

A key factor behind these challenges is the choice of the concurrency level, which plays a pivotal role in balancing training effectiveness against the energy budget of participating clients. Preliminary insights, reported in Table 4.1, illustrate this relationship using as a benchmark a five-layer CNN trained on the CIFAR-10 dataset (details of the setup are provided in Sec. 4.3.1). The metric of interest is the average energy expenditure per client, estimated as the mean number of rounds in which a client must participate for the global model to achieve a predefined accuracy threshold A_{th}. The target accuracy thresholds are selected close to the corresponding converged accuracy within 5,000 rounds, to enable an iso-accuracy comparison.

The experiments, conducted with 100 clients, vary both the number of classes per client L and the concurrency level C. In the balanced case (L=10), target accuracies are reached within only a few dozen rounds, with energy costs ranging between 28.3 (for $A_{\text{th}} > 79.0\%$) and 35.9 (for $A_{\text{th}} > 80.0\%$). By contrast, in highly skewed settings ($L \leq 4$), both convergence speed and energy efficiency degrade significantly. For instance, with a low concurrency (C=5), the training process manages to surpass only the lowest threshold $A_{\text{th}} > 79.0\%$ (failing at 79.5% and 80.0%), while the

corresponding energy cost becomes 8.03× higher compared to the balanced case (227.2 vs. 28.3).

Larger concurrency values ($C = 10$ or $C = 20$) alleviate some of the adverse effects of label skew by providing the aggregation step with more representative updates, but this comes at a steep price, namely, the energy demand escalates dramatically, reaching up to 16.69× higher for $A_{\text{th}} > 80.0\%$ (599.0 vs. 35.9). These results reveal that fixing the concurrency introduces a fundamental trade-off: limited concurrency hinders accuracy, while larger values inflate energy costs. Under this observation, the concurrency level emerges as a key control parameter in navigating the accuracy–efficiency balance of cross-device FL.

This work introduces *Federated Learning with Gradient-Aware Participation* (FedGAP) , a novel resource management strategy that adapts the concurrency dynamically according to the stage of training. The proposed empirical analysis suggests a clear pattern: in the early rounds, when the global model is far from convergence, low concurrency values already provide sufficiently informative updates, while involving larger participating pools only leads to unnecessary energy expenditure. Conversely, as training progresses and the model enters a stagnation phase with limited accuracy improvements, increasing the concurrency becomes advantageous to counteract the bias in local updates and accelerate convergence.

FedGAP leverages this principle by employing a concurrency scaling mechanism that is triggered whenever stagnation is detected, relying on the evolution of the global model's pseudo-gradient as a monitoring signal. Through this adaptive control, FedGAP strikes a balance between energy efficiency and training quality, proving particularly effective under extreme label skew.

Related Optimization Approaches

As a brief recap of Chapter 2, extreme label skew has been mainly tackled by mitigating local training updates inconsistencies with FedRS [71] being the most notable example. Other strategies investigated label augmentation, model aggregation, and client orchestration strategies. This work proposes the automatic management of concurrency to deal with the issue of energy consumption required to achieve competitive accuracy levels. This approach has received considerably less attention. AdaFL [114] constitutes an early attempt, employing a simple incremental rule where

concurrency is increased at fixed intervals. Our approach, FedGAP, departs from this manual strategy in two crucial ways: (*i*) it adapts the concurrency in response to the observed training dynamics of the global model rather than at pre-scheduled intervals, and (*ii*) it maintains flexibility across diverse label distributions and dataset complexities. As shown in Section 4.3, this design enables FedGAP to simultaneously reduce client-side energy consumption and enhance model accuracy. The work introduced in Chapter 3, FedDR [103], uses a two-phase protocol consisting of a long initial phase with small concurrency values followed by a shorter phase with larger ones. Despite being effective, FedDR is a budget-based method. In other words, it requires prior knowledge of clients' residual data budgets to govern the transition between phases.

4.2 Methodology

This work is built upon a centralized, synchronous FL framework based on FedAvg [21], introduced in Chapter 2. Table 4.2 reports the notation specifically used within this chapter.

4.2.1 Energy Model & Optimization Goal

We adopt the energy model described in [118]. The total energy expenditure of a client device s in a training round r, denoted by E_s^r,is expressed as the sum of two components:

$$E_s^r = E_s^{\text{comp}} + E_s^{\text{comm}}. \tag{4.1}$$

Here, E_s^{comp} accounts for the energy required to compute the local update. This term depends on hardware-related factors such as CPU architecture and settings, clock frequency, model size, and the number of local training iterations. The second term, E_s^{comm}, represents the communication energy spent when interacting with the server, i.e., for downloading and uploading the model parameters. Its value is determined by the communication technology (e.g., WiFi, 5G) and the amount of data transmitted.

Table 4.2 Summary of Notation.

Notation	Description
EWMA	Exponentially Weighted Moving Average
$\|\|\cdot\|\|_{\text{abs}}$	Element-wise absolute value of a tensor
$\boldsymbol{\delta}^r$	Pseudo-gradient at round r
$\mathbf{g^r}$	Gradient alignment at round r
g^r	Gradient alignment score at round r

The cumulative energy consumption of a client s over the entire training process can be expressed as:

$$E_s^{\text{tot}} = \sum_{r\in[1,R]} \mathbb{1}_{s\in\mathscr{S}^r} \cdot E_s^r \,, \tag{4.2}$$

where $\mathbb{1}_{s\in\mathscr{S}^r}$ is an indicator function that equals 1 if client s is selected to participate in round r, and 0 otherwise.

In cross-device FL, client devices exhibit heterogeneous hardware capabilities and network characteristics, resulting in widely varying latency and power consumption profiles. This diversity makes precise energy estimation both challenging and unreliable. To abstract away device-specific differences, we introduce a proxy metric referred to as *energy cost*, which quantifies the average number of rounds in which a client participates during training. Formally:

$$\text{Energy Cost} = \frac{\sum_{s\in\mathscr{S}} \sum_{r\in[1,R]} \mathbb{1}_{s\in\mathscr{S}^r}}{|\mathscr{S}|} \,. \tag{4.3}$$

This metric provides a clear, hardware-agnostic measure of the average effort expected from a client, allowing for straightforward comparisons of energy demands across different concurrency levels and experimental setups. In other words, Eq. (4.3) functions as a practical, device-neutral proxy for systematic energy assessment in federated learning scenarios. The goal of the proposed FedGAP strategy is to reduce the energy cost needed to reach a desired accuracy level by implementing a gradient-aware mechanism that adaptively adjusts the concurrency in response to the progression of global model training.

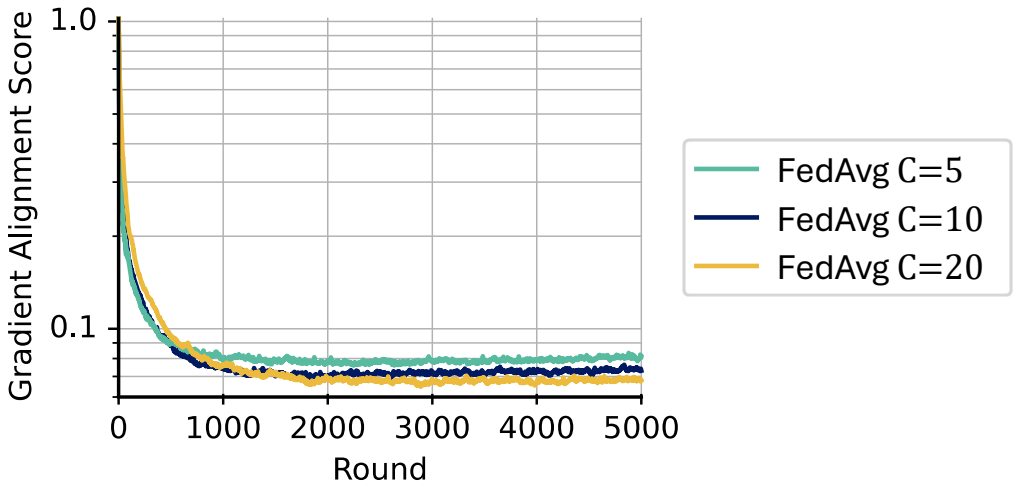


Fig. 4.1 Gradient alignment score during training on CIFAR-10 under extreme label skew ($L \leq 4$) using FedAvg with different concurrency levels (C).

4.2.2 Gradient-Aware Participation

Training a deep neural network is an optimization problem where the goal is to find the set of weights that minimizes a given loss function. As shown in the theoretical analysis of [66], in optimal settings, the evolution of the model weights $\mathbf{w}^r$ can be divided into two distinct phases. During the transient phase, the weights change significantly across rounds, and the corresponding pseudo-gradients $\boldsymbol{\delta}^r$ are consistently aligned, effectively guiding the model toward convergence. Once the model approaches the optimum, it enters a stationary phase in which the pseudo-gradients oscillate around zero, resulting in only minor updates to the weights. In practical cross-device FL scenarios, however, factors such as extreme label skew and partial client participation introduce instabilities preventing convergence. These non-idealities can cause the model to become trapped in sub-optimal regions, leading to a stagnant phase where pseudo-gradients remain non-zero and fail to drive meaningful progress toward the global optimum.

The central principle behind FedGAP is the continuous monitoring of the global model's pseudo-gradients across rounds in order to identify stagnant phases and dynamically adjust the concurrency. To realize this in practice, this work introduces the *gradient alignment score*, a metric that serves as a lightweight probe of training progress in FL. For a sliding window of λ rounds, the gradient alignment vector $\mathbf{g}^r$

is formally defined as:

$$\mathbf{g}^r = \frac{\left\| \sum_{i=r-\lambda+1}^{r} \boldsymbol{\delta}^i \right\|_{\text{abs}}}{\sum_{i=r-\lambda+1}^{r} \|\boldsymbol{\delta}^i\|_{\text{abs}}}, \tag{4.4}$$

where $\boldsymbol{\delta}^i$ denotes the pseudo-gradient at round i, and $\|\cdot\|_{\text{abs}}$ indicates the element-wise absolute value. Intuitively, $\mathbf{g}^r$ captures the alignment among recent pseudo-gradients at the parameter level, with values close to one reflecting consistent progress in a common direction and values close to 0 indicating oscillations or stagnation.

Direct computation of Eq. (4.4) would require storing λ pseudo-gradients, with an associated memory overhead growing proportionally to λ, thereby limiting its scalability. To overcome this limitation, FedGAP adopts an *exponentially weighted moving average* (EWMA) approximation that recursively updates the numerator and denominator, thereby eliminating the memory burden. More formally, FedGAP considers two tracking vectors:

$$\mathbf{m}^r = \frac{2}{\lambda+1}\boldsymbol{\delta}^r + \left(1 - \frac{2}{\lambda+1}\right)\mathbf{m}^{r-1}, \tag{4.5}$$

$$\mathbf{p}^r = \frac{2}{\lambda+1}\|\boldsymbol{\delta}^r\|_{\text{abs}} + \left(1 - \frac{2}{\lambda+1}\right)\mathbf{p}^{r-1}, \tag{4.6}$$

where λ determines the effective length of the observation window. The gradient alignment vector at round r can then be computed as:

$$\mathbf{g}^r = \frac{\|\mathbf{m}^r\|_{\text{abs}}}{\mathbf{p}^r}. \tag{4.7}$$

Finally, averaging the elements of $\mathbf{g}^r$ yields the scalar *gradient alignment score* g^r, which provides a compact indicator of the training dynamics and serves as the trigger for FedGAP's adaptive concurrency scaling.

The gradient alignment score provides an aggregated indicator of the convergence behavior, with values constrained to the interval $[0,1]$. Scores approaching 1 denote that consecutive pseudo-gradients are consistently aligned, a typical signature of the transient learning phase in which the model continues to make steady progress. Conversely, values near 0 reflect oscillations of pseudo-gradients around zero, signaling that the model is entering the convergence regime where weight updates become marginal. For less ideal but more practical settings, however, the score does not necessarily decay to zero; instead, it may plateau at intermediate values, indicating

Algorithm 5: FedGAP Control Logic.

$g_{\min} \leftarrow 1$
$t \leftarrow 0$
for $r = 1,...,R$ **do**
    if $g^r < g_{min} - \varepsilon$ **then**
        $g_{\min} \leftarrow g^r$
        $t \leftarrow 0$
    else
        $t \leftarrow t + 1$
    if $t > \lambda$ **then**
        Increment the concurrency by one.
        $g_{\min} \leftarrow 1$
        $t \leftarrow 0$

that the model has become trapped in a sub-optimal region and is unable to advance further. FedGAP leverages this signal to activate its adaptive concurrency scaling mechanism, as detailed in the following sections.

To further validate this assumption through empirical evidence, Figure 4.1 illustrates the trajectory of the gradient alignment score during a 5000-round training session on CIFAR-10 using a five-layer CNN. The experiment considers the case of extreme label skew ($L \leq 4$) and applies FedAvg with different concurrency levels C, consistent with the setup described in Table 4.1. At the start of training, g^r is close to 1 and exhibits a rapid decline within the initial rounds. In this transient stage, the impact of concurrency is negligible, as shown by the nearly overlapping curves for $C = 5$, $C = 10$, and $C = 20$. After sufficient rounds, however, the score stabilizes at values that depend on the concurrency. Notably, larger concurrency values lead to lower final alignment scores, indicating that involving more participants enhances the progression of learning. These observations motivate the adoption of a dynamic participation mechanism: smaller concurrency values should be adopted for early training phases to conserve energy without sacrificing model quality, whereas increasing the concurrency in later stagnant phases enables the model to escape suboptimal regions and converge toward superior solutions.

4.2.3 FedGAP Implementation

FedGAP adopts a threshold-based control mechanism driven by the gradient alignment score, as summarized in Algorithm 5. The variable $g_{\min}$ records the minimum gradient alignment score observed so far, while the counter t tracks the number of consecutive rounds in which the score fails to show a significant decrease. Both variables are initialized at the beginning of training, with $g_{\min} = 1$ and $t = 0$ (lines 1–2). Whenever g^r does not decrease by at least ε over the most recent λ rounds, the concurrency is increased by one (lines 3–10). After each adjustment, the concurrency is kept fixed for at least λ rounds, ensuring that the gradient alignment score has sufficient time to stabilize under the updated configuration. This is achieved by resetting $g_{\min}$ and t to their initial values (lines 11–12). The influence of the hyperparameters ε and λ is further examined in Sec. 4.3.3. Finally, it is important to emphasize that the computation of the gradient alignment score and the execution of FedGAP's control logic is performed centrally on the server, thereby imposing no additional computational burden on the participating clients.

4.3 Results

4.3.1 Experimental Setup

Datasets. The proposed approach was validated on two standard image classification benchmarks, CIFAR-10 and CIFAR-100, using their conventional train/test splits. The training set of each dataset was divided into non-overlapping shards and distributed across 100 clients. Following the widely used sharding partitioning scheme in [21], each client was assigned samples corresponding to at most L distinct labels. Concretely, the dataset was split into $100 \times L$ label-homogeneous shards, which were then randomly allocated to clients.

Model and Training Configuration. As the base architecture, we adopted the five-layer CNN introduced in [21]. Federated training was conducted for a total of 5000 rounds. In each round, participating clients executed $K = 20$ local training steps using SGD with the following hyperparameters: momentum of 0.9, weight decay of $1e{-3}$, learning rate of $1e{-2}$, and a batch size of 50.

Baselines and Implementation. FedGAP was benchmarked against three representative methods: FedAvg [21], FedRS [71], and AdaFL [114].

- **FedAvg**: the standard FL algorithm with a fixed concurrency of C workers, trained using the cross-entropy loss. The experiments tested $C \in \{5, 10, 20\}$ to study the trade-off between accuracy and energy cost.
- **FedRS**: an extension of FedAvg that modifies the local loss function to compensate for missing labels. As in FedAvg, we varied $C \in \{5, 10, 20\}$, and we additionally tuned the hyperparameter $\alpha \in \{0.5, 0.9\}$, retaining the best-performing configuration. Including FedRS in our comparisons allows us to highlight the limitations of loss-level adjustments, the most explored option of the previous literature, in terms of both model quality and energy efficiency.
- **AdaFL**: a dynamic policy that linearly increases the concurrency, serving as a direct competitor to FedGAP. Starting from an initial $C = 5$, AdaFL incrementally adds one worker every ΔR rounds until reaching a maximum size of 30. The core hyperparameter Δ was tuned between $\{100, 200\}$.
- **FedGAP**: the proposed method. Unless otherwise specified, we set the observation window to $\lambda = 100$ and the threshold to $\varepsilon = 5 \times 10^{-4}$, starting with 5 workers and allowing up to 30 participants.

Evaluation Metrics. Model performance was measured as classification accuracy on the test set. To mitigate short-term fluctuations, the reported results refer to a 30-round moving average, a practice widely adopted in FL evaluations [36]. In addition, the energy efficiency assessment used the *energy cost* metric defined in Eq. (4.3), which quantifies the average number of rounds a client contributes to in order to reach predefined accuracy thresholds.

4.3.2 Results and Discussion

Tables 4.3 and 4.4 compare the energy cost of FedGAP with that of the baseline methods on CIFAR-10 and CIFAR-100, respectively. The reported values represent the average energy cost required to reach predefined accuracy thresholds A_{th}, while numbers in parentheses denote the energy overhead of each baseline relative to

Table 4.3 Comparisons on CIFAR-10.

L	Method	$A_{\text{th}} > 79.0\%$	$A_{\text{th}} > 79.5\%$	$A_{\text{th}} > 80.0\%$
$\leq$**4**	**FedGAP** (Our)	**159.6**	**202.7**	**235.2**
	AdaFL	226.3 (1.42×)	235.7 (1.16×)	306.0 (1.30×)
	FedAvg C=5	227.2 (1.42×)	-	-
	FedAvg C=10	298.7 (1.87×)	317.3 (1.57×)	455.2 (1.94×)
	FedAvg C=20	398.2 (2.49×)	514.4 (2.54×)	599.0 (2.55×)
	FedRS C=5	-	-	-
	FedRS C=10	269.2 (1.69×)	-	-
	FedRS C=20	352.6 (2.21×)	-	-
$\leq$**3**	**FedGAP** (Our)	**240.5**	**281.2**	**351.8**
	AdaFL	328.8 (1.37×)	363.4 (1.29×)	417.9 (1.19×)
	FedAvg C=5	-	-	-
	FedAvg C=10	374.1 (1.56×)	479.1 (1.70×)	-
	FedAvg C=20	547.4 (2.28×)	615.4 (2.19×)	965.6 (2.74×)
	FedRS C=5	-	-	-
	FedRS C=10	-	-	-
	FedRS C=20	-	-	-

The lowest energy costs achieved are highlighted in bold. A dash (-) is used to indicate that the target accuracy has not been met.

FedGAP. Results are presented under different degrees of label skew, namely $L \leq 4$ and $L \leq 3$ for CIFAR-10, and $L \leq 5$ and $L \leq 3$ for CIFAR-100. For both FedRS and AdaFL, the results provided report the best-performing configuration of their respective hyperparameters.

Overall, FedGAP consistently reaches the highest accuracy thresholds while incurring the lowest energy cost. Across all experimental conditions, alternative methods exhibit significantly higher overheads, ranging from 1.16× to 2.74× on CIFAR-10 and from 1.15× to 2.60× on CIFAR-100.

The performance of FedAvg is strongly influenced by the concurrency C. Fixing C imposes a static compromise between accuracy and energy as larger concurrency values are necessary to attain competitive accuracy, but they substantially increase energy consumption. For instance, on CIFAR-10 with $L \leq 4$, surpassing $A_{\text{th}} > 79.0\%$ requires increasing the concurrency from $C = 5$ to $C = 20$, which nearly doubles the energy cost (from 227.2 to 398.2). Conversely, smaller concurrency values reduce energy usage but limit the achievable accuracy. In particular, FedAvg with $C = 5$ reached only the lowest accuracy target on CIFAR-10 with $L \leq 4$, while failing

Table 4.4 Comparisons on CIFAR-100.

L	Method	$A_{\text{th}} > 42.0\%$	$A_{\text{th}} > 44.0\%$	$A_{\text{th}} > 46.0\%$
$\leq$**5**	**FedGAP** (Our)	**154.9**	**211.2**	**360.4**
	AdaFL	217.4 (1.40×)	286.9 (1.36×)	506.7 (1.41×)
	FedAvg C=5	-	-	-
	FedAvg C=10	261.7 (1.69×)	431.9 (2.04×)	-
	FedAvg C=20	340.2 (2.20×)	504.8 (2.39×)	875.0 (2.43×)
	FedRS C=5	178.8 (1.15×)	-	-
	FedRS C=10	204.1 (1.32×)	-	-
	FedRS C=20	403.4 (2.60×)	-	-
$\leq$**3**	**FedGAP** (Our)	**213.5**	**293.8**	**494.8**
	AdaFL	261.5 (1.23×)	385.8 (1.31×)	764.3 (1.54×)
	FedAvg C=5	-	-	-
	FedAvg C=10	349.4 (1.64×)	-	-
	FedAvg C=20	417.4 (1.96×)	587.6 (2.00×)	928.4 (1.88×)
	FedRS C=5	-	-	-
	FedRS C=10	-	-	-
	FedRS C=20	-	-	-

The lowest energy costs achieved are highlighted in bold. A dash (-) is used to indicate that the target accuracy has not been met.

under higher skew (CIFAR-10 with $L \leq 3$) and on the more challenging CIFAR-100 task. Similarly, FedAvg with $C = 10$ fails to exceed 80% accuracy on CIFAR-10 with $L \leq 3$, falls below 46% on CIFAR-100 with $L \leq 5$, and never surpasses 44% accuracy on CIFAR-100 with $L \leq 3$.

FedRS demonstrates competitive energy efficiency at lower accuracy thresholds but struggles to attain higher accuracy levels. For instance, on CIFAR-100 with $L \leq 5$, FedRS with $C = 5$ achieves 42.0% accuracy at an energy cost of 178.8, substantially lower than FedAvg with $C = 10$ (261.7) and $C = 20$ (340.2). However, despite the lower cost at this level, FedRS fails to reach higher accuracy thresholds, even when larger concurrency values are employed. By contrast, FedAvg with larger C achieves higher accuracy, e.g., $A_{\text{th}} > 44.0\%$ with $C = 10$ and $A_{\text{th}} > 46.0\%$ with $C = 20$. Increasing C for FedRS, however, only raises the energy cost without meaningful improvements in maximum attainable accuracy. The situation worsens under more severe label skew ($L \leq 3$), where FedRS fails to meet any accuracy target on both CIFAR-10 and CIFAR-100. We attribute this behavior to the modified loss

function of FedRS, which accelerates convergence in the early training stages but slows progress in later phases, ultimately limiting overall effectiveness.

AdaFL, on the other hand, achieves more favorable accuracy-energy trade-offs compared to both FedAvg and FedRS. Its dynamic concurrency scaling enables it to consistently meet all predefined accuracy thresholds while requiring less energy than the static baselines. A single exception is observed on CIFAR-100 with $L \leq 5$ and accuracy target $A_{\text{th}} > 42.0\%$, where AdaFL incurs a cost of 217.4, whereas FedRS achieves the same accuracy at a lower cost of 178.8.

While AdaFL demonstrates competitive performance, FedGAP systematically achieves greater energy efficiency without compromising accuracy, owing to its gradient-aware participation policy. On CIFAR-10, FedAvg, FedRS, and AdaFL incur energy overheads of up to $2.74\times$, $2.21\times$, and $1.42\times$, respectively, compared to FedGAP. A similar trend holds for CIFAR-100, where the corresponding overheads rise to $2.43\times$, $2.60\times$, and $1.54\times$.

An illustrative example is provided by the CIFAR-100 benchmark under $L \leq 5$, where FedGAP achieves higher accuracy (44.0% vs. 42.0%) than AdaFL at a lower energy cost (211.2 vs. 217.4).

These results highlight the effectiveness of FedGAP's gradient-aware participation scaling strategy. Unlike AdaFL, which expands the concurrency at predetermined intervals regardless of training dynamics, FedGAP tailors the adjustment to the actual evolution of the learning process, increasing the number of active participants only when necessary. Taken together, the experimental evidence underscores the superior adaptability and robustness of FedGAP in balancing energy efficiency and model performance across different datasets and degrees of label skew.

4.3.3 FedGAP Sensitivity Analysis

In the previous evaluations, the hyperparameters of FedGAP were fixed to $\lambda = 100$ and $\varepsilon = 5\text{e}{-4}$. This subsection discusses the sensitivity of the method to these parameters. A grid-search analysis was carried out with $\lambda \in \{100, 200\}$ and $\varepsilon \in \{5\text{e}{-4}, 1\text{e}{-4}\}$, considering two representative benchmarks: CIFAR-10 and CIFAR-100 under the condition $L \leq 3$. The results, summarized in Tables 4.5 and 4.6, confirm the robustness of FedGAP, as all target accuracy levels were consistently achieved across the different settings.

Table 4.5 Sensitivity analysis on FedGAP hyper-parameters.
Results on CIFAR-10 with $L \leq 3$.

λ	ε	$A_{\text{th}} > \mathbf{79.0}$	$A_{\text{th}} > \mathbf{79.5}$	$A_{\text{th}} > \mathbf{80.0}$
100	**5**e−4	240.5	281.2	351.8
	1e−4	229.0	251.0	383.1
200	**5**e−4	197.5	208.3	242.4
	1e−4	215.8	252.1	273.6

Table 4.6 Sensitivity analysis on FedGAP hyper-parameters.
Results on CIFAR-100 with $L \leq 3$.

λ	ε	$A_{\text{th}} > \mathbf{42.0}$	$A_{\text{th}} > \mathbf{44.0}$	$A_{\text{th}} > \mathbf{46.0}$
100	**5**e−4	213.5	293.8	494.8
	1e−4	197.8	297.1	494.9
200	**5**e−4	183.5	240.2	365.6
	1e−4	191.5	233.5	319.6

The exploration further reveals that hyperparameter tuning can unlock additional energy savings. Among the two parameters, ε has only a marginal effect on performance, whereas λ plays a decisive role. For example, on CIFAR-10 with target $A_{\text{th}} > 80.0$, increasing λ from 100 to 200 yields a 1.45× reduction in energy cost (351.8 vs. 242.4, with $\varepsilon = 5\text{e}{-4}$). A similar trend is observed on CIFAR-100, where for $A_{\text{th}} > 46.0$, the same adjustment improves efficiency by 1.35× (494.8 vs. 365.6).

In conclusion, the sensitivity analysis highlights that FedGAP remains stable across different parameterizations, while larger values of λ are particularly advantageous when minimizing energy consumption is the primary objective.

4.4 Discussion and Final Remarks

This work introduced FedGAP, an energy-aware strategy for federated learning that enables the efficient training of classifiers across distributed end devices. FedGAP tackles one of the major shortcomings of conventional federated algorithms, namely, the sharp increase in client-side energy cost when local datasets contain only a limited subset of the target classes. By adaptively scaling the number of active

participants according to the evolution of the global model, FedGAP effectively reduces this overhead, achieving energy savings of up to 2.74×. The effectiveness of this approach is made possible through the *gradient alignment score*, a metric expressly designed to capture training dynamics in federated environments and guide adaptive participation policies.

Chapter 5

Securing Data and Models

To foster the widespread adoption of distributed intelligent services in IoT networks, it is essential to preserve privacy and intellectual property along the entire lifecycle of model training and deployment.

This chapter presents two complementary studies focusing on the model protection and its challenges. The first examines HE-based FL, a privacy-preserving approach that enables collaborative training while preventing the exposure of sensitive data and attributes, as well as unwanted access to the model weights with an untrusted server [119]. By allowing encrypted model updates to be securely aggregated, HE strengthens confidence in collaborative learning and mitigates risks of information leakage. Yet, its adoption is heavily undermined by the associated overhead affecting both computation and communication. This thesis investigates how to reduce such costs without incurring other penalties, mitigating the adoption barrier.

The second study focuses on the post-training deployment phase at the network edge, where models face new and often underestimated risks [120]. In particular, it reports the discovery of a previously undetected Side Channel Attack (SCA) , which exploits the Dynamic Voltage and Frequency Scaling (DVFS) through an ad-hoc load-testing method to infer sensitive model characteristics. Protecting models during inference service is crucial not only to maintain training data privacy, IP confidentiality, and service security and robustness, but also to safeguard the competitive advantage of model owners. Effective protection ensures that models trained, especially under costly privacy-protection schemes, can be monetized through serv-

ing these models at the edge, and thus, enabling recovery of the significant costs associated with training.

Together, these works highlight the need for end-to-end confidentiality (from federated training to edge inference deployment) to foster the widespread adoption of distributed services.

5.1 Tackling the Overhead of HE-based Privacy Preserving FL

5.1.1 Introduction and motivation

The practical advantage of FL in retaining data locally was originally deemed as a privacy-preserving strategy. Indeed, sharing model updates in place of raw data guarantees unwanted access and control over such data, protecting both users' and companies' interests as it provides compliance with the most recent international regulations [22]. However, the privacy brought by local training is not complete, and thus, while valuable, local training does not ensure the prevention of information leakage. Intelligent services and applications often rely on external servers and third-party services, which are deemed untrusted. Honest-but-curious servers (and malicious ones) pose significant threats to both privacy and intellectual property. Attacks such as gradient inversion [88, 89] demonstrated that sensitive user data can be extracted from model parameter updates. To mitigate these evolving threats, additional defensive mechanisms have become essential. Among them, HE has emerged as a promising strategy, as it directly prevents access to raw updates and model weights to untrusted parties, raising the privacy guarantees.

HE is a cryptographic paradigm allowing for a subset of arithmetic operations (addition and multiplication) to be run directly on encrypted data (from now on *ciphertext*) without the need for prior decryption. The results is still a ciphertext, thereby encrypted and secured by unwanted external access. The associated clear data (*plaintext*) can be recovered only by means of a dedicated private key, distributed to trusted users only. HE leaves the resulting output value unaltered, meaning that the output value, once decrypted, is the same as the operation would have been if computed directly onto the plaintext, or deviates within a bounded arithmetic error.

The HE paradigm favors FL scenarios as it enables clients to send local updates in the form of ciphertexts to the central server without compromising its aggregation functionality, as long as it envisages multiplications and additions of clients' updates, such as those encompassed by standard paradigms. The resulting upgraded version of the global model, with the parameters updated with the latest pseudo-gradients, is still a ciphertext whose plain values are not accessible from the server, thereby strengthening privacy and safeguarding IP. The clients participating in the next round directly download the latest ciphertext computed by the server and use a private key to decrypt it and retrieve the global model. Then, they proceed with the standard local training steps using a clear model and clear data. In this way, the loop can iterate until convergence without ever leaking the model parameters, safeguarding privacy by preventing adversarial and membership inference attacks on the server side.

HE for federated learning is gaining traction and prolific recent literature demonstrated its effectiveness for services where privacy is key, such as recommendation systems in consumer electronics [121] analysis of network traffic [122], and healthcare monitoring [123, 124]. This also led to industrial-grade implementation into famous FL frameworks of HE options, including FEDML [94], [93], FATE [91], PySyft [92], IBM Federated Learning [93], and NVIDIA Flare [90].

Although user-friendly development environments, tools, and libraries for HE are now available, several open challenges still hinder the large-scale adoption of HE-based FL, preventing services where stricter privacy is fundamental but resources are limited. The communicated payloads between the clients and the server grow significantly, often requiring 20-35$\times$ more data transmission compared to standard FL. Moreover, the overall training pipeline becomes considerably more complex and computationally demanding. On the client side, encoding and decoding operations introduce delays of up to 10$\times$ according to our experiments. On the server side, aggregation over encrypted updates can be up to 100$\times$ slower, caused by the additional operations required for aggregating encrypted tensors. These factors severely impact both computation and communication resources, which is particularly problematic in *cross-device* FL scenarios where clients operate under strict energy and communication constraints. Furthermore, many existing resource-optimization techniques are designed to accelerate FL are not compatible with HE, since they assume plaintext updates at the server [66, 68, 125, 126] to manage the training optimization.

While several recent works have sought to mitigate the overhead of HE-based FL through alternative encryption strategies [97–99, 91, 94], the complementary direction of optimizing the training process itself to make encryption stages lighter and more efficient remains relatively underexplored. This thesis advances that direction by introducing Private Tensor Freezing (PTF), a federated training protocol that applies a gradual tensor gating scheme fully compatible with HE, building from the work introduced in Chapter 3. PTF progressively freezes model tensors once they reach a stable representation, removing them from subsequent training and synchronization steps. In doing so, it alleviates cryptographic overhead as it reduces synchronization payloads and lowers the computational load on both clients and the server. This results in a more efficient use of communication and computing resources, while preserving both model accuracy and data privacy. The key technical challenge lies in identifying the appropriate point in the learning loop, at which tensors can be safely frozen without degrading accuracy while maintaining a method general and HE-compliant. Thus, PTF addresses this by extending ALF through a cooperative monitoring mechanism between clients and the server, enabling gradient-aware freezing decisions without exposing sensitive information.

Reducing the overhead of HE-based FL

As introduced in Chapter 2, multiple works investigated more efficient encryption schemes, providing more lightweight but also limited to specific stages [101] or fixed-point representations [91], while partial encryption schemes provide security-efficiency trade-offs hard to estimate a-priori [94]. This work employs the CKKS scheme via the open-source TenSEAL library [100], which natively supports batching. Nonetheless, as discussed later in Section 5.1.4, the computational and communication overheads remain significant. Our approach targets efficiency improvements on both client and server sides while preserving a floating-point representation, thereby retaining model accuracy. This work deliberately retains the complete security guarantees of the chosen HE scheme, though we acknowledge that relaxing security for efficiency remains a valid, orthogonal direction. Unlike prior works, the proposal follows a different path. Rather than optimizing the HE scheme itself or exploring privacy-efficiency trade-offs, this contribution focuses on limiting the data produced throughout the FL process, re-adapting ALF and the underlying assumptions in such an encrypted paradigm to assess the convergence status of different

tensors. The underlying goal is to make FL aware of HE's limitations and reduce both the amount of plaintext that must be encrypted and decrypted on the client side as well as the volume of ciphertexts transmitted and processed by the server. The early-learned tensors can be skipped in subsequent rounds, leading to substantial savings across all stages of FL, including encryption, decryption, communication, and aggregation. Importantly, our approach does not compete with existing HE optimizations but rather complements them, providing a practical enhancement to the overall efficiency of privacy-preserving FL, offering a practical optimization knob.

5.1.2 Private Tensor Freezing

Monitoring the Training Progress without access to pseudo-gradients

The proposed freezing strategy, Federated Learning with Private Tensor Freezing (PTF) , redesigns ALF to make it applicable within the HE settings by redefining tensor mobility when the server can no longer access plaintext pseudo-gradients. In fact, HE-based implementations prevent applying ALF, as it relies on server-side access to pseudo-gradients to detect when global and local update directions consistently oppose each other, which is key for making unanimous decisions about which layers can be considered stable and frozen to avoid further updates and synchronizations without incurring accuracy penalties. In an HE-based setup, this mechanism is no longer applicable, since the server only performs the evaluation of encrypted tensors and cannot inspect pseudo-gradients or raw parameters directly. This chapter presents a novel strategy that shifts mobility estimation and freezing decisions to the clients and approximates layer stability locally by tracking, over time, the trajectories of their updates with respect to successive global model snapshots. When these trajectories exhibit persistently small or opposing changes, the corresponding tensor is deemed close to convergence and is frozen, reducing both local computation and encrypted communication. This client-side reformulation preserves the core intuition of ALF while making it compatible with end-to-end encrypted aggregation.

Two main challenges arise when monitoring the tensor mobility in practical cross-device FL settings. The first concerns how to compute the mobility of a tensor within a global perspective. Since our freezing policy requires tensors to be frozen concurrently, such a decision must be taken unanimously among all the participating

clients. As a consequence, the clients must synchronize the information on the tensor mobility status without compromising the HE security. The second issue relates to clients who rejoin training after long periods of inactivity. Intuitively, their updates should contribute less to the mobility calculation compared to consistently active clients. To resolve the first challenge, this work advocates for centralizing the computation of tensor mobility, ensuring consistent freezing decisions.

For clarity, the formal tensor mobility description considers full client participation. As it is impractical in cross-device FL settings, its generalization to partial participation will be discussed later. After having downloaded the latest global model snapshot and decrypted its weights $\mathbf{w}_t^r$, a client can estimate the latest global update $\mathbf{g}_{t,s}^r$ for each tensor t as:

$$\mathbf{g}_{t,s}^r = \mathbf{w}_t^r - \mathbf{w}_{t,s}^{r-1}, \tag{5.1}$$

and its magnitude as:

$$\mathbf{h}_{t,s}^r = |\mathbf{w}_t^r - \mathbf{w}_{t,s}^{r-1}|. \tag{5.2}$$

To obtain tensor-level statistics, the client enforces the average of all elements in $\mathbf{g}_{t,s}^r$ and $\mathbf{h}_{t,s}^r$, obtaining $g_{t,s}^r$ and $h_{t,s}^r$, respectively. By considering their EWMA across consecutive rounds, the magnitude of their ratio $m_{t,s}^r$ is:

$$m_{t,s}^r = \frac{|\alpha \cdot g_{t,s}^{r-1} + (1-\alpha) \cdot g_{t,s}^r|}{\alpha \cdot h_{t,s}^{r-1} + (1-\alpha) \cdot h_{t,s}^r}. \tag{5.3}$$

To notice that the tensor mobility still ranges from 0 to 1, retaining the property of being close to one for aligned updates and zero for counteractive ones. This enables extending the tensor-gating scheme by automating the freezing mechanism, regulated by the threshold μ. Furthermore, $g_{t,s}^r$ and $h_{t,s}^r$ are two scalars that the clients can safely synchronize without introducing privacy risks.

Synchronizing such statistics with the server enables a private global agreement under partial participation of the tensor mobility. To this end, the participating clients upload the tuple $(g_{t,s}^r, h_{t,s}^r)$ at the end of each round, then, the server aggregates the received statistics as follows:

$$g_t^r = \sum_{s \in \mathscr{S}^r} g_{t,s}^r, \tag{5.4}$$

$$h_t^r = \sum_{s \in \mathscr{S}^r} h_{t,s}^r. \tag{5.5}$$

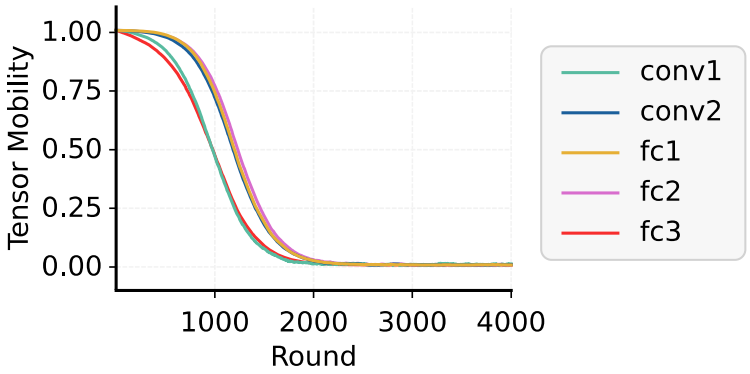


Fig. 5.1 Evolution of tensors' mobility for CNN-5 during training on CIFAR-10 with HE-based FedAvg (4000 rounds).

Then, it can compute the EWMA to monitor the tensor evolution within the latest rounds as:

$$m_t^r = \frac{|\alpha \cdot g_t^{r-1} + (1-\alpha) \cdot g_t^r|}{\alpha \cdot h_t^{r-1} + (1-\alpha) \cdot h_t^r}. \tag{5.6}$$

As the clients participate intermittently following stochastic eligibility patterns, they might have more updated or stale information, which the algorithm should take into account. Therefore, PTF enforces a participation-aware reweighting by applying derating factors to $\mathbf{g}_{t,s}^r$ and $\mathbf{h}_{t,s}^r$ proportional to the inactive time of the client. The difference between the current round r and the latest round $l_{t,s}$ during which a client s updated the tensor t can be used to calibrate the client statistics, weighting more fresh updates. Dividing the client statistics by the resulting difference provides:

$$\mathbf{g}_{t,s}^r = \frac{\mathbf{w}_t^r - \mathbf{w}_{t,s}^{l_{t,s}}}{r - l_{t,s}}, \tag{5.7}$$

$$\mathbf{h}_{t,s}^r = \frac{|\mathbf{w}_t^r - \mathbf{w}_{t,s}^{l_{t,s}}|}{r - l_{t,s}}. \tag{5.8}$$

To notice that, as the server computes the ratio between such statistics at each round, the reweighting factor influences the contribution of a specific client but not the scale of the tensor mobility threshold. This avoids the tensor mobility changes under different participation ratios, thereby remaining robust for different system configurations and scales.

Figure 5.1 illustrates the evolution of tensor mobility for a 5-layer CNN on CIFAR-10 across 4000 training rounds of standard HE-based FedAvg setup (see Section 5.1.3 for details). We tracked the mobility of five tensors corresponding to the weights of the convolutional layers (*conv1* and *conv2*) and the fully connected

layers (*fc1*, *fc2*, *fc3*). As expected, tensor mobility starts at one and progressively declines toward zero as training proceeds, approximating the layer mobility of ALF (Alg. 4).

FL with Private Tensor Freezing

In conventional FL, the set of trained tensors ($\mathscr{T}$) remains fixed across rounds ($\mathscr{T}^1$). Under PTF, however, this set is expected to shrink over time, as tensors with low mobility are progressively excluded from training based on the learning dynamics. Algorithm 6 presents the FL workflow incorporating PTF, with the blue-highlighted lines marking the differences from the baseline HE-based FL procedure shown in Algorithm 3.

During the initialization stage, the server assigns the value 1 to g_t^0, h_t^0, and l_t(line 6), while each client c initializes is local time-related variable $l_{t,s}$ to 0 (lines 8–10). During each round, the selected clients match the timing of local and global tensor updates. If these do not match for a client s, it downloads and decrypts the latest global model snapshot from the server (lines 15–17). If the local model does not correspond with the random initialization (i.e., $l_{t,s} > 0$), $g_{t,s}^r$ and $h_{t,s}^r$ are computed for every tensor in $\mathscr{T}^r$ (lines 18–22). Then, the local training begins (line 23) followed by the upload stage (lines 24–30), where clients synchronize both the updated variables ($g_{t,s}^r$, $h_{t,s}^r$) and encrypted tensors $\hat{\boldsymbol{\delta}}_s^r$. The clients also update the variable $l_{t,s}$ (line 29) and store a local version of the updated tensors (line 30). The server awaits all the ciphertexts to enforce the aggregation and evaluation stages (lines 31–33). The server computes g_t^r and h_t^r and updates the tensor mobility m_t^r (lines 35–38) with the EWMA (with α=0.995 in our setting). Finally, the freezing logic (lines 39–41) compares the mobility of each updated tensor with the threshold μ: if the tensor mobility $\mathbf{m}_t^r$ falls below the threshold (line 40), tensor t is removed from training and excluded from $\mathscr{T}^{r+1}$ (line 41).

5.1.3 Experimental Setup

The PTF performances were evaluated under the following experimental setup:

Datasets. We employed the CIFAR-10 and CIFAR-100 datasets as benchmarks, following a standard split for training and testing data. The training set was parti-

Algorithm 6: Workflow of HE-based FL with Private Tensor Freezing.

`/* Trusted Key Authority */`
Initialize sk and pk
Distribute sk and pk to the clients
Distribute pk to the server
`/* Server initialization */`
for $t \in \mathscr{T}^1$ **do**
  $\mathbf{w}_t^1 \leftarrow$ Random initialization
  $g_t^0 \leftarrow 1, h_t^0 \leftarrow 1, l_t \leftarrow 1$
`/* Client initialization */`
for $c \in \mathscr{S}$ **do in parallel**
  for $t \in \mathscr{T}^1$ **do**
    $l_{t,s} \leftarrow 0$
`/* Training Rounds */`
for $r = 1,...,R$ **do**
  Sample a subset $\mathscr{S}^r$ of clients
  `/* Clients Optimization */`
  for $c \in \mathscr{S}^r$ **do in parallel**
    for $t \in \mathscr{T}^1$ **do**
      if $l_t - l_{t,s} > 0$ **then**
        $\hat{\mathbf{w}}_{t,s}^{r,1} \leftarrow \hat{\mathbf{w}}_t^r$
        $\mathbf{w}_{t,s}^{r,1} \leftarrow \textsc{Decrypt}(\hat{\mathbf{w}}_{t,s}^{r,1}, sk)$
        if $t \in \mathscr{T}^r$ *and* $l_{t,s} > 0$ **then**
          $\mathbf{g}_{t,s}^r \leftarrow \frac{\mathbf{w}_t^r - \mathbf{w}_{t,s}^{l_{t,s}}}{r - l_{t,s}}$
          $\mathbf{h}_{t,s}^r \leftarrow \frac{|\mathbf{w}_t^r - \mathbf{w}_{t,s}^{l_{t,s}}|}{r - l_{t,s}}$
          $g_{t,s}^r \leftarrow$ Average all the elements of $\mathbf{g}_{t,s}^r$
          $h_{t,s}^r \leftarrow$ Average all the elements of $\mathbf{h}_{t,s}^r$
    $\mathbf{w}_{t,s}^{r,S} \leftarrow \textsc{Train}(\mathbf{w}_{t,s}^{r,1}, \mathscr{D}_c, S), \forall t \in \mathscr{T}^r$
    for $t \in \mathscr{T}^r$ **do**
      $\boldsymbol{\delta}_c^r \leftarrow \frac{1}{|\mathscr{S}^r|}(\mathbf{w}_{t,s}^{r,S} - \mathbf{w}_t^r)$
      $\hat{\boldsymbol{\delta}}^r \leftarrow \textsc{Encrypt}(\boldsymbol{\delta}_c^r, pk)$
      Upload $\hat{\boldsymbol{\delta}}_c^r$ to the server
      Upload $g_{t,s}^r$ and $h_{t,s}^r$ to the server
      $l_{t,s} \leftarrow r$
      $\mathbf{w}_{t,s}^{l_{t,s}} \leftarrow \mathbf{w}_{t,s}^{r,S}$
  `/* Server Optimization */`
  for $t \in \mathscr{T}^r$ **do**
    $\hat{\boldsymbol{\delta}}^r \leftarrow \textsc{Eval}(\sum_{c \in C^r} \hat{\boldsymbol{\delta}}_c^r, pk)$
    $\hat{\mathbf{w}}^{r+1} \leftarrow \textsc{Eval}(\hat{\mathbf{w}}^r + \hat{\boldsymbol{\delta}}^r, pk)$
    $l_t \leftarrow r$
    `/* Tensor Mobility Evaluation */`
    $g_t^r \leftarrow \sum_{c \in \mathscr{S}^r} g_{t,s}^r$
    $h_t^r \leftarrow \sum_{c \in \mathscr{S}^r} h_{t,s}^r$
    $m_t^r \leftarrow \frac{|\alpha \cdot g_t^{r-1} + (1-\alpha) \cdot g_t^r|}{\alpha \cdot h_t^{r-1} + (1-\alpha) \cdot h_t^r}$
    `/* Freezing Logic */`
    if $m_t^r < \mu$ **then**
      $\mathscr{T}^{r+1} \leftarrow \mathscr{T}^r \setminus \{t\}$

tioned among 100 clients through a Dirichlet partitioning method with a concentra-

tion parameter of 0.3, as in [127]. This creates a realistic data distribution providing statistical heterogeneity in terms of both size and label.

Models. The analysis conducted considered two different neural architectures, a simpler CNN-5 and a more resource-intensive but less parameterized ResNet-20 [128]. We expanded the scope of our assessment by using two models with inherently different neural architectures: CNN-5 [21] and ResNet-20 [128]. As a common practice for FL evaluation, batch normalization layers were replaced with group normalization [36].

Training. Each experiment encompassed 4000 FL rounds, using 10 concurrent clients per round (participation rate of 10%). Each participant conducted 20 local training iterations with SGD as the optimizer, with weight decay of $1e{-}3$, momentum 0.9, and batch size of 50. The local learning rate eta_l was fine-tuned in the set $\{0.01, 0.05, 0.1\}$ to provide a more comprehensive assessment, selecting the best performing one under the standard training without PTF. Then, the best performing learning rate (the one achieving the highest accuracy on the test set) was used for PTF as well.

Metrics. The prediction quality metric adopted is the global model's Top-1 classification accuracy measured on the test set. For a more stable assessment, a window average of 100 rounds was used for probing the accuracy over time and smoothing the oscillations, as suggested in [36]. The cost metrics relate to the communication and computation efficiency, using the total Payload (GB) consumed by each client (on average) for both download and upload, and the execution time (min) measured on the client side (accounting for decryption, training, and encryption) and separately on the server side (time taken to evaluate local updates and create a new global model snapshot).

Computing and HE Environment. The experimental campaign used a custom FL framework based on PyTorch [129] (version 2.1.0). Each experiment ran on a workstation equipped with a NVIDIA Quadro RTX 6000 GPU, Intel i9-10940X CPU, and 188 GB of RAM, hosting Ubuntu Server 20.04 as the operating system. The HE process used the TenSEAL library (v. 0.3.14), using the CKKS scheme, with 8192 polynomial modulus degree, 40 bits scale, and 200 bits for the coefficient modulus.

Table 5.1 Accuracy (Top-1), communication costs (Payload), and computational costs (client time T_{client} and server time T_{server}) of FedAvg without and with Homomorphic Encryption (HE-FedAvg). Results on CIFAR-10 and CIFAR-100.

Dataset	Model	Method	Top-1 (%)	Payload (GB)	T_{client} (min)	T_{server} (min)
CIFAR-10	CNN-5	FedAvg	78.32	2.4	0.5	0.1
		HE-FedAvg	78.53 (+0.21)	50.1 (20.9×)	7.4 (14.8×)	12.9 (129.0×)
	ResNet-20	FedAvg	81.86	0.8	2.1	0.3
		HE-FedAvg	81.75 (-0.11)	28.4 (35.5×)	5.9 (2.8×)	6.2 (20.7×)
CIFAR-100	CNN-5	FedAvg	45.06	2.4	0.6	0.1
		HE-FedAvg	44.98 (-0.08)	51.1 (21.3×)	7.6 (12.7×)	12.9 (129.0×)
	ResNet-20	FedAvg	52.84	0.8	2.1	0.3
		HE-FedAvg	52.37 (-0.48)	28.6 (35.8×)	5.9 (2.8×)	6.2 (20.7×)

5.1.4 Results

The experimental analysis first quantifies the communication and computation overhead, and assesses accuracy differences, by comparing CKKS-based HE-FedAvg against the standard FedAvg workflow. It then investigates how the proposed PTF mechanism can (*i*) alleviate these costs while preserving accuracy, (*ii*) adapt its behavior to the current training status, and (*iii*) enable additional overhead reductions without drastically affecting accuracy.

Overhead of Homomorphic Encryption on FL

The comparative analysis reported in Table 5.1 details the prohibitive costs introduced by HE when applied to FedAvg (HE-FedAvg vs FedAvg), reporting the previously mentioned metrics on CIFAR-10 and CIFAR-100.

In terms of accuracy (*Top-1*), both frameworks deliver comparable performance, with only minor fluctuations attributable to the approximation mechanisms of the CKKS scheme. These fluctuations do not suggest being systematic, sometimes leading to slight improvements (e.g., +0.21% for CNN-5 on CIFAR-10) and other times to marginal reductions (e.g., -0.48% for ResNet-20 on CIFAR-100). Nevertheless, the floating-point arithmetic supported by CKKS ensures that accuracy levels remain consistent with those attainable using plaintext updates. In contrast, the cost metrics reveal a substantial burden when HE is employed. For CIFAR-10, the communication overhead (*Payload*) increases by a factor of 20.9× for CNN-5 and 35.5× for ResNet-20, while the computational demands rise significantly on both

Table 5.2 Accuracy (Top-1), communication costs (Payload), and computational costs (client time T_{client} and server time T_{server}) of FedAvg without and with Homomorphic Encryption (HE-FedAvg). Results on CIFAR-10 and CIFAR-100.

Dataset	Model	Method	Top-1 (%)	Payload (GB)	T_{client} (min)	T_{server} (min)
CIFAR-10	CNN-5	**HE-FedAvg**	78.53	50.1	7.4	12.9
		HE-FedAvg + PTF	80.03 (+1.50)	31.9 (-36.3%)	4.9 (-33.8%)	8.2 (-36.4%)
	ResNet-20	**HE-FedAvg**	81.75	28.4	5.9	6.2
		HE-FedAvg + PTF	82.02 (+0.27)	20.8 (-26.7%)	4.7 (-20.3%)	5.6 (-10.7%)
CIFAR-100	CNN-5	**HE-FedAvg**	44.98	51.1	7.6	12.9
		HE-FedAvg + PTF	47.14 (+2.16)	32.0 (-37.4%)	4.9 (-35.5%)	8.2 (-36.4%)
	ResNet-20	**HE-FedAvg**	52.37	28.6	5.9	6.2
		HE-FedAvg + PTF	52.59 (+0.22)	19.6 (-31.5%)	4.6 (-22.0%)	5.3 (-14.5%)

the client (T_{client}) and server (T_{server}) sides. Client-side encryption and decryption dominate these costs, yielding worst-case slowdowns of 14.8× for CNN-5 and 2.8× for ResNet-20. In absolute terms, cryptographic operations are more time-intensive than the local training itself. On the server side, the aggregation time, which ordinarily completes within seconds, grows to several minutes. More importantly, this evaluation considered only 10 model updates per round, while under larger client participation, the server-side workload would rise dramatically, and thus, the incurred costs would impose a severe constraint on the scalability of the service.

Analogous trends are observed for CIFAR-100, with the additional effect of increased payload volume due to the larger label space (100 classes compared to 10), resulting in several extra GB of communication overhead. This analysis highlights the pressing need for methods that can reduce costs across both communication and computation, and for both clients and the server. The following subsection demonstrates how the proposed PTF approach effectively addresses this challenge.

PTF Evaluation

Table 5.2 reports the performance of the proposed PTF mechanism when integrated into the HE-based FL framework (HE-FedAvg + PTF) on CIFAR-10 and CIFAR-100. For comparison, the table also includes the baseline HE-FedAvg results without PTF. This initial evaluation was conducted with a mobility threshold of $\mu = 1e{-}3$; the influence of different μ values will be examined in subsequent experiments. Importantly, incorporating PTF does not compromise model accuracy; on the contrary, it yields consistent improvements. The observed gains in *Top-1* accuracy range

Table 5.3 Number of rounds (# Rounds) required to achieve a pre-determined target accuracy threshold (Top-1) in HE-FedAvg and HE-FedAvg + PTF. In addition, the number of frozen tensors (# TF) at the corresponding round is reported.

	HE-FedAvg	**HE-FedAvg + PTF**	
Top-1	**# Rounds**	**# Rounds**	**# TF**
79.5	2539	2316	14
80.0	2935	2359	16
80.5	3086	2407	16
81.5	3811	2539	19
82.0	-	2671	19

from +0.22% (ResNet-20 on CIFAR-100) up to +2.16% (CNN-5 on CIFAR-100). These results indicate that freezing stable tensors benefits the training process, effectively accelerating convergence as it removes unwanted updates that inject noise into converged tensors and naturally lead the optimizer to focus on focusing onto those parameters truly in need of refinement. Beyond accuracy, PTF introduces notable efficiency gains. Communication overhead is reduced by 26.7% to 37.4%, while computation time decreases significantly for both clients (20.3%-35.5%) and the server (10.7%-36.4%). These reductions stem from the careful design of the freezing protocol, which achieves substantial savings in both communication and computation while preserving compatibility with HE schemes. The key enabler of these savings is the tensor mobility, which accurately approximates the tensors' dynamics and reliably captures the convergence behavior of tensors to guide the freezing process.

To further demonstrate the ability of PTF to identify when tensors have effectively converged, the experimental analysis compared the number of training rounds required to reach incremental accuracy levels under HE-FedAvg and HE-FedAvg + PTF, further detailing the number of frozen tensors when reaching the target accuracy milestone. The results, summarized in Table 5.3 for CIFAR-10 with the ResNet-20 model, are representative of the trends observed across other benchmarks as well. Two key observations emerge from this analysis. First, PTF begins freezing tensors well before the model reaches peak accuracy. For example, at an accuracy of 79.5%, 14 tensors had already been frozen. Then, as training progressed, the number of frozen tensors increased to 19, while accuracy continued to rise by an additional 2.5%, eventually stabilizing at 82.0%. This behavior highlights that freezing is

Table 5.4 Sensitivity analysis on the mobility threshold μ. Results for CNN-5 trained on CIFAR-10.

μ	**Top-1** (%)	**Payload** (GB)	$\mathbf{T_{client}}$ (min)	$\mathbf{T_{server}}$ (min)
HE-FL	78.53	50.1	7.4	12.9
1e−1	79.52 (+0.99)	21.2 (-57.7%)	3.4 (-54.1%)	5.3 (-58.9%)
1e−2	79.92 (+1.39)	26.8 (-46.5%)	4.2 (-43.2%)	6.8 (-47.3%)
1e−3	80.03 (+1.50)	31.9 (-36.3%)	4.9 (-33.8%)	8.2 (-36.4%)

Table 5.5 Sensitivity analysis on the mobility threshold μ. Results for CNN-5 trained on CIFAR-100.

μ	**Top-1** (%)	**Payload** (GB)	$\mathbf{T_{client}}$ (min)	$\mathbf{T_{server}}$ (min)
HE-FL	44.98	51.1	7.6	12.9
1e−1	47.15 (+2.17)	20.1 (-60.7%)	3.3 (-56.6%)	5.0 (-61.2%)
1e−2	46.56 (+1.58)	26.1 (-48.9%)	4.1 (-46.1%)	6.6 (-48.8%)
1e−3	47.14 (+2.16)	32.0 (-37.4%)	4.9 (-35.5%)	8.2 (-36.4%)

initiated before the global model achieves its maximum accuracy, providing evidence that different tensors stabilize at different stages of training. Second, PTF accelerates convergence to given accuracy thresholds when compared with HE-FedAvg. As an illustrative case, HE-FedAvg required 3811 rounds to reach 81.5% accuracy, whereas PTF achieved the same performance within 2539 rounds, reducing the training effort by 1272 rounds. This finding suggests that PTF could enable further optimizations, for instance, by introducing an early stopping policy that terminates training depending on the reached model utility, thereby saving additional computational resources.

As a final consideration, Tables 5.4 and 5.5 present the sensitivity analysis of the mobility threshold μ, showing the resulting model accuracy and the associated communication and computation costs for three configurations, with μ spanning $\{1e{-}3, 1e{-}2, 1e{-}1\}$. Overall, *Top-1* accuracy tends to decrease as μ increases, with the notable exception of CIFAR-100, where the highest accuracy was obtained with the largest threshold (see the last row of Table 5.5). Importantly, several μ values still achieve better accuracy than the baseline implementation without PTF (row HE-FL in both tables).

These findings highlight an interesting trade-off between accuracy and cost. Larger thresholds trigger earlier freezing of tensors, leading to more aggressive reductions in resource consumption. In particular, for both datasets, setting $\mu = 1e{-}1$ halved the communication and computation costs, thereby reinforcing the effectiveness of PTF in optimizing resource utilization while maintaining competitive accuracy levels.

5.1.5 Discussion and Final Remarks

This work presented PTF, a novel tensor freezing protocol designed to mitigate the prohibitive overhead of incorporating homomorphic encryption into federated learning. While HE offers strong privacy guarantees, its computational and communication costs remain a major obstacle to practical deployment, especially in resource-constrained scenarios. PTF addresses this challenge by detecting tensors that have already converged and excluding them from subsequent training rounds. This approach reduces the overall training burden without compromising predictive performance. The protocol relies on a lightweight, HE-compliant monitoring mechanism based on tensor mobility, ensuring compatibility with encrypted learning pipelines. Experimental results demonstrate substantial efficiency gains: reductions of up to 37.4% in communication volumes and up to 35.5% and 36.4% in computation time on the client and server sides, respectively. These results highlight PTF's ability to make HE-based FL significantly more resource-efficient, thereby enhancing its feasibility for real-world applications and services.

5.2 Side-Channel Attacks for Inference-as-a-Service

5.2.1 Introduction and Motivation

Inference-as-a-Service (IaaS) is a delivery and licensing paradigm that allows connected clients to access pre-trained deep neural networks while safeguarding the IP of the service provider. Although initially designed for cloud infrastructures, the IaaS model can also be deployed at the network edge, for example, on IoT gateways, to improve response latency, scalability, and geographical coverage [130]. Edge-based IaaS typically adopts a microservice-oriented architecture. Microser-

Table 5.6 Overview of black-box SCAs against IaaS.

Reference	Access	Target Device	Information	Side-Channel Signal
This work	Remote	ARM Cortex-A	Architecture	DVFS state
[132]†	Remote	NVIDIA GPU	Architecture	Power consumption
[133]‡	Remote	NVIDIA Tegra	Family	Resource utilization
[134]*	Remote	Intel x86	Architecture	Cache timing
[135]	Remote	Intel x86	Layers Type	Performance counters
[136]	Remote	Intel x86	# of Layers	Latency
[137]*	Remote	Intel x86	Hyperparams	Cache timing
[138]	Physical	ARM Cortex-M	Weights	EM emission
[139]	Physical	ARM Cortex-A	Architecture	Power consumption
[140]	Physical	NVIDIA GPU	Architecture	Memory and PCIe bus
[141]	Physical	NVIDIA GPU	Weights	PCIe bus

* Manipulate the execution flow. †Require integrated power sensors.
‡ Neural networks that vary in the number of layers and/or filters but belong to the same family, yet share the same topology and core organization.

vices rely on lightweight virtualization technologies that enable the construction of hardware-independent inference engines, favoring deployment across a wide range of computing platforms. General edge platforms are resource-constrained embedded devices equipped with low-power CPUs such as ARM Cortex-A cores. Clients interact with the inference service through an Application Programming Interface (API) , which ensures functionality without disclosing direct access to the underlying neural network, thus protecting both model structure and weights. However, deploying microservices on distributed edge nodes increases their exposure to malicious activities [131]. In particular, SCAs pose a serious threat, as they can exploit physical leakages from the hosting device to recover sensitive information about the deployed neural network.

Previous research has investigated a wide spectrum of SCAs, targeting different hardware platforms and enabling the extraction of diverse types of information. Table 5.6 summarizes the state of the art, with a particular focus on black-box SCAs, i.e., techniques that operate without prior knowledge of the deployed deep neural network. The taxonomy distinguishes attacks along four dimensions: required level of access, targeted hardware device, nature of the information recovered, and type of side-channel leakage. *Physical attacks* rely on direct access to the device under test in order to capture physical signals, such as power traces [139], memory and

PCIe bus activity [140, 141], or electromagnetic (EM) emissions [138]. These high-resolution measurements can reveal detailed information about the neural network, ranging from its architecture to the reconstruction of model weights, thus allowing an adversary to replicate the network. Nevertheless, such approaches often demand intrusive probing and costly laboratory equipment, which restricts their feasibility in real-world deployments. *Remote attacks*, in contrast, are implemented purely in software by monitoring system metrics and performance counters that are accessible through operating system logs, typically without requiring elevated privileges. While more practical, these methods generally recover only partial knowledge of the target model. Examples include inferring subsets of hyperparameters (e.g., stride, pooling, padding in convolutional layers) [137], estimating the number of layers [136], identifying the types of layers but not the number of filters [135], or recognizing the family of the architecture [133]. Only a limited number of remote approaches can reconstruct the full neural architecture [134, 132]. Remote SCAs also suffer from intrinsic limitations that reduce their accuracy and robustness. For instance, cache-based attacks [134] require altering the execution flow through additional cache instructions, which may interfere with network execution. Similarly, power-based approaches leveraging software-accessible on-chip current sensors are constrained by low sampling rates and resolution [142]. Moreover, these sensors are typically available only in development kits and are often absent in production-grade hardware. As reported in [132], the detection accuracy of remote attacks remains relatively low, with a maximum Top-1 accuracy of 64.17%.

This work exposes a vulnerability in ARM Cortex-A CPUs that enables bypassing IaaS protections through a remote dictionary-based SCA. In particular, we investigate the use of DVFS traces as a novel leakage source for neural network fingerprinting. Prior research on DVFS-based SCAs has shown limited ability to extract meaningful signatures that could reveal the underlying architecture. To address this gap, we propose a load-testing procedure that issues carefully crafted sequences of queries to the prediction API according to predefined schemes. By regulating the service load, this procedure induces measurable variations in the device's voltage-frequency state. The resulting frequency traces are then analyzed with a machine learning classifier, trained offline, to infer the target neural architecture. This study validates the approach on the A72 and A57 embedded CPUs of the ARM Cortex-A family. Experimental results demonstrate that our approach achieves an average classification accuracy of 98.7% on the A72 and 92.4% on the A57 when distinguishing among 12

state-of-the-art CNNs optimized for embedded platforms. These results highlight that DVFS traces can be exploited as an effective side channel for neural architecture identification, advocating for the need to develop countermeasures against such attacks.

5.2.2 Background & Related Work

DVFS & Governors

DVFS is a runtime mechanism for managing power consumption and thermal dissipation in digital processors. Low-power ARM CPUs based on the Cortex-A architecture operate within a predefined set of voltage-frequency pairs, where each voltage level is mapped to a specific operating frequency. In Linux-based systems, DVFS is controlled through the CPUFREQ subsystem, which provides a software interface for adjusting CPU frequency states. CPUFREQ is composed of two key elements: the SCALING GOVERNORS and the SCALING DRIVER. The governor algorithms dynamically select the CPU frequency according to the observed system workload, while the driver is responsible for interfacing with the hardware to enforce the requested changes. The typical workflow of a governor involves continuous monitoring of CPU utilization, calculated as the ratio between the active (non-idle) CPU time and the total elapsed time since the last observation. The interval between two consecutive utilization measurements defines the sampling period, which, depending on the configuration setting, can vary between a few microseconds to several milliseconds.

The Linux kernel includes four standard scaling governors, namely CONSERVATIVE, ONDEMAND, SCHEDUTIL, and INTERACTIVE [143], whose availability and hyperparameters may depend on the hosting platform. The CONSERVATIVE governor adjusts the operating frequency gradually, following a hysteresis-based scheme. When CPU utilization falls below a bottom threshold (e.g., 20%), the frequency is decreased to a lower level (if available). Conversely, it increases to the next available level when utilization overcomes a predefined upper threshold (e.g., 80%). In contrast, the ONDEMAND, SCHEDUTIL, and INTERACTIVE governors implement workload-proportional scaling, where the operating frequency is selected to be approximately linear with the observed CPU utilization. The ONDEMAND governor proportionally maps 0% and 100% CPU utilization to the minimum and

maximum available frequencies, respectively. The SCHEDUTIL governor refines this strategy by applying an over-scaling factor of 1.25× to mitigate the risk of setting the frequency too low, which could degrade performance. Additionally, SCHEDUTIL instantly boosts the frequency to its maximum level in response to missed deadline events. The INTERACTIVE governor, as the name suggests, is designed to favor responsiveness, prioritizing rapid frequency up-scaling while penalizing down-scaling. Specifically, sudden increases in CPU utilization lead to immediate switching to the maximum frequency and delay frequency reductions to avoid oscillations that may compromise performance. As a result, the INTERACTIVE governor ensures reactivity at the cost of reduced energy savings. Users with root privileges can modify governor settings and parameters (e.g., the thresholds defining the switching policies), whereas generic users are restricted to reading the active governor policy and monitoring the current operating frequency.

DVFS Side-Channel Attacks

DVFS SCAs aim at stealing sensitive data or identifying the applications in use by gathering frequency traces from the victim device and leveraging ML models trained offline. For instance, [144] demonstrated that DVFS leakage can reveal users' browsing activities and even expose passwords entered on smartphones. In [145] and [146], DVFS SCAs were investigated in the context of application detection. Repeating the execution of the target application enables fingerprinting it through the analysis of the DVFS traces. Specifically, [145] validated the approach on 22 Android benchmarks running on an ARM Cortex-A CPU, while [146] provided a more exhaustive evaluation using two representative suites: SGX-BENCH and PARSEC 3.0. The findings confirmed that DVFS SCAs are effective for applications with long execution times and varying CPU usage (e.g., PARSEC 3.0), but fail in the case of short and compute-intensive workloads (e.g., SGX-bench), reducing to random guessing. As discussed in the next section, deep neural network inference falls into this latter category: individual inference requests correspond to short bursts of high CPU utilization (on the order of milliseconds), while continuous inference results in a sustained, near-maximal CPU load. This characteristic motivates the methodology proposed in this work.

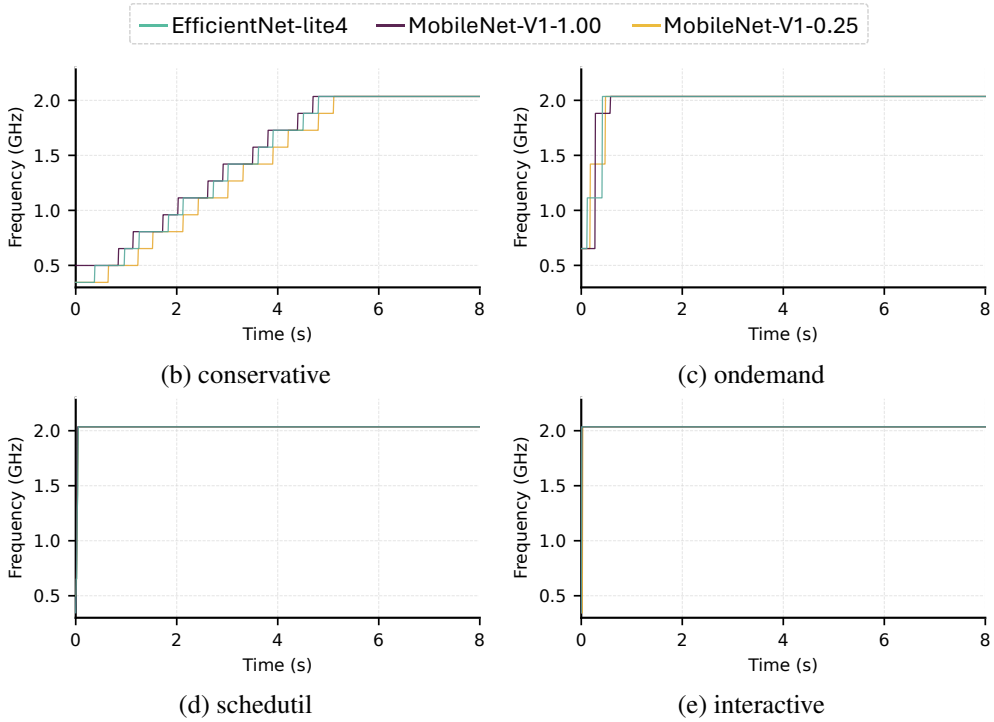


Fig. 5.2 Frequency traces examples collected on A57 using the DVFS SCAs standard procedure.

5.2.3 Methodology

Understanding the Limitations of DVFS SCAs

In conventional DVFS SCAs, the target application is repeatedly executed while monitoring how the CPU frequency evolves over time. The same principle can be applied to IaaS by issuing continuous queries to the prediction API. However, the deep neural networks used in embedded inference services are heavily optimized through algorithmic and code-level enhancements prioritizing inference latency [147]. As a result, their workloads are short and highly resource-demanding. This behavior forces DVFS governors to immediately scale the CPU to the highest available frequency and keep it there until inference ends, regardless of the underlying neural architecture. To illustrate this limitation, we deployed three networks with different sizes and algorithmic complexity (MobileNet-V1-0.25, MobileNet-V1-1.0, and EfficientNet-lite4) on an A57 CPU (see Sec. 5.2.3 for details). We then profiled frequency traces over 8 seconds of continuous queries under the four governors described in Sec. 5.2.2. As shown in the line plots of Fig. 5.2, the resulting traces

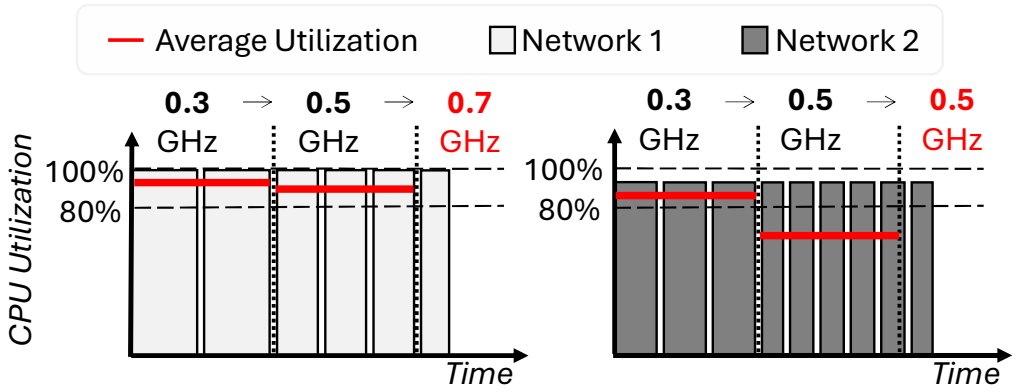


Fig. 5.3 Frequency evolution of the CPU during the execution of two architectures with the proposed strategy.

are nearly indistinguishable, with all governors quickly saturating at the maximum frequency level (2.0 GHz). These results highlight the fundamental limitation of existing DVFS SCAs: they fail to discriminate between neural architectures due to the uniformly high resource demand of embedded inference workloads. This limitation motivates the need for a new probing methodology, which we introduce in the next section.

Altering the Frequency Profiles

This work proposes a simple yet effective strategy to perturb the CPU frequency updates regulated by DVFS governors. Instead of issuing continuous API queries, we introduce controlled pause intervals between consecutive requests. Longer (shorter) idle periods reduce (increase) the observed CPU utilization, which in turn drives the governors towards lower (higher) operating frequencies. The number and duration of these pauses are inversely related to the service response time, a metric inherently influenced by the neural architecture used to serve the request and the current CPU frequency.

By carefully shaping the sequence of requests, an attacker can induce distinct frequency traces that evolve differently depending on the operating architecture. Over sufficiently long observation windows, these frequency profiles form discriminative signatures that can be exploited for neural architecture fingerprinting.

Fig. 5.3 illustrates how distinct frequency states can emerge from two neural architectures (Network 1 and Network 2, with Network 2 being less resource-demanding).

For simplicity, the example assumes a CONSERVATIVE governor, which increases the frequency whenever the average CPU utilization during an observation window exceeds 80%. In the plots, colored bars represent the processing stage of each inference request (light gray for Network 1, dark gray for Network 2). The bar width indicates CPU time required to process a request, dependent on both the neural architecture and CPU frequency, while the bar height reflects CPU utilization, also architecture-dependent. Both response time and utilization influence the governor's workload estimation within each observation window (vertical dotted lines). Initially, both networks run at the lowest frequency (0.3 GHz). Since average utilization exceeds the relative threshold of 80%, the governor up-scales the frequency to 0.5 GHz independently of the network. Concerning the second window, the inserted pause intervals (white spaces) reduce the observed utilization. For Network 1, the CPU Utilization only marginally decreases and remains above the threshold (80%), so the frequency rises to 0.7 GHz. For the less complex Network 2, the CPU utilization reduces significantly, and thus the frequency remains unchanged. Consequently, the frequency traces of the two networks become distinct, whereas without the pause intervals, the third-window frequency update would be identical for both networks. The pause duration acts as a hyperparameter that can be tuned to maximize differentiation across neural architectures and CPU models. Additional implementation details are provided in the following subsections.

Load Testing

Load testing is a performance evaluation method designed to assess system behavior under specific workloads or varying request volumes. A standard approach generates traffic via a pool of concurrent virtual users issuing requests at a predefined rate. Both the number of virtual users and the request rate can be tuned to simulate different scenarios and evaluate the service's stability and scalability.

This work adapts a typical load testing setup to implement the pausing mechanism described previously, using a single virtual-user configuration where the pause interval serves as the primary testing parameter. The pause interval is defined as the time between the service response and the subsequent client request, acting as a knob to modulate CPU utilization and induce frequency variations. Optimal pause values depend on factors such as the DVFS governor in use and the neural network's characteristics (latency and resource demand). To identify effective configurations,

we performed multiple load tests sweeping the pause length over a predefined set of values, as detailed in the following subsection.

Attack Overview

The proposed SCA consists of performing four sequential load tests, each lasting 10 s, with pause intervals of 5, 20, 50, and 100 ms.

The threat model is defined under the following assumptions. First, the attacker possesses a device (denoted as the *template device*) with specifications mirroring those of the victim device. In other words, the proposed attack falls in the category of PROFILED ATTACKS [148]. Second, the attacker can remotely access the victim device to install malware capable of monitoring and transmitting CPU frequency traces without requiring root privileges. Third, the target neural network belongs to a known pool of architectures, although the specific weights may differ. This assumption is realistic in practice, as transfer learning [149] commonly involves adopting publicly available neural architectures and fine-tuning them on proprietary datasets.

Following the standard structure of a profiled attack, our method comprises two main stages, referred to as the PROFILING phase and EXTRACTION phase.

During the PROFILING phase, frequency traces of the target neural architectures are collected to train a machine learning classifier capable of inferring the architecture from observed traces. To build this dataset, the load tests described in Sec. 5.2.3 are executed on the template device for each target architecture. A background process records the CPU frequency every 10 ms through the CPUFREQ interface. Each load test is repeated 50 times to account for small fluctuations caused by background operating system routines, and this procedure is performed for each of the four pause intervals. As a result, the dataset comprises multiple frequency traces, each annotated with the corresponding neural architecture label. The classifier employs a MiniRocket [150] feature extractor, widely used for time-series analysis, followed by a multinomial logistic regressor. Training is performed in a supervised fashion, by employing the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) solver and the stratified cross-validation (5-fold) to optimize the hyperparameters, while the training loop encompasses 200 iterations. The data collection and training procedure is repeated for all combinations of governors and template devices used in

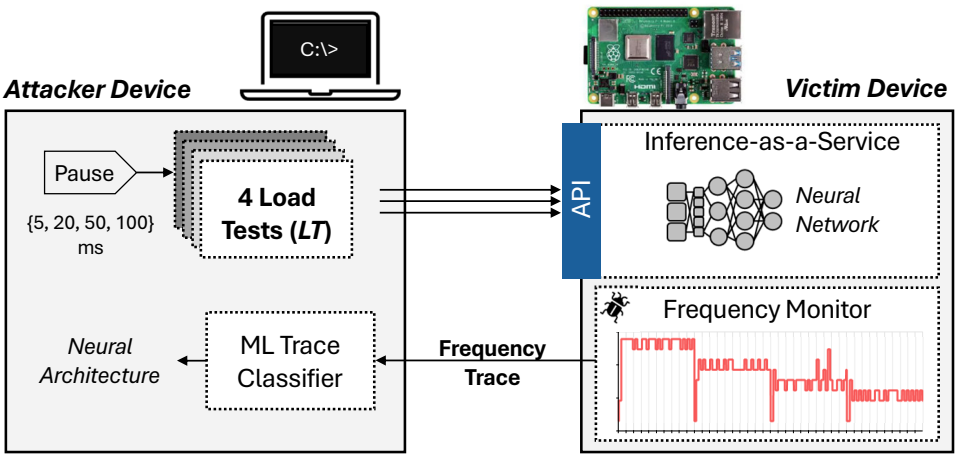


Fig. 5.4 Extraction phase overview.

Table 5.7 Hardware platforms, Operating System (OS), maximum (Max. Freq.) and minimum (Min. Freq.) frequency, and number of frequency levels of the target CPUs.

CPU	Platform	OS	Min. Freq.	Max. Freq.	Levels
A72	Raspberry Pi 4	Ubuntu 22.04	600 MHz	1.500 GHz	10
A57	NVIDIA TX2	Ubuntu 18.04	345 MHz	2.035 GHz	12

our experiments, yielding a dedicated ML classifier for each hardware and governor configuration.

As depicted in Fig. 5.4, during the EXTRACTION phase, the attack is executed to infer the target neural architecture. The standard adopted assumption is that the target (victim) device is one of the previously profiled template devices, but the neural architecture embedded into the inference service is unknown. The attacker deploys the four load tests to the targeted prediction endpoint while the pre-installed malware records the CPU frequency of the victim device. Once the load tests are completed, the collected frequency traces, along with the active governor information, are transmitted to the attacker. These data are then input into the ML classifier trained specifically for the corresponding governor. The classifier outputs the neural architecture used by the victim inference service.

Table 5.8 Governors and their sampling period (ms) on the target CPUs.

CPU	conservative	ondemand	schedutil	interactive
A72	10.0	10.0	10.0	N/A
A57	300.0	300.0	0.5	20.0

Table 5.9 Pool of neural network architectures.

Architecture	Layers	Size (MB)	Latency (ms)		Util. (%)	
			A72	A57	A72	A57
EfficientNet-lite4	92	14.34	119.60	89.04	96.8	96.9
EfficientNet-lite3	74	9.15	68.97	51.46	95.2	95.2
EfficientNet-lite2	65	6.83	44.96	33.37	93.6	93.6
EfficientNet-lite1	65	6.12	32.32	23.87	92.4	92.6
EfficientNet-lite0	50	5.18	21.62	16.48	90.7	90.3
MobileNet-V3-large	64	5.35	25.80	20.26	96.4	97.9
MobileNet-V3-small	54	2.52	10.20	8.06	92.5	95.6
MobileNet-V2	53	3.42	23.39	17.85	96.2	97.7
MobileNet-V1-1.0	28	4.09	27.86	21.03	96.8	98.0
MobileNet-V1-0.75	28	2.51	17.80	13.53	95.0	97.0
MobileNet-V1-0.50	28	1.31	9.68	7.44	91.9	95.1
MobileNet-V1-0.25	28	0.49	4.71	3.57	85.6	92.0

Experimental Setup

We used two commonly adopted commercial platforms as test benches: an NVIDIA Jetson TX2 and a Raspberry Pi 4. These platforms embed two different ARM Cortex-A CPUs, respectively the A72 and A57, whose technical specifications are summarized in Table 5.7. Although both CPUs have four cores, they differ in frequency range, number of frequency levels, and available DVFS governors. Table 5.8 lists the supported governors and their sampling periods (as defined by the CPU vendor). Notably, the sampling period can vary by an order of magnitude between the two CPU architectures, which influences the responsiveness of DVFS state updates.

Table 5.9 summarizes the CNNs used as benchmarks, all sourced from TensorFlow Hub [151] in TFLite format. The table presents average latency and CPU utilization values, measured over 100 consecutive inference runs executed at maximum frequency with 4-thread parallelism. The selected architectures have been

pre-trained on the ImageNet dataset and quantized to 8-bit, and belong to four distinct families: EfficientNet [152], MobileNet-V3 [153], MobileNet-V2 [154], and MobileNet-V1 [155]. The pool includes architectures with very similar characteristics. For instance, MobileNet-V1 variants share the same number of layers but have a different number of filters. Additionally, some architectures exhibit comparable latency and utilization, such as MobileNet-V1-0.50 and MobileNet-V3-small. These similarities pose challenges for detection accuracy, and our experiments aim to assess the effectiveness of our methodology in distinguishing between these closely related networks.

Results

We evaluate the performance of the attacker's classifier using standard multi-class classification metrics. Each processed sample could be a True Positive (TP), False Positive (FP), False Negative (FN), or True Negative (TN).

Our primary metric is the F1 score, defined as the harmonic mean of precision and recall.

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{Pr} \cdot \mathrm{Re}}{\mathrm{Pr} + \mathrm{Re}} \tag{5.9}$$

Precision (Pr) measures the fraction of correctly predicted positive instances (true positives) over all predicted positives (true positives plus false positives), while recall (Re) measures the fraction of correctly predicted positives over all actual positives (true positives plus false negatives).

$$\mathrm{Pr} = \frac{TP}{TP+FP}, \quad \mathrm{Re} = \frac{TP}{TP+FN}. \tag{5.10}$$

The results also report per-class F1 scores, which evaluate performance individually for each class by treating it as positive and the others as negative. The average of these per-class scores, denoted as Macro-F1, provides an overall measure of model performance across all classes. In addition, the results report the Top-1 Accuracy, defined as the ratio between the number of correct predictions and the total number of instances.

$$\text{Top-1 Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\big(\hat{y}_i^{(1)} = y_i\big), \tag{5.11}$$

and the Top-2 Accuracy, which considers a prediction correct if the true label appears among the two classes with the highest predicted probabilities. All metrics are reported on a test set equal in size to the training set, i.e., 600 traces (50 traces per network) for each CPU/governor configuration.

$$\text{Top-2 Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\big(y_i \in \{\hat{y}_i^{(1)}, \hat{y}_i^{(2)}\}\big) \tag{5.12}$$

The experimental results in Tables 5.10 and 5.11 demonstrate the efficacy of the proposed SCA. The Macro-F1 scores range from 97.5% to 99.7% on the A72 and from 83.6% to 99.2% on the A57, depending on the governor. These results confirm that the attacker's classifier can reliably identify the target network architecture, even across different CPUs and governor settings.

Prediction quality mainly depends on the target network and the governor sampling period. Most incorrect predictions are between architectures with similar characteristics. The Top-2 accuracy shows that even when Top-1 accuracy is low, the true class is usually among the top two predictions, as also highlighted by the last row of the table. For example, the A57 with the CONSERVATIVE governor represents the worst-case scenario, yielding a Top-1 accuracy (83.5%) but still achieving 97.3% Top-2 accuracy. Overall, Top-2 accuracy is consistently close to 100%, with misclassifications limited to specific combinations of governor and network.

Regarding the sampling period, larger values reduce the number of frequency switches, thereby limiting the information leakage. This is evident for the ONDEMAND and CONSERVATIVE governors, which achieve significantly lower scores on the A57 (sampling period 300 ms) compared to the A72 (10 ms). For instance, for the CONSERVATIVE governor, the Macro-F1 drops from 99.7% (A72) to 83.6% (A57). Conversely, the SCHEDUTIL governor achieves higher scores on the A57 (sampling period 0.5 ms) than on the A72 (10 ms). Intuitively, longer sampling periods underfit more fine-grained statistics, making it harder to profile the target network.

Fig. 5.5 provides visual evidence of the effectiveness of the proposed load testing methodology. The figure reports frequency traces collected for three networks deployed on the A57 CPU, namely MobileNet-V1-0.25, MobileNet-V1-1.0, and EfficientNet-lite4, under the CONSERVATIVE, ONDEMAND, SCHEDUTIL, and INTERACTIVE governors. These are the same networks shown in Fig. 5.2. Unlike standard DVFS SCAs, the traces are now clearly distinguishable, enabling a successful attack.

Table 5.10 Results on A72.

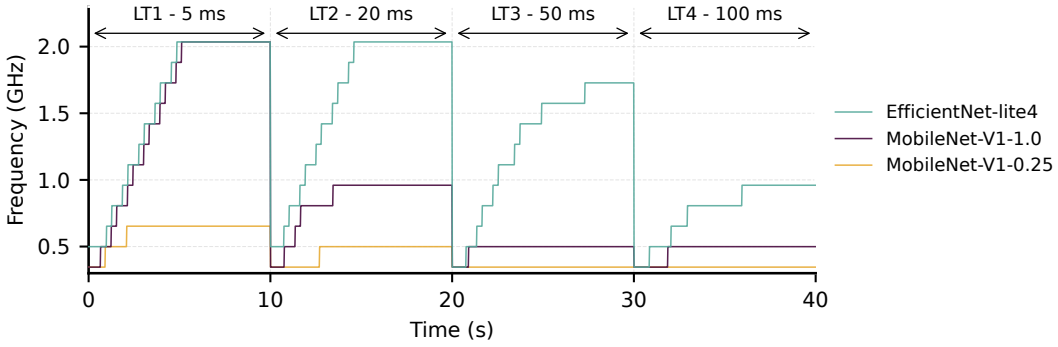
	Architecture	**conservative**	**ondemand**	**schedutil**
1	EfficientNet-lite4	100.0%	100.0%	100.0%
2	EfficientNet-lite3	100.0%	100.0%	100.0%
3	EfficientNet-lite2	100.0%	100.0%	100.0%
4	EfficientNet-lite1	99.0%	100.0%	100.0%
5	EfficientNet-lite0	100.0%	100.0%	100.0%
6	MobileNet-V3-large	100.0%	100.0%	100.0%
7	MobileNet-V3-small	100.0%	94.2%	86.5%
8	MobileNet-V2	100.0%	100.0%	100.0%
9	MobileNet-V1-1.0	99.0%	100.0%	100.0%
10	MobileNet-V1-0.75	99.0%	100.0%	100.0%
11	MobileNet-V1-0.50	99.0%	93.8%	84.5%
12	MobileNet-V1-0.25	100.0%	100.0%	99.0%
	Macro-F1	99.7%	99.0%	97.5%
	Top-1 Accuracy	99.7%	99.0%	97.5%
	Top-2 Accuracy	100.0%	100.0%	99.8%
	Misclassifications	-	11-7	11-7

Table 5.11 Results on A57.

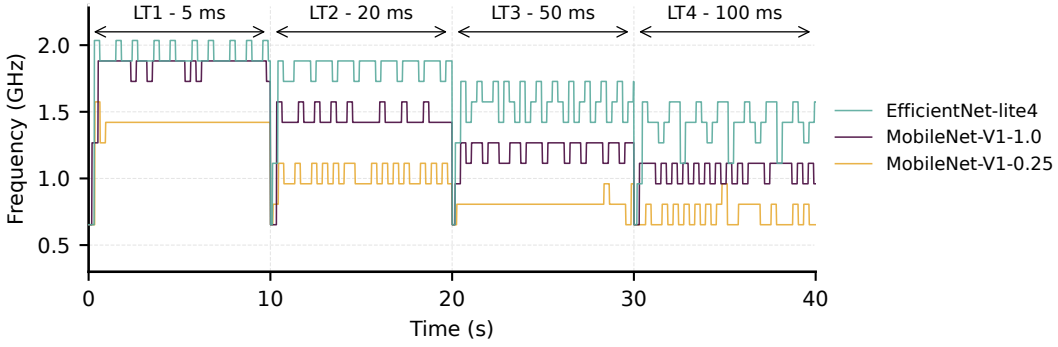
	Architecture	**conservative**	**ondemand**	**schedutil**	**interactive**
1	EfficientNet-lite4	100.0%	100.0%	100.0%	65.3%
2	EfficientNet-lite3	100.0%	100.0%	100.0%	65.4%
3	EfficientNet-lite2	95.9%	100.0%	100.0%	98.0%
4	EfficientNet-lite1	93.2%	100.0%	100.0%	98.0%
5	EfficientNet-lite0	67.9%	100.0%	100.0%	99.0%
6	MobileNet-V3-large	77.9%	72.9%	95.1%	94.0%
7	MobileNet-V3-small	63.4%	100.0%	100.0%	100.0%
8	MobileNet-V2	68.0%	99.0%	99.0%	99.0%
9	MobileNet-V1-1.0	80.8%	59.3%	95.9%	90.7%
10	MobileNet-V1-0.75	93.9%	100.0%	100.0%	99.0%
11	MobileNet-V1-0.50	62.6%	100.0%	100.0%	100.0%
12	MobileNet-V1-0.25	99.0%	100.0%	100.0%	100.0%
	Macro-F1	83.6%	94.3%	99.2%	92.4%
	Top-1 Accuracy	83.5%	94.5%	99.2%	92.3%
	Top-2 Accuracy	97.3%	99.7%	99.8%	99.8%
	Misclassifications	11-7, 8-5, 9-6	9-6	9-6	2-1

This improvement stems from the testing procedure, which highlights the behavior of different governors. With CONSERVATIVE, the frequency rises gradually and stabilizes at distinct levels. With ONDEMAND, it jumps to a workload-proportional value with small fluctuations caused by the system noise. With SCHEDUTIL, it provides sharp oscillations, showing sudden surges caused by its short sampling period (0.5 ms). Finally, the INTERACTIVE governor abruptly changes the operating frequency, but it is less inclined to reduce it compared to SCHEDUTIL.

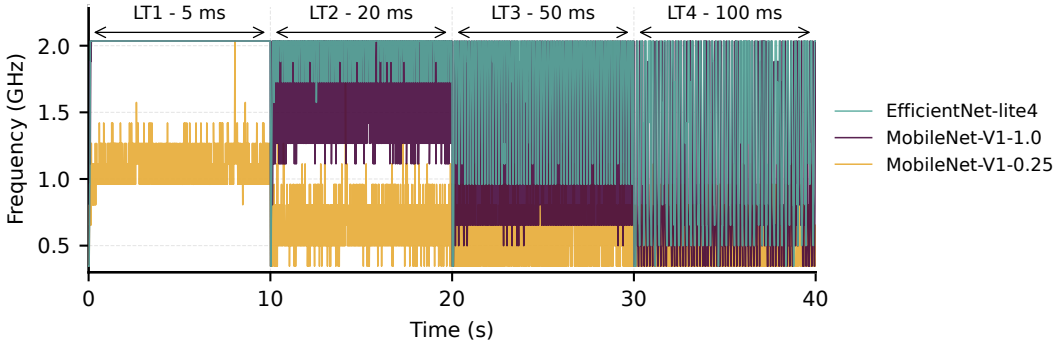
Moreover, the plots highlight the importance of running a set of load tests, each featuring a different pause interval. The traces collected with a pause of 5 ms for the CONSERVATIVE and SCHEDUTIL governors make it evident (Figs. 5.5a and 5.5c). In these cases, the traces of MobileNet-V1-1.0 and EfficientNet-lite4 overlap, whereas they become distinguishable when longer pause intervals are applied (20 ms, 50 ms, and 100 ms). In other words, only the combination of multiple load tests ensures that traces are distinguishable across all networks and governors.



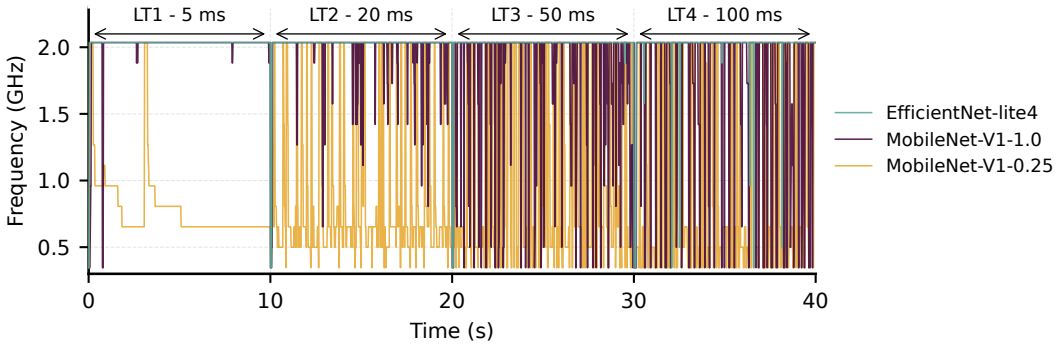
(a) conservative



(b) ondemand



(c) schedutil



(d) interactive

Fig. 5.5 Frequency traces examples collected on the A57 when using the proposed load testing methodology. The four load tests are delineated by dashed arrows, with the corresponding pause durations annotated above each arrow.

5.2.4 Discussion and Final Remarks

This work has demonstrated that DVFS-based SCAs can successfully perform network fingerprinting in edge inference services, circumventing IaaS protection mechanisms through purely software-based procedures. By leveraging a load-testing strategy, the generated frequency traces act as distinctive signatures, allowing the accurate identification of the victim's neural architecture. This section highlighted the urgency of developing effective countermeasures, as the identified leakage channel represents a concrete threat to the confidentiality of deployed models. Knowledge of the underlying neural architecture can empower malicious actors to mount more efficient adversarial attacks that manipulate predictions, or to execute white-box attacks aimed at stealing sensitive information or replicating the network functionality, posing severe risks for both security and privacy in distributed intelligence environments. This work underscores that strong security guarantees must also extend to the inference stage, not only to federated training. Neglecting inference-time protection risks nullifying the substantial overhead invested in training-time countermeasures and additionally suggests that analogous side-channel vectors could be exploited against FL and ppFL deployments as well.

Chapter 6

Federated Learning Evaluation

Successful deployment of federated learning services demands evaluation approaches that not only measure model performance but also account for a broader spectrum of factors influencing real-world outcomes. Comprehensive evaluations should characterize the model's capabilities across a range of extra-functional metrics, tailored to the intended application scenario. Without a thorough pre-deployment assessment, the effectiveness of novel algorithms may remain unclear, hindering accurate assessment of their true advantages. This highlights the importance of systematically exploring the hyper-parameter search space to rigorously assess quality, robustness, and efficiency before deployment.

Different Hyperparameter Optimization (HPO) frameworks [156–159] rapidly identify optimal pre-deployment configurations to find optimal pre-deployment configurations, all of which rely on a pre-defined hyperparameter grid to specify the search space. While significant emphasis has been placed on tuning the initial learning rates for clients and the server, the influence of learning rate decay schedules is often neglected. Given that learning rate configurations are closely intertwined with the training budget and overall convergence properties, omitting decay schedules from the search space may be insufficient to thoroughly assess the potential of FL strategies. Thus, this chapter investigates the role of two-sided learning rate schedules [160] and, more specifically, budget-driven scheduling functions in the context of cross-device FL, demonstrating the need for their inclusion in the search space for avoiding sub-optimal performances and misleading evaluations.

Effective pre-deployment evaluations in privacy-sensitive domains also require realistic data conditions. However, in privacy-sensitive domains, representative labelled datasets are often unavailable, which hinders preliminary assessments of feasibility and benefits and, in turn, limits and slows down practical deployments. Therefore, this chapter concludes the thesis with a final contribution that provides a realistic use-case study for evaluating the potential of privacy-preserving FL (based on Homomorphic Encryption) for fine-grained household sociodemographic classification from smart meter data. To this end, it introduces a dedicated benchmark that combines a domain-specific synthetic data generation pipeline with an HE-based FL evaluation framework [161]. The analysis reveals that HE-based FL can closely approach the performance of a centralized baseline while keeping data local and secured, thus offering a concrete, deployment-oriented assessment of ppFL in a real-world-inspired scenario.

6.1 Fair Evaluation of FL with Two-sided Learning Rate Tuning

6.1.1 Introduction and Motivation

As introduced throughout the thesis, in the context of FL, two persistent bottlenecks remain: training quality and training budget. The former is affected by structural constraints such as client data heterogeneity and limited client participation per round, both of which can hinder the accuracy and stability of the global model. The latter reflects the overall resource footprint on clients, server, and network, which is typically modelled with the number of training rounds required to achieve a target performance [38, 39]. An increase in training rounds directly translates into higher demands for computation, memory, and communication, resulting in substantial recurring costs.

Extensive research has sought to address these challenges [162–164, 35]. Nonetheless, determining the most effective FL strategy for a given setting and fairly comparing alternative implementations remain open issues. This complexity stems from the strong dependence of FL performance on multiple factors, including the the local training steps, the degree and type of data heterogeneity, and the number of partici-

pating clients per round [165, 39, 166]. As a result, no single algorithm consistently dominates across all scenarios, and the choice of method is highly context-dependent [165]. A major obstacle, therefore, is the absence of standardized evaluation practices that enable fair and reliable comparisons. Recent works have made progress by analyzing diverse data distributions and varying training parameters to approximate a broader spectrum of FL conditions [165, 39, 166], showing that poor experimental setups can easily lead to inconsistent conclusions.

Within this scenario, a crucial yet relatively overlooked aspect concerns the role of learning rate configurations. Modern FL implementations implement FL with two-sided learning rates [35, 163, 167] (as introduced in Chapter 2). The tuning process requires the simultaneous adjustment of two parameters: the client-side learning rate for local training and the server-side learning rate for updating the global model. This two-sided optimization strategy has been established as a standard practice for recent FL algorithms [168–171], as it can both speed up convergence [167] and improve the final model utility [38]. The adopted learning rate settings exhibit a profound impact on the conclusions extracted from the experimental campaigns, and insufficient tuning may obscure the effective potential of an optimization strategy.

The standard evaluation methodology seeks to tackle this challenge by enforcing a grid-search exploration over predefined sets of local and global learning rate values, aiming to determine the optimal configuration for each FL algorithm under a given setting [38, 163, 168–171]. The performance of the model under its best configuration is then used as the benchmark for comparison (against other fine-tuned strategies), with the underlying assumption that well-initializing the two learning rates suffices for achieving optimal performance. In this work, we challenge this assumption and show that the standard two-sided tuning procedure falls short of providing a reliable performance assessment. In particular, two critical aspects are commonly overlooked in FL research, which are the influence of learning rate schedules on both convergence speed and final model utility, and the importance of budget-aware tuning, for those cases where the budget can be identified a-priori or can be settled to meet specific criteria. Ignoring these factors risks underestimating the true potential of an algorithm, as it may prevent the identification of more effective operating points. To address this gap, we propose an enhanced grid-search-based tuning strategy that introduces two innovations: (*i*) expanding the configuration space to include the scheduling profiles of both global and local learning rates as

tunable variables, and (*ii*) re-parametrizing the schedules to make them explicitly driven by the training budget constraints.

To evaluate the impact of our revised search space comprising two-sided learning rate tuning, we carried out an extensive experimental campaign on three benchmarks: two image classification tasks (CIFAR-10 and CIFAR-100) and a causal language modeling task (Penn Treebank, PTB). For each task, we deployed a dedicated deep neural network architecture and simulated three state-of-the-art FL algorithms spanning different training budgets. The study further examined the influence of such search space across several training variables, including data heterogeneity, client participation, and synchronization frequency. Overall, the extensive results collected revealed four key findings: (*i*) expanding the configuration space to jointly optimize initial values and schedules uncovers superior solutions that remain hidden when only initialization is tuned; (*ii*) performance improvements depend on the specific FL algorithm, often altering the ranking observed under the standard configuration space; (*iii*) optimal learning rate settings (global/local initializations and schedules) are inherently tied to the available training budget; and (*iv*) the most effective learning rate schedules consistently follow budget-aware decay patterns.

For clarification purposes, this contribution lies in redefining the learning rate configuration space to enable a more reliable evaluation of FL algorithms, instead of developing a new search algorithm. That said, the proposed configuration space can be readily incorporated into existing FL hyperparameter optimization frameworks [156–159], which already provide advanced search strategies to accelerate tuning and enhance overall efficiency.

6.1.2 Learning Rate Optimization

The learning rate plays a central role in training neural networks, as it controls the step size taken during the optimization process to reduce the loss function. In conventional SGD, it acts as a scalar coefficient that scales the gradient of the loss function, whereas for adaptive optimizers, such as ADAM[172], modify this mechanism by dynamically rescaling the learning rate using past gradient information, which produces parameter-specific step sizes that vary across iterations. These optimizers have been adopted both on the server side[163] and on the client side [173]. However,

despite their adaptive behavior, the initial learning rate still plays a decisive role, as it constrains the maximum step size that can be taken in each optimization update.

In FL, learning rate tuning is more complex than in centralized training, as it requires the joint optimization of two parameters: the global learning rate applied at the server and the local learning rate used during client-side updates. This process is further complicated by FL-specific factors (e.g., the number of participating clients and the synchronization frequency), which can strongly influence the optimal configuration of both learning rates [162].

Another fundamental aspect is the learning rate schedule, which defines how the learning rate evolves during training. An early study in [174] proposed decaying the client learning rate using an inverse time schedule. Despite its empirical benefits, the exponential decay remains the most adopted choice [175, 126]. Decaying the local learning rate has also been shown to be beneficial when applying adaptive optimizers on the server side [163], which has led to the exploration of more sophisticated scheduling strategies [176], which adapt the client learning rate at each local iteration rather than only at the start of each round. Another line of research has focused on scheduling the server learning rate. FEDEXP [126] and FEDHYPER [177] are the two most prominent examples, which adjust the global learning rate based on recent client updates and historical information, respectively. These methods tackle the challenge of providing automatic schedules that are robust to different initializations, and thus designed gradient-based schedules that adapt the learning rate according to the model's progress, rather than relying on predefined decay functions. Yet, an improper initialization of the learning rate can still lead to significant performance drops, and the optimal values remain sub-optimal compared to simpler but well-tuned standard schedules.

A further important aspect which has been largerly overlooked in the FL literature is the interplay between learning rate scheduling and training budget. Previous research in centralized training has shown that the optimal learning rate schedule should depend on the total number of training iterations (especially when the budget is limited), and that decaying the learning rate from a predefined initial value to near zero by the end of training is beneficial [178, 179]. However, the investigation of budget-aware learning rate schedules in FL, particularly when combined with the joint optimization of local and global learning rates, remains largely unexplored. Our work aims to fill this gap by extending the concept of budget-aware schedules

from centralized training to FL, using the number of training rounds as a proxy for the available resources, and exploring the joint optimization of learning rates and schedules on both client and server sides.

6.1.3 Two-sided Learning Rate Tuning

This work focuses on a synchronous federated learning framework operating under standard cross-device conditions. The main workflow has been outlined in Algorithm 2 in the background section 2, which is based on the two-sided learning rate formulation [35, 163, 167].

As a polite reminder, the parameter optimization is applied for both local and global optimizations in Algorithm 1, and the workflow is designed to accommodate any optimizer. As will be detailed in the experimental section, we employed three widely used optimization methods in our study: FEDAVG [37], FEDAVGM [162], and FEDADAM [163].

Independent of the specific optimizers used on the client and server sides, the learning rate settings significantly affect both the final performance and convergence speed. A crucial step for effective tuning is defining the configuration space, which concerns the set of configuration variables to be optimized as well as the set of possible values. The configuration space we are proposing for two-sided optimization of the learning rate comprises four distinct knobs, namely:

1. the initial value of the local learning rate, $\eta_{\mathrm{l},0}$;
2. the initial value of the global learning rate, $\eta_{\mathrm{g},0}$;
3. the local learning rate schedule, $\eta_{\mathrm{l}}^{(r)}$;
4. the global learning rate schedule, $\eta_{\mathrm{g}}^{(r)}$.

A comprehensive description of these variables is provided in the following subsections.

Learning Rates Initial Value

The initial values of the two learning rates ($\eta_{\mathrm{l},0}$ and $\eta_{\mathrm{g},0}$) have a substantial impact on the final accuracy of the trained models and the convergence speed of the learning

process. The range of values that enable effective and efficient training depends on the selected OPTIMIZE function. For example, adaptive optimizers like FEDADAM typically perform best with smaller initial values for the global learning rate compared to SGD-like methods [163]. Furthermore, the interdependence between $\eta_{l,0}$ and $\eta_{g,0}$ can vary across different global optimization algorithms [38]. Improper initialization of the learning rates can prevent convergence, resulting in highly suboptimal model utility. In the worst-case scenario, a poor initialization could render the training efforts futile, producing models with predictive capabilities failing to overcome random guessing.

Similarly to the standard practice, this study investigates the joint exploration of various combinations of $\eta_{l,0}$ and $\eta_{g,0}$, spanning a broad spectrum of initial values (further details are provided in Section 6.1.4) to identify close to optimal solutions through simultaneous tuning.

Learning Rate Schedules

This proposal seeks to expand the configuration space of the two learning rates (local and global) by incorporating the selection of their respective schedules. According to Algorithm 2, both $\eta_l^{(r)}$ and $\eta_g^{(r)}$ are updated through round-based schedules, i.e., they are functions of the current round r. Although alternative methods exist that employ gradient-based schedules, such as FEDEXP and FEDHYPER, this study concentrated on round-based schedules, as they are more easily adaptable to different training budgets. Nonetheless, for the sake of completeness, a comparative analysis between round-based and gradient-based schedules is presented in the experimental section.

While the existing scheduling options are numerous, this study focused on three widely used choices: *Constant*, *Linear*, and *Exponential* schedules. The *Constant* schedule maintains the learning rate at its initial value throughout the entire training process, which can be formally expressed as:

$$\eta(r) = \eta_0, \forall r \in [1, R] \tag{6.1}$$

Conversely, both the *Linear* and *Exponential* schedules foresee a progressive reduction of the learning rate as training unfolds, with the key difference being their decay rates. The *Linear* schedule reduces the learning rate at a constant pace, whereas the

Exponential schedule starts with a fast decay that decelerates towards the end of the training process.

For a more formal definition, the *Linear* schedule can be expressed as:

$$\eta(r) = \eta_0 \cdot \left(1 - \frac{r-1}{T-1}\right), \forall r \in [1, R], \tag{6.2}$$

while the *Exponential* schedule is defined as:

$$\eta(r) = \eta_0 e^{-\frac{\gamma(r-1)}{T-1}}, \forall r \in [1, R], \tag{6.3}$$

where η_0 is the initial value, and γ is a hyperparameter that functions as a control variable, regulating the decay rate. The parameter T is a positive integer that can be adjusted to modify the final value of the learning rate. When $T > R$, the learning rate at the last round will be greater than zero; In contrast, setting $T = R$ yields a budget-aware schedule in which the learning rate decays to zero as the training budget depletes. This approach has demonstrated advantages in centralized training scenarios operating under budget constraints [178, 179]. The present study aims to assess whether this strategy can be effectively adapted for federated learning, and to identify which learning rate schedules yield the most favorable results.

In order to provide a clearer illustration of the effect of budget-aware scheduling, Figures 6.1 and 6.2 depict the evolution of $\eta(r)$ for the *Linear* and *Exponential* schedules, respectively. The aforementioned plots presuppose an initial value of $\eta_0 = 1$, with $\gamma = 4$ for the *Exponential* schedule. Each color in the figures represents a distinct value of T. In round-based learning rate scheduling, the same T is applied irrespective of the budget R. In other words, the learning rate always follows a specific curve. For example, under the assumption that $T = 2000$ (red curve), with a budget of $R = 500$, the learning rate at the last round is 0.75 using the *Linear* schedule or 0.375 for the *Exponential* one. The objective of budget-aware scheduling is to have the learning rate follow a different curve depending on the actual budget R, always reaching zero at the final round. For instance, when $R = 500$, the learning rate follows the green curve; when $R = 2000$, it follows the red curve, providing a more aggressive decay depending on the budget.

In our experiments, we explored an expanded configuration space that encompasses various schedule pairs of the three profile functions (*Constant*, *Linear*, and *Exponential*) for both $\eta_{\mathrm{g}}^{(r)}$ and $\eta_{\mathrm{l}}^{(r)}$, alongside the search for $\eta_{\mathrm{g},0}$ and $\eta_{\mathrm{l},0}$. Fur-

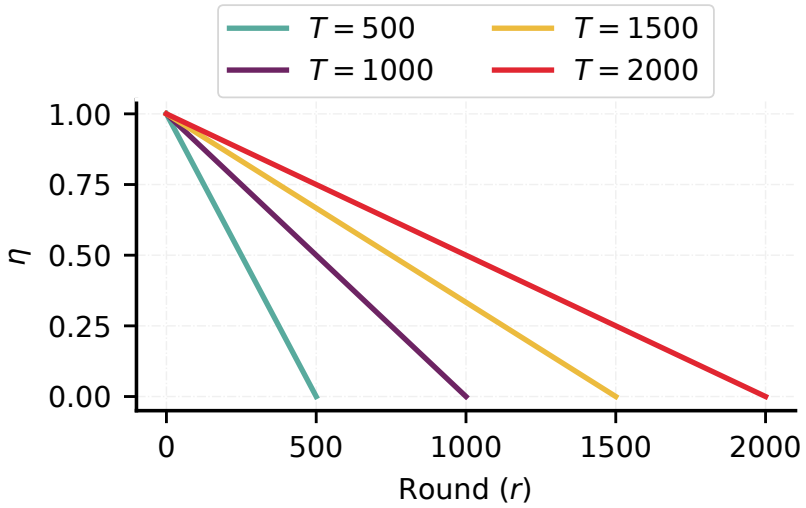


Fig. 6.1 Evolution of the learning rate with a *Linear* schedule for different decay rates regulated by $T \in \{500, 1000, 1500, 2000\}$.

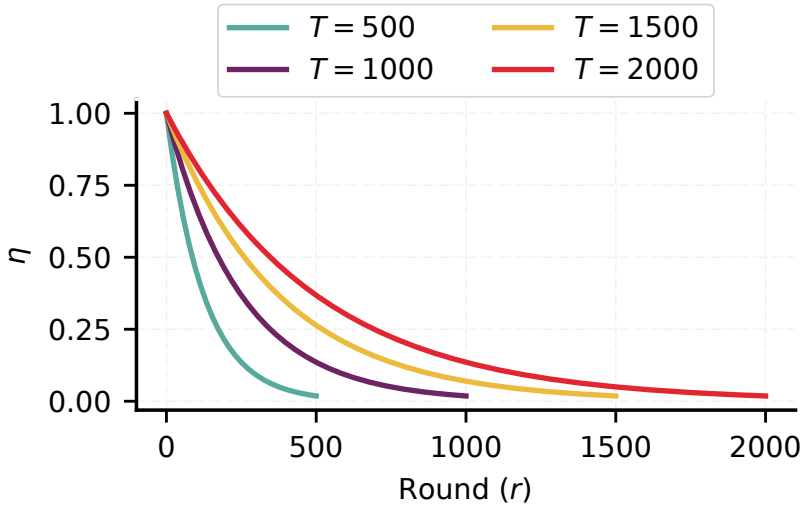


Fig. 6.2 Evolution of the learning rate with an *Exponential* schedule for different decay rates regulated by $T \in \{500, 1000, 1500, 2000\}$.

thermore, we investigated multiple round budgets, comparing the quality of results obtained by leaving T fixed for all schedules, no matter the budget, against budget-aware schedules, obtained with $T = R$. The experimental setup will provide finer details in Section 6.1.4.

6.1.4 Experimental settings

The experimental campaign was conducted through simulation experiments implemented with PyTorch, version 2.1.

Benchmarks

Our experiments spanned three benchmarks: two image classification tasks on CIFAR-10 and CIFAR-100, and a causal language modeling task on Penn Treebank (PTB).

CIFAR-10 (Image Classification) For the CIFAR-10 image classification task, we utilized a standard CNN architecture with five layers (CNN5) as described in [37]. The CNN5 architecture comprises two convolutional layers with 64 filters of size 5×5. These are followed by two fully-connected layers with 384 and 192 neurons, respectively. The final classification layer has 10 neurons, one output for each target class. Each local training iteration processed data in batches of 50 samples. The data partitioning method split the data across the 100 clients through the approach proposed in [180] that provides heterogeneous and imbalanced splits effectively emulating realistic deployment scenarios. The data allocation procedure utilized a Dirichlet distribution with a concentration parameter of 0.3.

CIFAR-100 (Image Classification) For the more complex CIFAR-100, we employed the ResNet-20[181] architecture, using the group normalization layers in place of batch normalization ones, which is a common practice in FL contexts [182]. We applied the same data augmentation technique, batch size, and partition strategy used for CIFAR-10.

PTB (Word-level Causal Language Modeling) The PTB dataset [183] is a text corpus that includes articles from the Wall Street Journal, accounting for 929,000 tokens for training and 82,000 for testing, based on the split used in [184]. The entire vocabulary consists of 10 thousand words plus a special token for indicating the end of a sentence. For the task, we used a two-layer Transformer architecture from [185] with a token embedding size of 200. The model includes two attention

heads and a feed-forward network with 200 hidden units, using the ReLU activation function, and the dropout set at 0.2. The training procedure was performed entirely from scratch, with both the model parameters and vocabulary embeddings initialized at random. During each local iteration, the model processes mini-batches of 10 sequences, each sequence containing 35 tokens. Training data is distributed among 100 clients using the partitioning approach proposed in [186], such that each client receives 250 sequences. This allocation induces heterogeneous class distributions across clients, as each client's texts are restricted to a limited subset of words.

Local Training

The same training setup was used across all benchmarks unless otherwise specified. Each round involves the random selection of 10 concurrent clients out of 100 (10 concurrent clients), i.e., 10% participation rate. Each training participant accomplishes 20 local training steps, employing the SGD optimizer with a weight decay of $1e{-}3$ to minimize the cross-entropy loss function.

Evaluation Metrics

To evaluate the performance of the global model, we used the top-1 categorical accuracy for the image classification tasks (higher is better) and perplexity for the causal language modeling task (lower is better). In both cases, we monitored the model's performance on the test set, averaging the results over the last ten rounds. This averaging is a common practice to mitigate the inherent oscillations of the training process and provide a more reliable comparison [163].

6.1.5 Results

Impact of Schedule Tuning on FL Performance Evaluation

To assess the impact of two-sided learning rate optimization on FL performance evaluation, this study compared three standard federated optimization methods featuring the two-sided learning rate framework described in Chapter 2: FEDAVG [37], FEDAVGM [162], setting the momentum parameter to 0.9, and FEDADAM [163],

setting the first-momentum parameter to 0.9, the second-momentum to 0.99, and the adaptivity to 1e−3.

This study conducted a grid-search algorithm to discover the learning rate settings that maximize performance, following the standard evaluation protocol. The central question addressed in this study is whether tuning only the initial learning rate yields a robust evaluation, and how the outcomes might change when learning rate schedules are systematically incorporated into the optimization process. To address this question, we focused our analysis on a training session with $R = 2000$ iterations as the round budget, comparing the results across the three benchmark tasks introduced in Section 6.1.4.

For each setting (task and scenario), we conducted a grid search for two different Configuration Spaces (CS)s , namely:

i) CS_{standard}: This configuration space explores only the initialization values of the learning rates, $\eta_{\text{g},0}$ and $\eta_{\text{l},0}$. The schedules for both global and local updates are predetermined fixed decay schedules at both the global and the local side and excluded from the hyperparameter search, consistent with the approach adopted in previous studies such as [38, 163, 168–171]. The hyperparameter values examined within this configuration space correspond to those presented in the first two rows of Table 6.1. For the schedules, we implemented a *Constant* schedule for $\eta_{\text{g}}^{(r)}$ and an *Exponential* schedule for $\eta_{\text{l}}^{(r)}$, a configuration is commonly found in the literature. For example, in [175, 126], At each round, the local learning rate was multiplied by a scaling factor of 0.998, which is mathematically equivalent to applying the exponential schedule described in Section 6.3 with $T = 2000$ and decay rate $\gamma = 4$.

ii) CS_{our}: This is our proposed configuration space, which comprises the CS_{standard} sweep range for $\eta_{\text{g},0}$ and $\eta_{\text{l},0}$ as CS_{standard}, but expands the search to include the schedules ($\eta_{\text{g}}^{(r)}$ and $\eta_{\text{l}}^{(r)}$). The pairs of schedules explored are reported in Table 6.1 (the schedule names are abbreviated with their first three letters). The resulting search grid consists of 490 distinct configuration points formed by combining 10 initial values for $\eta_{\text{g},0}$, 7 values for $\eta_{\text{l},0}$, and 7 pairs of $\eta_{\text{g}}^{(r)}$-$\eta_{\text{l}}^{(r)}$ for the global and local schedules. For the *Linear* schedule, we consistently set $T = 2000$, and for *Exponential*, we used $T = 2000$ and $\gamma = 4$.

Table 6.2 provides a comprehensive overview of the results obtained for each benchmark and federated optimizer under the two considered configuration spaces. For each benchmark, the table reports the best-performing settings identified through

Table 6.1 Proposed configuration space for the initial learning rate values ($\log_{10}$ scale) and schedules (Con: Constant, Exp: Exponential, Lin: Linear).

Setting	**Grid**
$\eta_{g,0}$	{0.5, 0.4, 0.3, 0.2, 0.0, -0.5, -1.0, -1.5, -2.0, -2.5}
$\eta_{l,0}$	{0.0, -0.5, -1.0, -1.5, -2.0, -2.5, -3.0}
$\eta_g^{(r)}$-$\eta_l^{(r)}$	{Con-Con, Con-Exp, Con-Lin, Exp-Con, Exp-Exp, Lin-Con, Lin-Lin}

Table 6.2 Comparison of the best configurations from CS_{standard} and CS_{our} across benchmarks and global optimization methods. The table reports the optimal pairs of $\log_{10}$ initial learning rates ($\eta_{g,0}$, $\eta_{l,0}$), the best global/local schedule types (Fixed, Exponential, Linear), and the corresponding metrics evaluated on the test set. For CS_{our}, accuracy gains over CS_{standard} are also reported. Metrics are Top-1 Accuracy (%) for CIFAR-10/100 (higher is better) and Perplexity for PTB (lower is better).

Benchmark	**Optimizer**	**Search Space**	$\eta_{g,0}/\eta_{l,0}$	**Schedules**	**Metric (Gain)**
CIFAR-10 CNN5	**FEDAVG**	CS_{standard}	0.3/-0.5	Con-Exp	79.11
		CS_{our}	0.3/-0.5	Con-Lin	81.81 (+2.70)
	FEDAVGM	CS_{standard}	0.2/-1.5	Con-Exp	80.93
		CS_{our}	0.2/-1.5	Con-Lin	83.48 (+2.55)
	FEDADAM	CS_{standard}	-2.0/-1.5	Con-Exp	77.88
		CS_{our}	-1.5/-2	Lin-Con	81.81 (+3.93)
CIFAR-100 ResNet-20	**FEDAVG**	CS_{standard}	0.2/-0.5	Con-Exp	43.78
		CS_{our}	0.4/-0.5	Con-Lin	50.90 (+7.12)
	FEDAVGM	CS_{standard}	0.0/-1.5	Con-Exp	51.05
		CS_{our}	0.2/-1.0	Con-Lin	56.96 (+5.91)
	FEDADAM	CS_{standard}	-1.5/-0.5	Con-Exp	52.05
		CS_{our}	-1.5/-0.5	Con-Lin	55.40 (+3.35)
PTB Transformer	**FEDAVG**	CS_{standard}	0.2/-0.5	Con-Exp	117.34
		CS_{our}	0.2/-0.5	Con-Exp	117.34 (-0.00)
	FEDAVGM	CS_{standard}	0.0/-1.0	Con-Exp	124.59
		CS_{our}	0.0/-1.5	Con-Lin	115.18 (-9.41)
	FEDADAM	CS_{standard}	-2/-1.0	Con-Exp	120.56
		CS_{our}	-2/-1.5	Con-Lin	115.50 (-5.06)

the grid search spanning the related configuration space. Specifically, it includes the optimal combination of the initial learning rates $\eta_{g,0}/\eta_{l,0}$, the selected pair of global-local scheduling policies, and the corresponding evaluation metric. In the case

of CS_{standard}, the scheduling pair is consistently set to *Con-Exp*, as the scheduling strategy is predefined and therefore not part of the optimization process.

The results highlight the critical role of jointly optimizing two-sided schedule functions, from which several key insights emerge.

First, the *Con-Exp* schedule pair used in CS_{standard} proves suboptimal, as configurations from CS_{our} consistently deliver superior performance. On CIFAR-10, accuracy improves by +2.55% with FEDAVGM and +3.93% with FEDADAM. Interestingly, with FEDAVG, both configuration spaces share identical learning rate initializations, yet switching from Con-Exp to Con-Lin schedules alone yields a +2.70% accuracy increase, demonstrating the importance of two-sided schedule tuning. Even larger gains occur on CIFAR-100 (up to +7.12% with FEDAVG), showing that the impact grows with task complexity. A similar trend appears on PTB, where CS_{our} lowers model perplexity by up to 9.41 (with FEDAVGM).

Second, learning rate initialization and scheduling functions are intertwined. Optimal values of $\eta_{\text{g},0}$ and $\eta_{\text{l},0}$ often differ between CS_{standard} and CS_{our}, indicating that co-optimization is favorable. For instance, with FEDAVGM on CIFAR-100, the best initialization shifts from (0, -1.5) to (0.2, -1), confirming the need for joint tuning.

Third, two-sided schedule optimization can alter the ranking of optimization strategies. On CIFAR-10, FEDAVG outperforms FEDADAM in CS_{standard} (79.11% vs. 77.88%), but they equally perform (81.81%) under CS_{our}. On CIFAR-100, FEDADAM leads in CS_{standard} (52.05%), whereas FEDAVGM prevails in CS_{our} (56.96%). Similarly, on PTB, FEDAVG performs best in CS_{standard} (117.34) but is surpassed by FEDAVGM in CS_{our} (115.18). These findings indicate that (*i*) restricting the configuration space to fixed schedules leads to suboptimal tuning, and (*ii*) neglecting two-sided schedule optimization can produce misleading comparisons among federated optimizers.

To further substantiate our findings and quantify the benefits of two-sided learning rate schedule optimization, Table 6.3 reports the best-performing initialization pairs for each schedule combination defined in Table 6.1. The analysis focuses on the FEDAVGM optimizer, though similar patterns are observed for FEDAVG and FEDADAM. On CIFAR-10, the top-performing schedule pair (Con-Lin) achieves a 1.19% accuracy gain over the second-best (Lin-Lin), reaching 83.48% vs. 82.29%, and a 4.42% improvement over the weakest configuration (Exp-Exp, 79.06%). On

Table 6.3 Best-performing initialization pairs of global and local learning rates ($\log_{10}$ of $\eta_{g,0}$ and $\eta_{l,0}$, respectively) for each schedule combination (Con: Constant, Exp: Exponential, Lin: Linear) in CS_{our}, along with the corresponding evaluation metric on the test set. The reported metric is Top-1 Accuracy (%) for CIFAR-10/100 (higher is better) and Perplexity for PTB (lower is better).

Benchmark	$\eta_{g,0}/\eta_{l,0}$	Schedules	Metric
CIFAR-10 CNN5	-0.5/-1.0	Con-Con	79.06
	0.2/-1.5	Con-Exp	80.93
	0.2/-1.0	Exp-Con	80.65
	0.0/-1.0	Exp-Exp	80.38
	0.2/-1.5	Con-Lin	83.48
	0.0/-1.0	Lin-Con	82.15
	0.0/-1.0	Lin-Lin	82.29
CIFAR-100 ResNet-20	0.0/-1.0	Con-Con	52.02
	0.0/-1.5	Con-Exp	51.05
	0.2/-1.0	Exp-Con	51.09
	0.2/-1.5	Exp-Exp	49.17
	0.2/-1.0	Con-Lin	56.96
	0.2/-1.0	Lin-Con	56.01
	0.2/-1.0	Lin-Lin	55.24
PTB Transformer	0.0/-2	Con-Con	120.86
	0.0/-1.0	Con-Exp	124.59
	-0.5/-1.0	Exp-Con	131.31
	-0.5/-0.5	Exp-Exp	119.11
	0.0/-1.5	Con-Lin	115.18
	0.0/-1.5	Lin-Con	116.90
	-0.5/-1.0	Lin-Lin	119.69

CIFAR-100, accuracy gains range from 0.95% (Con-Lin vs. Lin-Con) up to 7.79% (Con-Lin vs. Exp-Exp). For PTB, the Con-Lin schedule reduces model perplexity by 1.72 (vs. Lin-Con) and by as much as 16.13 (vs. Exp-Con). These results confirm that the choice of global-local learning rate schedules has a substantial impact on model performance, reinforcing the importance of jointly optimizing both sides for consistent and reliable federated training outcomes.

Impact of Budget-Aware Learning Rate Schedules

In this subsection, we investigate how learning rate configurations influence training quality under different computational budgets, and whether adapting these settings to the target budget can enhance final performance. To this end, we repeated the grid-search exploration under varying round budgets, specifically $R \in \{500, 1000, 1500\}$, and compared three configuration space settings:

1. CS_{standard}, employing the fixed *Con-Exp* schedule pair with $T = 2000$ across all budgets (as in the previous subsection);
2. CS_{our} *with* $T = 2000$, exploring the initial values and schedule pairs defined in Table 6.1, while keeping $T = 2000$ constant for all schedules;
3. CS_{our} *with* $T = R$, using the same search space as above but introducing budget-aware configurations for the *Linear* and *Exponential* schedules, i.e., setting $T = R$.

All experiments were conducted using the same global optimizer, FEDAVGM.

Table 6.4 reports the best-performing configurations for each benchmark across the three explored settings. Overall, the optimal configurations identified within CS_{our}, both with $T = 2000$ and $T = R$, consistently differ from those obtained using CS_{standard}. For CIFAR-10 and CIFAR-100, accuracy improves with larger training budgets regardless of the configuration space, yet the best results are achieved with CS_{our} *with* $T = R$, yielding gains of 3.08%-4.22% on CIFAR-10 and 2.30%-4.62% on CIFAR-100.

Another interesting and counterintuitive point is the instability of the standard search space. CS_{standard} provides a non-decreasing trend for growing budgets on the PTB benchmark, yielding the lowest perplexity (122.63) at $R = 1000$, which grows for additional budget. This suggests that suboptimal learning rate adaptations can negatively affect convergence stability. Such issues are mitigated under CS_{our}, where perplexity consistently decreases as the budget increases. Compared to CS_{standard}, CS_{our} *with* $T = 2000$ reduces perplexity by 2.69-7.20, and CS_{our} *with* $T = R$ achieves even greater improvements (5.34–7.19), reaching near-optimal performance (117.42) already at the lowest budget. These results confirm that adaptive, budget-aware learning rate schedules accelerate convergence and enhance efficiency.

Table 6.4 Comparison of the best-performing configurations in CS_{standard}, CS_{our} with $T = 2000$, and CS_{our} with $T = R$. Only the best metric per benchmark and budget is highlighted in yellow.

Benchmark	R	Search Space	$\eta_{g,0}/\eta_{l,0}$	Schedules	Metric (Gains)
CIFAR-10 CNN5	500	CS_{standard}	-0.5/-1.0	Con-Exp	75.40
		$CS_{\text{our,T=2000}}$	0.0/-1.0	Exp-Exp	77.67 (+2.27)
		$CS_{\text{our,T=R}}$	0.0/-1.5	Con-Lin	78.48 (+3.08)
	1000	CS_{standard}	-0.5/-0.5	Con-Exp	77.48
		$CS_{\text{our,T=2000}}$	0.0/-1.0	Exp-Exp	79.46 (+1.98)
		$CS_{\text{our,T=R}}$	0.0/-1.0	Con-Lin	81.70 (+4.22)
	1500	CS_{standard}	0.0/-1.0	Con-Exp	79.36
		$CS_{\text{our,T=2000}}$	0.0/-1.0	Lin-Lin	81.26 (+1.90)
		$CS_{\text{our,T=R}}$	0.0/-1.0	Con-Lin	83.03 (+3.67)
CIFAR-100 ResNet-20	500	CS_{standard}	0.0/-1.0	Con-Exp	43.18
		$CS_{\text{our,T=2000}}$	0.0/-1.0	Lin-Lin	44.35 (+1.17)
		$CS_{\text{our,T=R}}$	0.0/-1.0	Con-Lin	45.48 (+2.30)
	1000	CS_{standard}	0.0/-1.0	Con-Exp	48.25
		$CS_{\text{our,T=2000}}$	0.2/-1.0	Lin-Con	50.45 (+2.20)
		$CS_{\text{our,T=R}}$	0.0/-1.0	Con-Lin	51.46 (+3.21)
	1500	CS_{standard}	0.0/-1.5	Con-Exp	49.68
		$CS_{\text{our,T=2000}}$	0.2/-1.0	Lin-Con	53.79 (+4.11)
		$CS_{\text{our,T=R}}$	0.2/-1.0	Con-Lin	54.30 (+4.62)
PTB Transformer	500	CS_{standard}	0.0/-1.0	Con-Exp	122.76
		$CS_{\text{our,T=2000}}$	0.0/-1.0	Con-Exp	122.76 (-0.00)
		$CS_{\text{our,T=R}}$	0.0/-0.5	Con-Exp	117.42 (-5.34)
	1000	CS_{standard}	0.0/-1.0	Con-Exp	122.63
		$CS_{\text{our,T=2000}}$	0.0/-1.5	Con-Lin	119.94 (-2.69)
		$CS_{\text{our,T=R}}$	-0.5/-0.5	Con-Exp	118.21 (-4.42)
	1500	CS_{standard}	0.0/-1.0	Con-Exp	123.43
		$CS_{\text{our,T=2000}}$	0.0/-1.5	Con-Lin	116.23 (-7.20)
		$CS_{\text{our,T=R}}$	0.0/-1.0	Con-Exp	116.24 (-7.19)

In summary, the analysis demonstrates that (*i*) suboptimal learning rate schedules, such as those in CS_{standard}, can negate the benefits of extended training, leading to inefficient resource usage, and (*ii*) tailoring learning rate schedules to the available training budget significantly improves overall model performance.

Impact of Training Variables on Learning Rate Settings

Table 6.5 Comparison of the best-performing configurations in CS_{standard}, CS_{our} with $T = 2000$, and CS_{our} with $T = R$ on CIFAR-10, trained with FEDAVGM under varying numbers of participating clients ($|\mathscr{S}^{(r)}|$). Only the best Top-1 Accuracy per benchmark and budget is highlighted in yellow.

$\lvert\mathscr{S}^{(\mathbf{r})}\rvert$	R	Search Space	$\eta_{\text{g},0}/\eta_{\text{l},0}$	Schedules	Metric (Gains)
2	500	CS_{standard}	-1/-1.0	Con-Exp	65.97
		$CS_{\text{our, T=2000}}$	-0.5/-1.0	Exp-Exp	69.34 (+3.37)
		$CS_{\text{our, T=R}}$	-0.5/-1.0	Con-Lin	71.02 (+5.05)
	1000	CS_{standard}	-1/-0.5	Con-Exp	71.95
		$CS_{\text{our, T=2000}}$	-0.5/-1.0	Exp-Con	73.04 (+1.09)
		$CS_{\text{our, T=R}}$	-0.5/-1.0	Con-Lin	78.67 (+6.72)
	1500	CS_{standard}	-0.5/-1.0	Con-Exp	74.17
		$CS_{\text{our, T=2000}}$	-0.5/-1.0	Exp-Con	75.66 (+1.49)
		$CS_{\text{our, T=R}}$	-0.5/-1.0	Con-Lin	77.88 (+3.71)
	2000	CS_{standard}	-0.5/-1.0	Con-Exp	76.75
		$CS_{\text{our, T=2000}}$	-0.5/-1.0	Con-Lin	80.34 (+3.59)
		$CS_{\text{our, T=R}}$	-0.5/-1.0	Con-Lin	80.34 (+3.59)
5	500	CS_{standard}	-0.5/-1.0	Con-Exp	70.68
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Exp-Con	71.52 (+0.84)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	75.28 (+4.60)
	1000	CS_{standard}	-0.5/-1.0	Con-Exp	77.09
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Exp-Con	79.29 (+2.22)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	81.86 (+4.77)
	1500	CS_{standard}	0.0/-1.5	Con-Exp	79.75
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Lin-Lin	80.51 (+0.76)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	83.01 (+3.26)
	2000	CS_{standard}	0.0/-1.5	Con-Exp	81.86
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Con-Lin	83.02 (+1.36)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	83.02 (+1.36)

The optimal initialization and scheduling of learning rates are inherently tied to other training variables. To capture these dependencies, we extended our analysis to examine three fundamental FL parameters that directly affect model performance and computational cost [39]: the number of participating clients ($|\mathscr{S}^{(r)}|$), the number of local training steps per round (K), and the degree of data heterogeneity across clients.

The number of active clients acts as a control knob for resource allocation in federated systems. We evaluated smaller values of $|\mathscr{S}^{(r)}|$ (2 and 5) to emulate resource-constrained scenarios and quantify the potential gains from learning rate optimization. Since participating devices are often battery-powered and constrained in their local computation, we also varied K (5 and 10) to represent limited on-device training. Lastly, we considered the effect of data heterogeneity by experimenting with lower Dirichlet parameters ($\alpha \in \{0.1, 0.2\}$), corresponding to increasingly non-IID data distributions. Together, these configurations capture realistic deployment conditions characterized by scarce computational resources and high statistical diversity, both known to hinder convergence.

Tables 6.5, 6.6, and 6.7 summarize the results for the CNN-5 model trained on CIFAR-10 using FEDAVGM under different round budgets, i.e., with R spanning $\{500, 1000, 1500, 2000\}$. Beyond confirming the conclusions drawn in Section 6.1.5, i.e., the importance of schedule optimization and the interplay between initialization values and schedule types, the results provide additional insights.

In all cases, budget-aware schedules yield the highest accuracies, with notable improvements in the most challenging conditions. For instance, training with two active clients over 1000 rounds improved accuracy by +6.72% (Tab 6.5, second row), while high heterogeneity conditions at 500 rounds achieved +4.25% gain (Tab 6.7, first row). Moreover, Table 6.5 (with $|\mathscr{S}^{(r)}| = 5$) shows that CS_{our} *with* $T = R$ achieves 81.86% accuracy after only 1000 rounds, matching the performance of CS_{standard} at 2000 rounds, effectively halving the required training budget. Similar patterns emerge in Tables 6.6 and 6.7, demonstrating that learning rate schedules lead to improved training efficiency under diverse and constrained FL settings.

Table 6.6 Comparison of the best-performing configurations in CS_{standard}, CS_{our} with $T = 2000$, and CS_{our} with $T = R$ on CIFAR-10, trained with FEDAVGM under varying numbers of local training steps (K). Only the best Top-1 Accuracy per budget is highlighted in yellow.

K	R	Search Space	$\eta_{g,0}/\eta_{l,0}$	Schedules	Metric (Gains)
5	500	CS_{standard}	-0.5/-0.5	Con-Exp	71.63
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Exp-Exp	72.55 (+0.92)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	74.35 (+2.72)
	1000	CS_{standard}	-0.5/-0.5	Con-Exp	76.82
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Lin-Lin	77.36 (+0.54)
		$CS_{\text{our, T=R}}$	-0.5/-0.5	Con-Lin	79.41 (+2.59)
	1500	CS_{standard}	-0.5/-0.5	Con-Exp	77.54
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Lin-Lin	78.88 (+1.34)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	82.13 (+4.59)
	2000	CS_{standard}	-0.5/-0.5	Con-Exp	79.74
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Con-Lin	82.18 (+2.44)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	82.18 (+2.44)
10	500	CS_{standard}	-0.5/-0.5	Con-Exp	73.65
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Exp-Con	75.36 (+1.71)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	78.22 (+4.57)
	1000	CS_{standard}	0.0/-1.0	Con-Exp	78.84
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Lin-Lin	79.93 (+1.09)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	81.92 (+3.08)
	1500	CS_{standard}	0.0/-1.0	Con-Exp	81.16
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Lin-Lin	82.03 (+0.87)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	83.20 (+2.04)
	2000	CS_{standard}	0.0/-1.0	Con-Exp	81.54
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Con-Lin	83.03 (+1.49)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	83.03 (+1.49)

Table 6.7 Comparison of the best-performing configurations in CS_{standard}, CS_{our} with $T = 2000$, and CS_{our} with $T = R$ on CIFAR-10, trained with FEDAVGM under varying data distributions (lower α indicates higher statistical heterogeneity). Only the best Top-1 Accuracy per benchmark and budget is highlighted in yellow.

α	R	Search Space	$\eta_{g,0}/\eta_{l,0}$	Schedules	Metric (Gains)
0.1	500	CS_{standard}	-0.5/-0.5	Con-Exp	68.28
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Exp-Con	69.71 (+1.43)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Lin-Con	72.53 (+4.25)
	1000	CS_{standard}	-0.5/-0.5	Con-Exp	73.37
		$CS_{\text{our, T=2000}}$	-0.5/-0.5	Con-Exp	73.37 (+0.00)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Lin-Con	76.46 (+3.09)
	1500	CS_{standard}	-0.5/-0.5	Con-Exp	75.09
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Lin-Lin	76.60 (+1.61)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Lin-Con	78.36 (+3.27)
	2000	CS_{standard}	-0.5/-0.5	Con-Exp	75.68
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Lin-Con	79.62 (+3.94)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Lin-Con	79.62 (+3.94)
0.2	500	CS_{standard}	-0.5/-1.0	Con-Exp	74.48
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Exp-Con	76.42 (+1.94)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	77.83 (+3.35)
	1000	CS_{standard}	-0.5/-0.5	Con-Exp	77.33
		$CS_{\text{our, T=2000}}$	0.0/-1.5	Lin-Lin	78.77 (+1.44)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	80.11 (+2.78)
	1500	CS_{standard}	-0.5/-0.5	Con-Exp	78.50
		$CS_{\text{our, T=2000}}$	0.0/-0.5	Lin-Lin	80.76 (+2.26)
		$CS_{\text{our, T=R}}$	0.0/-1.5	Con-Lin	81.23 (+2.73)
	2000	CS_{standard}	0.0/-1.5	Con-Exp	80.06
		$CS_{\text{our, T=2000}}$	0.0/-1.0	Con-Lin	82.35 (+2.29)
		$CS_{\text{our, T=R}}$	0.0/-1.0	Con-Lin	82.35 (+2.29)

Comparison between Round-based and Gradient-based Schedules

This subsection evaluates two-sided learning rate tuning under gradient-based scheduling strategies, comparing its performance with recently proposed adaptive methods: FEDEXP [126] and FEDHYPER [177]. These approaches dynamically adjust the learning rates throughout training based on gradient information, providing a more responsive alternative to fixed schedules.

We conducted a quantitative comparison across three configuration spaces:

1. FEDEXP *CS*, which optimizes only the initial global and local learning rates ($\eta_{g,0}$, $\eta_{l,0}$) while adopting the FEDEXP schedule. We used the FEDEXP-M variant with $\varepsilon = 1e{-}3$ and an *Exponential* local schedule ($T = 2000$, $\gamma = 4$).

2. FEDHYPER *CS*, which searches for the same initial values under the FEDHYPER schedule. We adopted the FEDHYPER-G version with a bound value of 3 and a *Constant* local schedule.

3. CS_{our}, which jointly optimizes initialization values and round-based schedules as defined in Table 6.1.

For both FEDEXP *CS* and FEDHYPER *CS*, the initial learning rate grids match those listed in the first two rows of Table 6.1. Each search explores 70 initialization pairs, derived from 10 values of $\eta_{g,0}$ and 7 values of $\eta_{l,0}$. All experiments employ FEDAVGM as the global optimizer.

Since FEDEXP and FEDHYPER are designed to be robust against variations in learning rate initialization, we explicitly examined this aspect in our analysis. To quantify robustness, we counted the number of initialization pairs that led to successful model training[1]. For these successful configurations, we further computed the minimum, mean, and maximum test performance. The results are summarized in Table 6.8.

For a fair comparison with round-based scheduling, the column CS_{our} reports analogous statistics obtained from grid searches spanning the same initialization pairs used in FEDEXP/FEDHYPER *CS*, but employing round-based schedules instead. This study reports the best-performing schedule configuration for each benchmark, namely *Con-Lin* for CIFAR-10/100 and *Con-Exp* for PTB.

[1] A training is considered successful if the resulting accuracy exceeds random guessing.

Table 6.8 Robustness to learning rate initialization values in FEDEXP *CS*, FEDHYPER *CS*, and CS_{our} (Con–Lin schedules). Reported are the count of successful initialization pairs and the minimum, mean, and maximum Top-1 Accuracy (or Perplexity for PTB) over the configuration space. Results are shown for CIFAR-10/100 and PTB trained with FEDAVGM for 2000 rounds. The best metric per row is highlighted in yellow.

Benchmark	Metric	FedExP CS	FedHyper CS	CS_{our}
CIFAR-10	**Count**	30	50	30
	Min Top-1	77.72	70.84	49.39
	Mean Top-1	79.21	76.80	75.81
	Max Top-1	80.50	79.45	83.48
CIFAR-100	**Count**	50	60	47
	Min Top-1	10.77	15.31	10.51
	Mean Top-1	32.93	41.01	36.61
	Max Top-1	44.29	52.88	56.96
PTB	**Count**	30	50	45
	Max PPL	211.22	416.15	639.01
	Mean PPL	176.37	222.09	261.03
	Min PPL	133.55	132.17	115.18

Table 6.9 Performance of the best-performing configurations in FEDEXP *CS*, FEDHYPER *CS*, and CS_{our} with $T = R$. Top-1 Accuracy (higher is better) for CIFAR-10/100 and Perplexity (lower is better) for PTB are reported. Values in parentheses indicate the metric gap relative to the best-performing configuration. The best metric per row is highlighted in yellow.

Benchmark	Metric	FedExP CS	FedHyper CS	CS_{our}
CIFAR-10	**R=500**	59.81 (-18.76)	75.52 (-2.96)	78.48
	R=1000	73.24 (-8.46)	77.00 (-4.70)	81.70
	R=1500	77.98 (-5.05)	78.05 (-4.98)	83.03
CIFAR-100	**R=500**	21.77 (-23.71)	40.18 (-5.30)	45.48
	R=1000	32.54 (-18.92)	48.42 (-3.04)	51.46
	R=1500	40.25 (-14.05)	51.41 (-2.89)	54.30
PTB	**R=500**	174.76 (+57.34)	152.38 (+34.96)	117.42
	R=1000	128.20 (+9.99)	120.96 (+2.75)	118.21
	R=1500	130.32 (+14.08)	125.02 (+8.78)	116.24

The results confirm the effectiveness of gradient-based schedules in mitigating the adverse effects of sub-optimal learning rate initialization. For example, in CIFAR-10, FEDHYPER *CS* yields a larger number of successful runs than CS_{our} (50 vs. 30). Both

FEDEXP *CS* and FEDHYPER *CS* also display reduced sensitivity to initialization choices, as reflected by narrower accuracy ranges compared to CS_{our}. Nevertheless, the highest accuracy is achieved with the round-based schedules in CS_{our}, which outperform FEDEXP *CS* and FEDHYPER *CS* by 2.98% and 4.03%, respectively. Similar patterns are observed for CIFAR-100 and PTB. However, gradient-based schedules exhibit higher variability across datasets, indicating that their robustness is not consistent. For instance, in CIFAR-100, the accuracy gap between the best and worst configurations in FEDEXP *CS* reaches 33.52% (10.77% vs. 44.29%), compared to only 2.78% on CIFAR-10. Overall, while gradient-based schedules simplify hyperparameter tuning and enhance resilience to poor initializations, they tend to trade off peak performance. Conversely, round-based schedules demand more extensive search but yield higher final accuracy.

A further limitation of gradient-based schedules lies in their lack of budget adaptability. To assess this, we evaluated them under varying training budgets, as summarized in Table 6.9, alongside the best-performing configurations of CS_{our} with $T = R$. Across all benchmarks and round budgets, gradient-based schedules show pronounced performance degradation. For instance, FEDEXP *CS* suffers up to a 23.71% accuracy drop on CIFAR-100 at $R = 500$, while FEDHYPER *CS* declines by 5.20%. These findings further highlight the advantages of budget-aware round-based scheduling, which ensures better adaptability and stability under constrained training conditions.

6.1.6 Discussion and Final Remarks

This work highlights the critical role of two-sided learning rate schedule optimization in assessing the performance of cross-device FL. Our results demonstrate that tuning the initial learning rate values alone is insufficient to achieve optimal outcomes, and substantial gains can be realized by considering the learning rate schedules as tunable variables, particularly under constrained training budgets.

This work used an extensive experimental campaign to demonstrate that adapting local and global learning rates to the available round budget significantly improves training efficiency, an aspect often overlooked in prior FL research, yet essential for practical deployments. The parametric analyses conducted further revealed that schedule tuning can lead to notable resource savings, emphasizing that any

rigorous evaluation of FL algorithms should incorporate learning rate schedule optimization and budget-aware re-tuning. Ignoring this step risks drawing incomplete or misleading conclusions.

The insights emerged in this study highlight the need to optimize the learning rate as training proceeds through different schedules. Future work may focus on developing more efficient search strategies for the proposed configuration space, limiting the effective search space, and finding even better solutions. Additionally, exploring alternative types of learning rate schedules could unlock further benefits for FL in diverse and resource-constrained environments.

6.2 Benchmarking ppFL for Smart Grids: A Case Study on Sociodemographic Characteristics

This thesis concludes with a practical use-case study that evaluates privacy-preserving federated learning under realistic data conditions. However, privacy-sensitive IoT data are often unavailable for public testbeds, which hinders feasibility assessments and benchmark studies and makes direct deployment of new ppFL solutions risky and costly. In light of this, this final contribution advances a novel use case targeting a fine-grained sociodemographic classification of smart meter data. It combines a domain-specific synthetic data generation pipeline with an HE-based FL evaluation framework, using a synthetic yet realistic workload to show how HE-based FL can overcome the limitations of isolated local training and to quantify the remaining gap with respect to centralized training before real-world deployment.

6.2.1 Introduction and Motivation

The socio-demographic composition of the household strongly shapes the energy consumption patterns observed in residential dwellings. Variables such as occupational status, income bracket, and age distribution play a decisive role in determining when and how electricity is consumed over the course of the day. Direct access to such information is, however, uncommon because it relies on detailed surveys, which are both resource-intensive and often hindered by consumers' reluctance. Despite these limitations, it is possible to derive meaningful approximations of household

characteristics by analyzing high-resolution smart-meter data [187]. For example, dwellings inhabited by two individuals engaged in full-time employment typically show reduced electricity demand during working hours, whereas households composed of retired occupants tend to exhibit sustained daytime usage. These distinctive temporal signatures enable energy providers to infer key household attributes directly from consumption profiles, offering valuable insights without the need for intrusive data-collection methods.

Energy utilities and retailers benefit from accurately profiling their customers for different strategic interests. Indeed, this knowledge enables targeted demand-response programs that minimize impact to daily routines [188]. It also supports the design of personalized tariff schemes, tailored incentives, and service packages that improve customer satisfaction and long-term loyalty, by providing a closer alignment with consumers' expectations. Profiling further allows the identification of unusual consumption behaviors, rewarding efficient users while assisting those with excessive demand [189]. In turn, consumers gain from more responsive, efficient, and personalized energy services.

In deregulated energy markets, consumers are free to select their preferred electricity provider, fostering competition among retailers through competitive tariffs and diversified service offerings. Although this environment stimulates price reductions and broadens consumer choice, it simultaneously introduces challenges in data management, as smart meter information is collected and stored across decentralized energy utilities. Stringent privacy regulations further constrain this process, frequently preventing utilities from sharing customer-level data with external actors and thereby restricting insights into household characteristics to their own consumer base [190]. FL has emerged as a compelling avenue to address these limitations, as it facilitates the collective training of predictive models across different stakeholders with no need for data exchange [191]. FL allows utilities to extract knowledge from multiple datasets with no need for their centralization, thereby enhancing the prediction and generalization capability of modelling household energy behavior. This approach not only supports enhanced personalization of services and improved system-level efficiency but also ensures compliance with the regulatory standards governing data protection.

Building on these considerations, the present work investigates the potential of FL to support the development of machine-learning models for household profile

identification. The contributions of this study can be summarized as a twofold, intertwined contribution. First, a synthetic data generation pipeline is introduced, designed to produce highly realistic electricity consumption traces. This pipeline is based on an agent-based simulation framework [192], [193], which assigns each household member a set of sociodemographic characteristics and reconstructs their energy usage behavior with fine-grained temporal resolution. The resulting dataset not only captures the variability of consumption patterns across diverse household types but also provides a robust benchmark for evaluating the performance of collaborative training among multiple utilities. Second, a privacy-preserving FL architecture is implemented, based on the HE paradigm. This design enables decentralized model training while safeguarding sensitive customer information from honest-but-curious adversaries. Experimental results show that models trained separately by individual utilities often perform poorly due to inconsistent and fragmented data. By contrast, the proposed FL approach operates under privacy-preserving constraints, and it achieves performance levels approaching those of an idealized centralized model. These findings underscore the effectiveness of FL in enabling accurate household characterization within decentralized energy utilities, while maintaining compliance with data protection requirements.

6.2.2 Preliminaries on Data-driven Electricity Consumption Analysis

ML for Electricity Consumption Analysis

Previous research has extensively examined the task of inferring household characteristics from smart meter data by applying classification techniques to daily power consumption profiles. These methods provide detailed insights into consumer behavior by associating distinct consumption patterns with sociodemographic attributes such as household size, employment status, income distribution, and age composition. This information enables utilities to refine customer segmentation and design energy services that are both more effective and better aligned with the needs of different user groups. For example, in [194] was explored a range of classification algorithms for predicting household attributes using smart meter readings. Their methodology relied on feature engineering, where raw load data were aggregated by extracting statistical indicators, e.g., average, minimum, and maximum values, across

multiple temporal windows. Similar strategies have been adopted in other studies, frequently in combination with Support Vector Machines [195]. In contrast, deep learning approaches are particularly advantageous because they can automatically learn discriminative features during training, eliminating the need for manual feature extraction and allowing models to process raw consumption data directly. Most recent contributions in this area have focused on deep learning classifiers trained on unprocessed load measurements. Among these, 1D-CNNs have gained prominence as the preferred architecture for household attribute inference [196, 197]. CNNs are especially effective due to their capacity to construct hierarchical representations from sequential data, making them well-suited to the analysis of large-scale power consumption profiles. More recent studies have further enhanced this framework by integrating CNNs with Long Short-Term Memory (LSTM) networks, improving the ability to capture temporal dependencies and dynamic variations in energy usage patterns [198].

FL for Smart Meter data

FL enables multiple energy utilities to jointly develop a global model without the need to disclose or exchange their proprietary datasets. Within this paradigm, each energy utility trains a local version of the model using its own private data and transmits only the model parameters or updates to a central coordinating server. The server then aggregates these contributions to iteratively refine and improve the shared global model over successive synchronization rounds. This process not only preserves data locality but also offers privacy advantages, making it possible to extend machine-learning services across organizational boundaries while remaining compliant with stringent data protection frameworks [22].

Fig. 6.3 shows schematically the FL workflow in such an application scenario. It involves two primary entities: a central third-party server responsible for coordinating model aggregation and synchronization, and a set of participating energy utilities. Each energy utility possesses a private dataset $\mathscr{D}_c$, comprising energy consumption data collected from the smart meters and labelled with the sociodemographic attributes, to be utilized for local model training.

FL has emerged as a powerful paradigm in the smart grid domain, demonstrating versatility across a broad range of applications, including anomaly detection, load forecasting, and household profiling. In the field of anomaly detection, [199] intro-

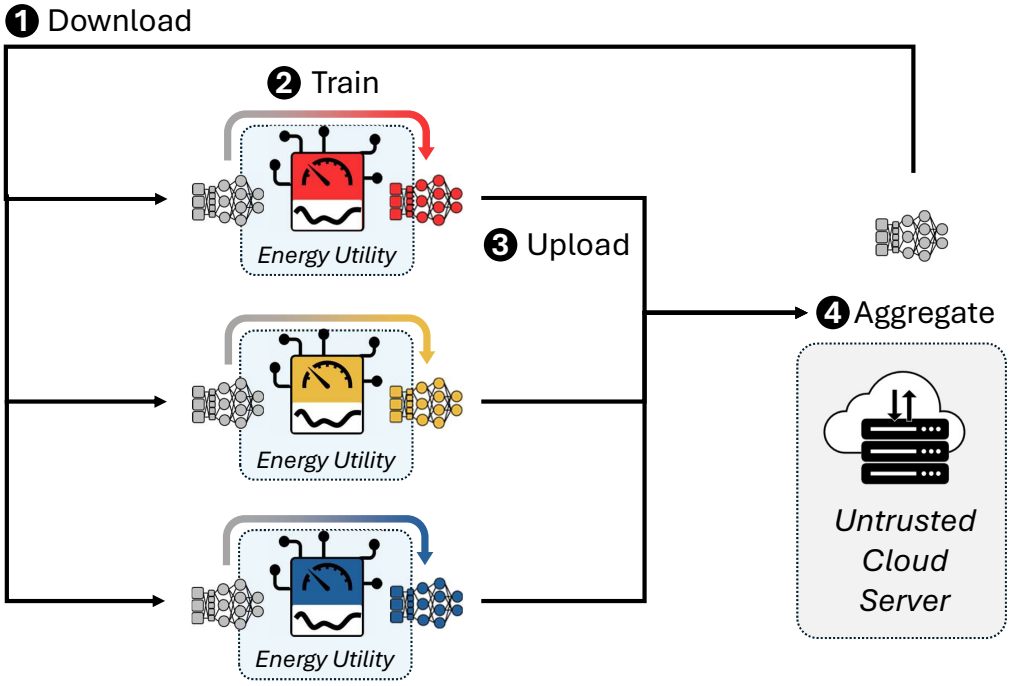


Fig. 6.3 Federated Learning for smart meters data.

duced an unsupervised FL framework designed for the identification of irregular patterns in data collected by remote terminal units at electrical substations. Other studies have applied FL to detect cyberattacks within transmission systems [200], as well as to identify energy theft through the analysis of household electricity consumption records [201]. In the context of load forecasting, FL has been leveraged to train models on distributed smart meter data in order to predict household-level energy demand [202] and to estimate net consumption in residential solar photovoltaic systems [203], providing utilities with actionable insights for anticipating both local generation and demand patterns. Research efforts focused on household profiling have similarly employed FL to capture behavioral and structural characteristics of dwellings. For instance, the authors of [204] proposed an unsupervised clustering method to group customers exhibiting similar consumption patterns, while other studies have focused on inferring dwelling characteristics, appliance efficiency, or broader sociodemographic attributes from smart meter traces [205]. Distinct from these prior contributions, the present study seeks to infer fine-grained information regarding household composition, moving beyond aggregate or high-level profiling to provide a more detailed understanding of the internal structure and characteristics of individual households.

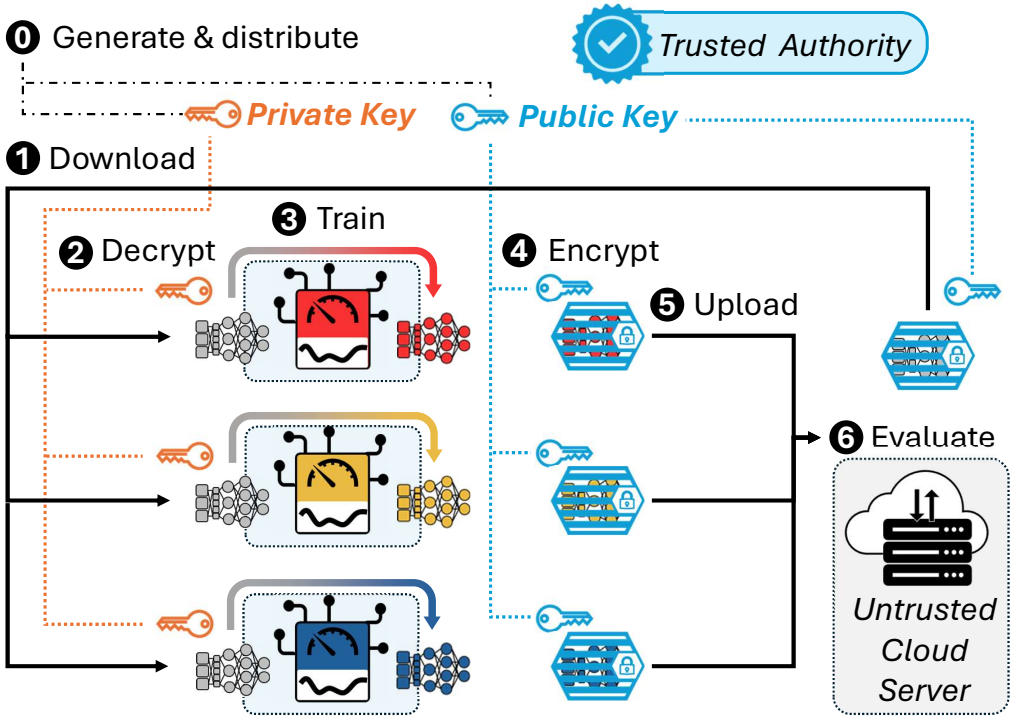


Fig. 6.4 Federated Learning with Homomorphic Encryption for smart meters data.

Privacy-preserving FL with Homomorphic Encryption (HE-based ppFL)

Although FL preserves data locality by ensuring that raw datasets remain with the respective participants, this property alone is insufficient to guarantee full privacy protection. Adversarial entities, including honest-but-curious servers, can exploit vulnerabilities in the protocol by performing attacks such as gradient inversion [88, 206], which reconstructs sensitive information from shared model updates. Consequently, additional privacy-preserving mechanisms are required to mitigate such threats and strengthen the overall security of FL systems.

In this work, we adopt the CKKS scheme as implemented in the open-source TenSEAL library [100], which provides native support for batching, a strong optimization within this context. Further optimizations can also be integrated at different stages of the FL stack, including efficient key generation [101], partial encryption strategies [207], and selective synchronization protocols [119], all of which aim to mitigate the performance bottlenecks inherent in privacy-preserving federated systems.

Our objective is to develop an accurate classifier to infer household composition from electricity consumption profiles. To enable privacy-preserving collaboration among energy utilities, we adopt the algorithm introduced in the background section 3 based on the HE-based FL paradigm, with the only difference being the adoption of full participation. The schematic overview of this HE-FL architecture is depicted in Fig. 6.4.

6.2.3 Training Framework

Dataset Description

Comprehensive datasets that combine electricity consumption records with detailed information on household composition are generally difficult to obtain. Even when such datasets are publicly available, they are often too limited in size to support rigorous testing and validation of machine-learning algorithms. As a result, researchers tend to rely on the same few benchmark datasets, which restricts the scope of comparative evaluations and may hinder the generalizability of findings. Synthetic data offers an effective solution to this limitation, as it allows for the controlled generation of arbitrarily large datasets tailored to specific testing requirements. Beyond scalability, synthetic datasets also enable researchers to manage critical aspects such as class balance and population diversity, thereby facilitating more thorough and systematic assessments of algorithmic performance. In this work, we employ a synthetic dataset produced using the agent-based simulation framework introduced in [192, 193]. This simulator is designed to reproduce realistic residential electricity consumption patterns based on data sources widely available across countries, namely Time-Use Surveys (TUS) [208]. TUS datasets record, at 10-minute resolution, the daily activities of representative population samples, including behaviors such as cooking, sleeping, or commuting. To translate these activity patterns into household electricity usage, the simulator incorporates two categories of agents: (i) user agents, which model the activity schedules of individuals through semi-Markov processes [209], and (ii) household agents, which convert activities into corresponding appliance-level energy consumption. User agents are parameterized by gender and employment status, attributes known to influence the amount of time individuals spend at home and the nature of their activities, and thus their electricity demand [210]. Employment status categories include full-time and part-time workers, students, unemployed or

retired individuals, children, and homemakers. For the present study, we adopt the Italian Time-Use Survey [211] as the input to the simulator, enabling the generation of a dataset that reflects the residential electricity consumption of Italian households. The resulting dataset comprises 1,000 synthetic families and their corresponding energy consumption profiles over an entire year. While the simulator natively outputs load data at 10-minute resolution, the series were subsequently resampled to an hourly resolution in order to align with the analytical requirements of this work.

According to the *Istituto Nazionale di Statistica* (ISTAT), Italian household structures have undergone notable demographic changes in the past two decades. The share of single-person households has steadily increased and now accounts for more than 36% of all families, whereas large households with five or more members represent less than 5% of the total [212]. To align with these demographic patterns, and for testing purposes, the dataset used in this study was restricted to families with up to four members. Within this framework, the following household compositions and proportions were defined:

- Single-person households: full-time worker (10%), part-time worker (5%), student (5%), and retired individual (5%).
- Two-person households: retired couple (5%), student couple (5%), two full-time workers (5%), full-time worker with homemaker (5%), and full-time worker with child (5%).
- Three-person households: (i) two full-time workers with a child (10%), (ii) single parent with two children (5%), (iii) full-time worker, part-time worker, and one child (5%), (iv) full-time worker, part-time worker, and an unemployed/retired member (5%).
- Four-person households: (i) two full-time workers with two children (5%), (ii) full-time worker, part-time worker, and two children (10%), (iii) full-time worker, homemaker, student, and one child (5%), (iv) two part-time workers, a student, and one child (5%).

Although these household types represent only a subset of all possible configurations, they were selected to capture the principal variations in terms of the time availability of household members and the presence of children. These two factors

are particularly relevant, as they constitute key determinants of electricity consumption patterns. In fact, they strongly influence the extent to which households are able and willing to modify their consumption behavior in response to external signals, such as demand-response requests [213] or dynamic pricing schemes [214].

6.2.4 Experimental Setup & Results

Data Partitioning

The synthetic dataset constructed for this study comprises 17 distinct household compositions, which serve as the target classes for household characteristic identification. For evaluation purposes, the dataset was divided into training and test sets. Specifically, five households were randomly selected from each of the 17 classes to form a balanced test set, while the remaining households were reserved for training. To reproduce a realistic distributed learning environment, the training data was further partitioned among 10 fictitious energy utility companies. The partitioning strategy followed the procedure outlined in [127], whereby households were assigned to utilities according to class ratios sampled from a Dirichlet distribution with a concentration parameter of 0.3. This approach produces statistically heterogeneous partitions, resulting in utility-specific datasets that vary in both size and class composition. Such a configuration closely mirrors real-world scenarios, where local utilities manage customer bases with diverse demographic and geographic characteristics.

Training Scenarios

To assess the performance of the proposed approach, we considered three distinct training scenarios:

1. *Siloed Training*: a baseline configuration where each energy utility trains a model exclusively on its local dataset. This scenario highlights the drawbacks of isolated learning and serves as a reference for quantifying the improvements gained through collaborative approaches.

2. *Centralized Training*: an idealized setting in which all data is collected into a single database for conventional training. While unrealistic in practice due to

privacy and regulatory restrictions on data sharing, this scenario provides an upper bound on achievable performance.

3. *Privacy-Preserving FL*: a federated learning setup enhanced with homomorphic encryption, enabling utilities to train a model while preserving data confidentiality jointly.

Preprocessing, Model & Training Hyperparameters

Before training, a preprocessing pipeline was applied to organize hourly consumption time series. Data was segmented into non-overlapping monthly windows and reshaped into tensors with four channels, each representing one week of consumption. This design allows the model to capture both short-term and long-term patterns while maintaining computational efficiency. The predictive model consists of a sequential 1D-CNN with two convolutional layers followed by two fully connected layers. The first convolutional layer employs 32 filters with a kernel size of 7, and the second uses 64 filters with a kernel size of 3. Each convolution is followed by a ReLU activation function and a max-pooling operation with a kernel size and stride of 2. The output feature maps are then passed through a fully connected layer, producing a 512-dimensional representation with ReLU activation, which is subsequently fed into the final classification layer. The same architecture and preprocessing pipeline were adopted consistently across all training scenarios.

With respect to training hyperparameters, the same configuration was adopted for both *Siloed Training* and *Centralized Training*. In these scenarios, models were trained for 20,000 iterations using SGD with a learning rate of 0.01, Nesterov momentum set to 0.9, and a batch size of 64. In the case of *HE-based ppFL*, training encompassed 1000 synchronization rounds, where each party performed 10 local training iterations per round, under the same optimizer settings as the baseline scenarios. The HE implementation employed the CKKS scheme from the TenSEAL library, configured with a polynomial modulus degree of 8192, a 40-bit scale, and a 200-bit coefficient modulus.

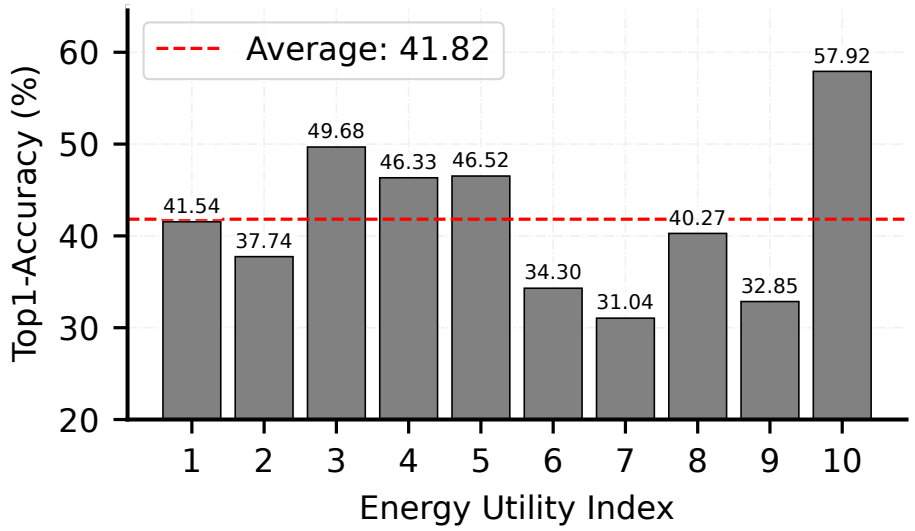


Fig. 6.5 Test accuracy over ten energy utilities in Siloed Training.

Results

Figure 6.5 presents the classification accuracy achieved in the *Siloed Training* scenario, evaluated on the test set for each energy utility (indexed from 0 to 9). The results highlight two main findings. First, there is a pronounced variability in performance across utilities, with accuracy values ranging from 31.04% to 57.92%. This disparity reflects the statistical heterogeneity of the local datasets, which differ in size, class coverage, and distribution. In practice, the quality and representativeness of local data strongly determine the effectiveness of the trained models, making performance highly dependent on the specific data available to each utility. Second, the overall accuracy levels remain unsatisfactory for practical deployment. Even the best-performing utility achieves only 57.92%, indicating a limited ability to generalize across the diverse sociodemographic profiles and consumption behaviors represented in the test set. Together, these results underscore the inherent limitations of *Siloed Training*, where the absence of cross-utility collaboration prevents models from reaching adequate classification quality.

A comparison across the three training strategies further highlights the limitations of *Siloed Training*. As reported in Table 6.10, this setting yields an average classification accuracy of only 41.82%, confirming the poor generalization ability of models trained on fragmented and statistically heterogeneous datasets. By contrast,

Table 6.10 Household characterization identification accuracy under three training scenarios.

Training Scenario	Test Accuracy
Siloed Training	41.82%*
Centralized Training	81.09%
HE-based ppFL	77.92%

* Average accuracy computed over the 10 energy utilities.

the *Centralized Training* scenario achieves a substantially higher accuracy of 81.09%, representing an unattainable upper bound in practice, as strict data protection regulations prohibit the centralization of raw consumption data across organizational boundaries. Within this context, *Privacy-Preserving FL* emerges as a compelling alternative. It attains 77.92% accuracy, representing just 3.17 percentage points below the centralized benchmark, while enabling secure collaboration through data locality and homomorphic encryption. These findings demonstrate that FL can deliver high-quality sociodemographic classification in compliance with regulatory constraints, thus reconciling accuracy with data sovereignty.

6.2.5 Discussion and Final Remarks

This work introduced a training framework for privacy-preserving household characterization from electricity consumption data. The proposed approach combines two main contributions: (*i*) a synthetic data generation pipeline capable of capturing fine-grained sociodemographic patterns in household energy consumption, and (*ii*) a federated learning framework leveraging homomorphic encryption to enable secure collaborative training. Our experiments show that siloed training yields poor and inconsistent performance across utilities, highlighting its limitations in real-world applications. On the other hand, federated learning achieves accuracy levels approaching those of centralized training, demonstrating its effectiveness in balancing model quality with strict data privacy requirements. By bridging the gap between predictive performance and regulatory compliance, this study provides a robust foundation for advancing private, collaborative, and data-driven intelligence in distributed energy systems.

Chapter 7

Conclusions

The technological breakthroughs across hardware, software, and synergic optimizations have progressively enabled inference at the edge of the IoT. Processing high-dimensional data at the edge promotes real-time, privacy-preserving, and efficient intelligent services. However, the at-scale adoption of edge intelligence is currently limited by the complexity of training deployable models, primarily due to the challenges of aggregating decentralized sensor data. Federated Learning offers a promising avenue to bridge the extremes generated by decentralized learning, where each node can extract knowledge from its own data only, and centralized learning, which faces privacy and data management challenges preventing the aggregation of sensor data. In cross-device FL, fleets of edge devices collaborate in training a single shared model, with no need for sharing raw data, thereby enhancing privacy.

7.1 Summary of the Contributions

This thesis has advanced the adoption of Federated Learning in cross-device environments by addressing fundamental challenges in *communication efficiency*, *energy efficiency*, and *security*. The contributions presented throughout this work provide convergence-aware and budget-aware strategies that move FL closer to real-world deployment at scale.

Communication efficiency This work presented different strategies to promote the communication efficiency of FL, investigating a gradient-based and a budget-

aware approach. It introduced a layer-freezing technique that progressively excludes stable layers from synchronization. This strategy enabled savings of up to 84% in communication volume with negligible accuracy loss, avoiding client-side overhead. In addition, it presented a dynamic resource management strategy that tackles the limited data bundles of the involved devices, insufficient to achieve competitive accuracy when facing extreme label skew. The novel budget management strategy enhances the resulting model quality across multiple constraints, achieving up to +12.20% accuracy compared to state-of-the-art solutions.

Energy efficiency This thesis further investigated the issue of extreme label skew and presented a novel adaptive concurrency management strategy. The proposed approach manages the number of participating clients as training proceeds, as a function of the model training status. By leveraging a gradient feedback signal, it detects learning stagnation phases, during which the model struggles to obtain sufficient convergence. To escape such phases, it increases participation, thereby improving label representation at the aggregation stage and enhancing the fidelity compared to the global data distribution. This strategy enhances energy efficiency by up to +2.74× compared to state-of-the-art techniques.

Security and Privacy To lower the adoption barrier of homomorphic encryption in FL, this thesis introduced a tensor-freezing mechanism that reduced the computational and communication overhead caused by encryption, an orthogonal knob advancing the feasibility of HE-based privacy-preserving FL in cross-device settings. Removing converged tensors enables a reduction of up to 37.4% in data volume and 35.5% less compute time on the client side (36.4% on the server side). In addition, it identified a previously undetected deployment-time vulnerability that exploits CPU frequency traces when the service is strategically queried. The discovered side-channel attack is capable of fingerprinting neural network architectures in edge inference services. Together, these contributions highlight both the potential of stronger training-stage privacy mechanisms and the risks posed by deployment-stage threats, emphasizing that the security and IP protection of intelligent services in distributed environments must be addressed end-to-end.

Evaluation To further promote the practical adoption of FL, it investigated a pre-deployment evaluation methodology for assessing FL strategies under diverse budget constraints and operational scenarios and revisited the role of the learning rate, traditionally limited to initial value tuning. By extending the hyperparameter search space to include both client- and server-side learning rate decay schedules, this work advanced fairness in algorithmic comparisons and revealed previously overlooked benefits. Finally, it presented an application-oriented case study to evaluate the feasibility of privacy-preserving FL with HE for fine-grained sociodemographic attribute classification from energy traces, demonstrating accuracy closely aligned with centralized settings while ensuring confidentiality.

In summary, this thesis contributes a coherent set of methods and strategies targeting multiple fronts that close the gap for federated learning deployments in cross-device settings, providing more *communication-efficient, resource-aware, and secure* solutions. By addressing in a holistic perspective the system bottlenecks, client heterogeneity, and protection of both models and data, the proposed solutions bring FL closer to large-scale deployment in IoT environments. These results push the frontiers of FL applicability, paving the way toward a more *sustainable, robust, and private decentralized machine learning*, therefore advancing distributed intelligence services at scale.

7.2 Future Work

Immediate future research will extend to broader experimental and theoretical analysis of federated learning in heterogeneous IoT environments, extending the optimizations to additionally tackle heterogeneous resources. The analysis will address the compounded effects of both data heterogeneity and system heterogeneity, i.e., extending this work to account for different device capabilities and the network variability, while considering additional constraints such as edge devices' memory. Further research will explore the unique challenges of IoT scenarios, such as sensors capturing diverse data complexities and heterogeneous sensing capabilities, as well as the impracticalities of manual data annotation across decentralized sources. Finally, a major focus will target the overhead introduced by privacy-preserving strategies such as homomorphic encryption to reduce the current adoption challenges.

Future directions Beyond the immediate scope, additional research will explore alternative privacy-preserving and efficient distributed learning strategies, as well as other security concerns raised by model and data poisoning attacks. The outcomes will further accelerate the path towards ubiquitous, continuously learning, AIoT services. Crucially, it will also address a fundamental bottleneck: the automation of hardware design to boost the development of more efficient, compact, and powerful edge computing platforms. Privacy-preserving learning solutions offer a promising solution to foster the widespread creation of novel architectures. This co-evolution establishes a virtuous cycle: improvements in learning drive the demand for smarter, more specialized hardware, which in turn enables richer and more efficient learning capabilities. Together, these synergic advances pave the way toward a *self-sustaining* ecosystem of intelligent, privacy-conscious, and high-performance AIoT systems.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021.

[5] Colby R. Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. In *MLSys*. mlsys.org, 2021.

[6] Shriyank Somvanshi, Md. Monzurul Islam, Gaurab Chhetri, Rohit Chakraborty, Mahmuda Sultana Mimi, Sawgat Ahmed Shuvo, Kazi Sifatul Islam, Syed Aaqib Javed, Sharif Ahmed Rafat, Anandi Dutta, and Subasish Das. From tiny machine learning to tiny deep learning: A survey. *CoRR*, abs/2506.18927, 2025.

[7] Guanghan Wu, Sasu Tarkoma, and Roberto Morabito. Consolidating tinyml lifecycle with large language models: Reality, illusion, or opportunity? *CoRR*, abs/2501.12420, 2025.

[8] Statista. Number of connected iot devices worldwide, 2024. Accessed: 2025-08-31.

[9] IoT Analytics. Number of connected iot devices to grow to 18.8 billion by end of 2024. State of IoT Summer 2024: Market Assessment and Global Forecast, September 2024. Accessed: 2025-08-31.

[10] Jinesh. How many iot devices are there in 2025? [all you need to know]. https://autobitslabs.com/how-many-iot-devices-are-there/, June 2025. Accessed: 2025-09-03.

[11] Arm. Ethos-u55. https://developer.arm.com/Processors/Ethos-U55. Accessed: 2025-09-03.

[12] Valentino Peluso and Andrea Calimera. Weak-mac: Arithmetic relaxation for dynamic energy-accuracy scaling in convnets. In *Int. Symp. on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

[13] Valentino Peluso, Roberto Giorgio Rizzo, Antonio Cipolletta, and Andrea Calimera. Inference on the edge: Performance analysis of an image classification task using off-the-shelf cpus and open-source convnets. In *6th Int. Conf. on Social Networks Analysis, Management and Security (SNAMS)*, pages 454–459, 2019.

[14] Matteo Grimaldi, Valentino Peluso, and Andrea Calimera. Optimality assessment of memory-bounded convnets deployed on resource-constrained RISC cores. *IEEE Access*, 7:152599–152611, 2019.

[15] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. Hasco: Towards agile hardware and software co-design for tensor computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1055–1068. IEEE, 2021.

[16] Paola Vitolo, Antonio De Vita, Luigi Di Benedetto, Danilo Pau, and Gian Domenico Licciardo. Low-power detection and classification for in-sensor predictive maintenance based on vibration monitoring. *IEEE Sensors Journal*, 22(7):6942–6951, 2022.

[17] Thorir Mar Ingolfsson, Andrea Cossettini, Xiaying Wang, Enrico Tabanelli, Giuseppe Tagliavini, Philippe Ryvlin, Luca Benini, and Simone Benatti. Towards long-term non-invasive monitoring for epilepsy via wearable eeg devices. In *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 01–04. IEEE, 2021.

[18] Christian Geckeler, Steffen Kirchgeorg, Georg Strunck, Frederik Bendix Thostrup, Florencia Sangermano, Andrea Desiderato, Martina Lüthi, Meret Jucker, Mailyn Adriana Gonzalez Herrera, Nicolás D Franco-Sierra, et al. Field deployment of biodivx drones in the amazon rainforest for biodiversity monitoring. *IEEE Transactions on Field Robotics*, 2025.

[19] Xu Liu, Taha Aksu, Juncheng Liu, Qingsong Wen, Yuxuan Liang, Caiming Xiong, Silvio Savarese, Doyen Sahoo, Junnan Li, and Chenghao Liu. Empowering time series analysis with synthetic data: A survey and outlook in the era of foundation models. *CoRR*, 2025.

[20] Tong Xia, Abhirup Ghosh, Xinchi Qiu, and Cecilia Mascolo. Flea: Addressing data scarcity and label skew in federated learning via privacy-preserving feature augmentation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3484–3494, 2024.

[21] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)* [37], pages 1273–1282.

[22] Herbert Woisetschläger, Alexander Erben, Bill Marino, Shiqiang Wang, Nicholas D. Lane, Ruben Mayer, and Hans-Arno Jacobsen. Federated learning priorities under the european union artificial intelligence act. *arXiv:2402.05968*, 2024.

[23] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

[24] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *CoRR*, abs/1811.03604, 2018.

[25] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *CoRR*, abs/1906.04329, 2019.

[26] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *ICASSP*, 2019.

[27] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Federated visual classification with real-world data distribution. In *ECCV*, 2020.

[28] Lars Wulfert, Christian Wiede, and Anton Grabmaier. Tinyfl: On-device training, communication and aggregation on a microcontroller for federated learning. In *NEWCAS*. IEEE, 2023.

[29] Hu Ding, Yan Yan, Yang Lu, Jing-Hao Xue, and Hanzi Wang. Uncertainty-aware label refinement on hypergraphs for personalized federated facial expression recognition. *IEEE Trans. Circuits Syst. Video Technol.*, 35(5):4675–4685, 2025.

[30] Lei Zhang, Guanyu Gao, and Huaizheng Zhang. Spatial-temporal federated learning for lifelong person re-identification on distributed edges. *IEEE Trans. Circuits Syst. Video Technol.*, 35(2):1884–1896, 2025.

[31] Xiao-Dong Xie, Yu-Wei Zhan, Zhen-Xiang Ma, Hong-Mei Liu, Zhen-Duo Chen, Xin Luo, and Xin-Shun Xu. Distributed learning for privacy-preserving semi-supervised video anomaly detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 2025.

[32] Ting Yang, Xiangwei Feng, Shaotang Cai, Yuqing Niu, and Haibo Pen. A privacy-preserving federated reinforcement learning method for multiple virtual power plants scheduling. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 72(4):1939–1950, 2025.

[33] Marco Bornstein, Nawaf Nazir, Ján Drgona, Soumya Kundu, and Veronica Adetola. Finding MIDDLE ground: Scalable and secure distributed learning. In *CIKM*, 2024.

[34] Ashkan Yousefpour, Shen Guo, Ashish Shenoy, Sayan Ghosh, Pierre Stock, Kiwan Maeng, Schalk-Willem Krüger, Michael G. Rabbat, Carole-Jean Wu, and Ilya Mironov. Green federated learning. *arXiv:2303.14604*, 2023.

[35] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for federated learning. In *Proc. Int. Conf. Mach. Learn., ICML.*, 2020.

[36] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *ICLR*, 2021.

[37] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, volume 54, pages 1273–1282, 2017.

[38] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Agüera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas N. Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horváth, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konečný, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtárik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake E. Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. A field guide to federated optimization. *arXiv:2107.06917*, 2021.

[39] Mahdi Morafah, Weijia Wang, and Bill Lin. A practical recipe for federated learning under statistical heterogeneity experimental design. *IEEE Trans. Artif. Intell.*, 5(4):1708–1717, 2024.

[40] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Commun. Surv. Tutorials*, 22(3):2031–2063, 2020.

[41] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE Trans. Neural Networks Learn. Syst.*, 31(10):4229–4238, 2020.

[42] Sunwoo Lee, Tuo Zhang, Chaoyang He, and Salman Avestimehr. Layer-wise adaptive model aggregation for scalable federated learning. *CoRR*, abs/2110.10302, 2021.

[43] Zheqi Zhu, Yuchen Shi, Jiajun Luo, Fei Wang, Chenghui Peng, Pingyi Fan, and Khaled B. Letaief. Fedlp: Layer-wise pruning mechanism for communication-computation efficient federated learning. *CoRR*, abs/2303.06360, 2023.

[44] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. Synchronize only the immature parameters: Communication-efficient federated learning by freezing parameters adaptively. *IEEE Trans. Parallel Distrib. Syst.*, pages 1–18, 2023.

[45] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[46] Jed Mills, Jia Hu, and Geyong Min. Faster federated learning with decaying number of local sgd steps. *IEEE Transactions on Parallel and Distributed Systems*, 34(7):2198–2207, 2023.

[47] Wang Luping, Wang Wei, and LI Bo. Cmfl: Mitigating communication overhead for federated learning. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*, pages 954–964. IEEE, 2019.

[48] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE Trans. Neural Netw. Learn. Syst.*, 31(9):3400–3413, 2020.

[49] Lucas Grativol, Mathieu Léonardon, Guillaume Muller, Virginie Fresse, and Matthieu Arzel. Federated learning compression designed for lightweight communications. In *ICECS*, 2023.

[50] Runmeng Du, Daojing He, Zikang Ding, Miao Wang, Sammy Chan, and Xuru Li. GSASG: global sparsification with adaptive aggregated stochastic gradients for communication-efficient federated learning. *IEEE Internet Things J.*, 11(17):28253–28266, 2024.

[51] Alexander Herzog, Robbie Southam, Othmane Belarbi, Saif Anwar, Marcello Bullo, Pietro Carnelli, and Aftab Khan. Selective updates and adaptive masking for communication-efficient federated learning. *IEEE Trans. Green Commun. Netw.*, 8(2):852–864, 2024.

[52] Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(3):1162–1176, 2020.

[53] Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. Ternary compression for communication-efficient federated learning. *IEEE Trans. Neural Networks Learn. Syst.*, 2022.

[54] Jed Mills, Jia Hu, and Geyong Min. Communication-efficient federated learning for wireless edge intelligence in iot. *IEEE Internet of Things Journal*, 7(7):5986–5994, 2019.

[55] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Fedopt: towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10(8):2864, 2020.

[56] Jinliang Yuan, Shangguang Wang, Hongyu Li, Daliang Xu, Yuanchun Li, Mengwei Xu, and Xuanzhe Liu. Towards energy-efficient federated learning via int8-based training on mobile dsps. In *WWW*, 2024.

[57] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 429–450, 2020.

[58] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2020.

[59] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Conf. on Computer Vision and Pattern Recognition (CVPR), 2021*, pages 10713–10722, 2021.

[60] Xutong Mu, Yulong Shen, Ke Cheng, Xueli Geng, Jiaxuan Fu, Tao Zhang, and Zhiwei Zhang. Fedproc: Prototypical contrastive federated learning on non-iid data. *Future Gener. Comput. Syst.*, 143:93–104, 2023.

[61] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *8th Int. Conf. on Learning Representations (ICLR)*, 2020.

[62] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[63] Manzil Zaheer, Sashank J. Reddi, Devendra Singh Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9815–9825, 2018.

[64] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd Int. Conf. on Learning Representations (ICLR)*, 2015.

[65] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. A survey on federated learning for resource-constrained iot devices. *IEEE Internet Things J.*, 2022.

[66] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. Synchronize only the immature parameters: Communication-efficient federated learning by freezing parameters adaptively. *IEEE Trans. Parallel Distributed Syst.*, 2024.

[67] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. Communication-efficient federated learning with gradual layer freezing. *IEEE Embed. Syst. Lett.*, 15(1):25–28, 2023.

[68] Erich Malan, Valentino Peluso, Andrea Calimera, Enrico Macii, and Paolo Montuschi. Automatic layer freezing for communication efficiency in cross-device federated learning. *IEEE Internet Things J.*, 2024.

[69] Thai Vu Nguyen, Long Bao Le, and Anderson Avila. Automatic structured pruning for efficient architecture in federated learning. *CoRR*, abs/2411.01759, 2024.

[70] Qimei Chen, Han Cheng, Yipeng Liang, Guangxu Zhu, Ming Li, and Hao Jiang. Tinyfel: Communication, computation, and memory efficient tiny federated edge learning via model sparse update. *IEEE Internet Things J.*, 2024.

[71] Xin-Chun Li and De-Chuan Zhan. Fedrs: Federated learning with restricted softmax for label distribution non-iid data. In *KDD*, 2021.

[72] Jie Zhang, Zhiqi Li, Bo Li, Jianghe Xu, Shuang Wu, Shouhong Ding, and Chao Wu. Federated learning with label distribution skew via logits calibration. In *ICML*, 2022.

[73] Ziqing Fan, Ruipeng Zhang, Jiangchao Yao, Bo Han, Ya Zhang, and Yanfeng Wang. Federated learning with bilateral curation for partially class-disjoint data. In *NeurIPS*, 2023.

[74] Yujun Shi, Jian Liang, Wenqing Zhang, Vincent Y. F. Tan, and Song Bai. Towards understanding and mitigating dimensional collapse in heterogeneous federated learning. In *ICLR*, 2023.

[75] Haozhao Wang, Peirong Zheng, Xingshuo Han, Wenchao Xu, Ruixuan Li, and Tianwei Zhang. Fednlr: Federated learning with neuron-wise learning rates. In *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, pages 3069–3080, Barcelona, Spain, 2024.

[76] Gihun Lee, Minchan Jeong, Yongjin Shin, Sangmin Bae, and Se-Young Yun. Preservation of the global knowledge by not-true distillation in federated learning. In *NeurIPS*, 2022.

[77] Jianghu Lu, Shikun Li, Kexin Bao, Pengju Wang, Zhenxing Qian, and Shiming Ge. Federated learning with label-masking distillation. In *MM*. ACM, 2023.

[78] Di Yu, Xin Du, Linshan Jiang, Shunwen Bai, Wentao Tong, and Shuiguang Deng. Fedlec: Effective federated learning algorithm with spiking neural networks under label skews. *arXiv:2412.17305*, 2024.

[79] Kuangpu Guo, Yuhe Ding, Jian Liang, Zilei Wang, Ran He, and Tieniu Tan. Exploring vacant classes in label-skewed federated learning. In *AAAI*, 2025.

[80] Haoran Xu, Jiaze Li, Wanyi Wu, and Hao Ren. Federated learning with sample-level client drift mitigation. In *Proc. Conf. Artif. Intell. (AAAI)*, pages 21752–21760, Apr. 2025.

[81] Kyle Sang, Tahseen Rabbani, and Furong Huang. Balancing label imbalance in federated environments using only mixup and artificially-labeled noise. *arXiv:2409.13235*, 2024.

[82] Saheed A. Tijani, Xingjun Ma, Ran Zhang, Frank Jiang, and Robin Doss. Federated learning with extreme label skew: A data extension approach. In *IJCNN*, 2021.

[83] Yanbiao Ma, Wei Dai, Wenke Huang, and Jiayi Chen. Geometric knowledge-guided localized global distribution alignment for federated learning. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 20958–20968, Jun. 2025.

[84] Jian Xu, Meilin Yang, Wenbo Ding, and Shao-Lun Huang. Stabilizing and improving federated learning with highly non-iid data and client dropout. *Appl. Intell.*, 55(3):216, 2025.

[85] Yiqun Diao, Qinbin Li, and Bingsheng He. Exploiting label skews in federated learning with model concatenation. In *AAAI*, 2024.

[86] Qiao Ketu, Wang Yi, Wang Baoquan, Luo Zhengdong, and Zhou Xi. Fedse: Group-based sequential training strategies for mitigating label skew in federated learning. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process (ICASSP)*, pages 1–5, Hyderabad, India, 2025.

[87] Nguyen Nang Hung, Truong Thao Nguyen, Trong Nghia Hoang, Hieu H Pham, Thanh Hung Nguyen, and Nguyen Phi Le. Safa: Handling sparse and scarce data in federated learning with accumulative learning. *IEEE Trans. Comput.*, 2025.

[88] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients - how easy is it to break privacy in federated learning? In *NeurIPS 2020*, 2020.

[89] Jiahui Geng, Yongli Mou, Qing Li, Feifei Li, Oya Beyan, Stefan Decker, and Chunming Rong. Improved gradient inversion attacks and defenses in federated learning. *IEEE Trans. Big Data*, 2023.

[90] Holger R Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, YuanTing Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona G Flores, and Andrew Feng. NVIDIA FLARE: Federated learning from simulation to real-world. In *FL-NeurIPS*, 2022.

[91] Zhifeng Jiang, Wei Wang, and Yang Liu. FLASHE: additively symmetric homomorphic encryption for cross-silo federated learning. *arXiv:2109.00675*, 2021.

[92] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, Théo Ryffel, Zarreen Naowal Reza, and Georgios Kaissis. *PySyft: A Library for Easy Federated Learning*, pages 111–139. Springer International Publishing, 2021.

[93] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Jeremy Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, Mark Purcell, Ambrish Rawat, Tran Ngoc Minh, Naoise Holohan, Supriyo Chakraborty, Shalisha Witherspoon, Dean Steuer, Laura Wynter, Hifaz Hassan, Sean Laguna, Mikhail Yurochkin, Mayank Agarwal, Ebube Chuba, and Annie Abay. IBM federated learning: an enterprise framework white paper V0.1. *arXiv:2007.10987*, 2020.

[94] Weizhao Jin, Yuhang Yao, Shanshan Han, Carlee Joe-Wong, Srivatsan Ravi, Salman Avestimehr, and Chaoyang He. FedML-HE: An efficient homomorphic-encryption-based privacy-preserving federated learning system. In *FL@FM-NeurIPS*, 2023.

[95] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[96] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

[97] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *ATC*, 2020.

[98] Pan Yao, He Wang, Chao Zheng, Jing Yang, and Liming Wang. Efficient federated learning aggregation protocol using approximate homomorphic encryption. In *CSCWD*, 2023.

[99] Junhao Han and Li Yan. Adaptive batch homomorphic encryption for joint federated learning in cross-device scenarios. *IEEE Internet Things J.*, 2024.

[100] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption. In *DPML ICLR Workshop*, 2021.

[101] Chunrong He, Guiyan Liu, Songtao Guo, and Yuanyuan Yang. Privacy-preserving and low-latency federated learning in edge computing. *IEEE Internet Things J.*, 2022.

[102] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Agüera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas N. Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horváth, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konečný, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtárik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake E. Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. A field guide to federated optimization. *CoRR*, abs/2107.06917, 2021.

[103] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. Draft & refine: Efficient resource management in federated learning under pathological labels skew. In *ICECS*, 2024.

[104] Kelam Goutam, S. Balasubramanian, Darshan Gera, and R. Raghunatha Sarma. Layerout: Freezing layers in deep neural networks. *SN Comput. Sci.*, 1(5):295, 2020.

[105] Yuki Miyauchi, Haruki Mori, Tetsuya Youkawa, Kazuki Yamada, Shintato Izumi, Masahiko Yoshimoto, Hiroshi Kawaguchi, and Atsuki Inoue. Layer skip learning using LARS variables for 39% faster conversion time and lower bandwidth. In *25th Int. Conf. on Electronics, Circuits and Systems (ICECS)*, pages 673–676, 2018.

[106] Pooneh Safayenikoo and Ismail Akturk. Weight update skipping: Reducing training time for artificial neural networks. *IEEE J. Emerg. Sel. Topics Circuits Syst.*, 11(4):563–574, 2021.

[107] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *30th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 6076–6085, 2017.

[108] Tien-Ju Yang, Dhruv Guliani, Françoise Beaufays, and Giovanni Motta. Partial variable training for efficient on-device federated learning. In *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP) 2022*, pages 4348–4352, 2022.

[109] Kilian Pfeiffer, Martin Rapp, Ramin Khalili, and Jörg Henkel. Coco-fl: Communication- and computation-aware federated learning via partial NN freezing and quantization. *CoRR*, abs/2203.05468, 2022.

[110] Hakim Sidahmed, Zheng Xu, Ankush Garg, Yuan Cao, and Mingqing Chen. Efficient and private federated learning with partially trainable networks. *CoRR*, abs/2110.03450, 2021.

[111] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[112] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations (ICLR)*, 2021.

[113] Beibei Li, Shang Ma, Ruilong Deng, Kim-Kwang Raymond Choo, and Jin Yang. Federated anomaly detection on system logs for the internet of things: A customizable and communication-efficient approach. *IEEE Trans. Netw. Serv. Manag.*, 2022.

[114] Qingming Li, Xiaohang Li, Li Zhou, and Xiaoran Yan. Adafl: Adaptive client selection and dynamic contribution evaluation for efficient federated learning. In *ICASSP*, 2024.

[115] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. Gradient-aware participation for energy reduction in federated learning with extreme label skew. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2025.

[116] Pian Qi, Diletta Chiaro, and Francesco Piccialli. Small models, big impact: A review on the power of lightweight federated learning. *Future Gener. Comput. Syst.*, 2025.

[117] Pian Qi, Diletta Chiaro, and Francesco Piccialli. FL-FD: federated learning-based fall detection with multimodal data fusion. *Inf. Fusion*, 2023.

[118] Xinchi Qiu, Titouan Parcollet, Javier Fernández-Marqués, Pedro P. B. de Gusmao, Yan Gao, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. A first look into the carbon footprint of federated learning. *J. Mach. Learn. Res.*, 24:129:1–129:23, 2023.

[119] Valentino Peluso, Erich Malan, Andrea Calimera, and Enrico Macii. Private tensor freezing for an efficient federated learning with homomorphic encryption. In *ICCD*, 2024.

[120] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. Enabling dvfs side-channel attacks for neural network fingerprinting in edge inference services. In *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2023.

[121] Brij B Gupta, Akshat Gaurav, and Varsha Arya. Secure and privacy-preserving decentralized federated learning for personalized recommendations in consumer electronics using blockchain and homomorphic encryption. *IEEE Trans. Consum. Electron.*, 2023.

[122] Neveen Mohammad Hijazi, Moayad Aloqaily, Mohsen Guizani, Bassem Ouni, and Fakhri Karray. Secure federated learning with fully homomorphic encryption for iot communications. *IEEE Internet Things J.*, 2024.

[123] Li Zhang, Jianbo Xu, Pandi Vijayakumar, Pradip Kumar Sharma, and Uttam Ghosh. Homomorphic encryption-based privacy-preserving federated learning in iot-enabled healthcare system. *IEEE Trans. Netw. Sci. Eng.*, 2023.

[124] Eric Appiah Mantey, Conghua Zhou, Joseph Henry Anajemba, John Kingsley Arthur, Yasir Hamid, Atif Chowhan, and Obinna Ogbonnia Otuu. Federated learning approach for secured medical recommendation in internet of medical things using homomorphic encryption. *IEEE J. Biomed. Health Inform.*, 2024.

[125] Ziyao Wang, Jianyu Wang, and Ang Li. Fedhyper: A universal and robust learning rate scheduler for federated learning with hypergradient descent. In *ICLR*. OpenReview.net, 2024.

[126] Divyansh Jhunjhunwala, Shiqiang Wang, and Gauri Joshi. Fedexp: Speeding up federated averaging via extrapolation. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2023.

[127] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *8th Int. Conf. on Learning Representations (ICLR)* [61].

[128] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[129] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[130] Valentino Peluso, Roberto Giorgio Rizzo, Antonio Cipolletta, and Andrea Calimera. Inference on the edge: Performance analysis of an image classification task using off-the-shelf cpus and open-source convnets. In *SNAMS*, 2019.

[131] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Trans. Dependable Secur. Comput.*, 19(6):4270–4284, 2022.

[132] Sayed Erfan Arefin and Abdul Serwadda. Deep neural exposure: You can run, but not hide your neural network architecture! In *IH&MMSec*, 2021.

[133] Kartik Patwari, Syed Mahbub Hafiz, Han Wang, Houman Homayoun, Zubair Shafiq, and Chen-Nee Chuah. DNN model architecture fingerprinting attack on CPU-GPU edge devices. In *EuroS&P*, 2022.

[134] Yuntao Liu and Ankur Srivastava. GANRED: gan-based reverse engineering of dnns via cache side-channel. In *CCSW*, 2020.

[135] Bhargav Achary Dandpati Kumar, R. Sai Chandra Teja, Sparsh Mittal, Biswabandan Panda, and C. Krishna Mohan. Inferring DNN layer-types through a hardware performance counters based side channel attack. In *AIMLSystems*, 2021.

[136] Vasisht Duddu, Debasis Samanta, D. Vijay Rao, and Valentina Emilia Balas. Stealing neural networks via timing side channels. *CoRR*, abs/1812.11720, 2018.

[137] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *USENIX Security Symposium*, 2020.

[138] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. SCA strikes back: Reverse-engineering neural network architectures using side channels. *IEEE Des. Test*, 39(4):7–14, 2022.

[139] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. Open DNN box by power side-channel attack. *IEEE Trans. Circuits Syst.*, 67-II(11):2717–2721, 2020.

[140] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Deepsniffer: A DNN model extraction framework based on learning architectural hints. In *ASPLOS*, 2020.

[141] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. Hermes attack: Steal DNN models with lossless inference accuracy. In *USENIX Security Symposium*, 2021.

[142] Aslan Büşra and Yilmazer-Metin Ayse. A study on power and energy measurement of nvidia jetson embedded gpus using built-in sensor. In *UBMK*, 2022.

[143] Linux Governors. Accessed on 2023/03/20.

[144] Debopriya Roy Dipta and Berk Gülmezoglu. DF-SCA: dynamic frequency side channel attacks are practical. In *ACSAC*, 2022.

[145] Nikhil Chawla, Arvind Singh, Monodeep Kar, and Saibal Mukhopadhyay. Application inference using machine learning based side channel analysis. In *IJCNN*, 2019.

[146] Chen Liu, Monodeep Kar, Xueyang Wang, Nikhil Chawla, Neer Roggel, Bilgiday Yuce, and Jason M. Fung. Methodology of assessing information leakage through software-accessible telemetries. In *HOST*, 2021.

[147] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim M. Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. Machine learning at facebook: Understanding inference at the edge. In *HPCA*, 2019.

[148] Mottaqiallah Taouil, Abdullah Aljuffri, and Said Hamdioui. Power side channel attacks: Where are we standing? In *DTIS*, 2021.

[149] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *ECCV*, 2020.

[150] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *KDD*, 2021.

[151] TensorFlow Hub. Accessed on 2023/03/20.

[152] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[153] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *ICCV*, 2019.

[154] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[155] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[156] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *MLSys*, 2019.

[157] Huanle Zhang, Lei Fu, Mi Zhang, Pengfei Hu, Xiuzhen Cheng, Prasant Mohapatra, and Xin Liu. Federated learning hyperparameter tuning from a system perspective. *IEEE Internet Things J.*, 10(16):14102–14113, 2023.

[158] Juan Marcelo Parra Ullauri, Xunzheng Zhang, Anderson Bravalheri, Reza Nejabati, and Dimitra Simeonidou. In *SAC*, pages 1209–1216. ACM, 2023.

[159] Zhen Wang, Weirui Kuang, Ce Zhang, Bolin Ding, and Yaliang Li. Fedhpobench: A benchmark suite for federated hyperparameter optimization. In *ICML*, pages 35908–35948. PMLR, 2023.

[160] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. Refined two-sided learning rate tuning for robust evaluation in federated learning. *IEEE Transactions on Artificial Intelligence*, 7(2):906–917, 2026.

[161] Erich Malan, Claudia De Vizia, Marco Castangia, Valentino Peluso, Andrea Calimera, and Enrico Macii. Privacy-preserving federated learning for household characteristic identification. In *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 2040–2045, 2025.

[162] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv:1909.06335*, 2019.

[163] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2021.

[164] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In Dhillon et al. [57], pages 429–450.

[165] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 965–978. IEEE, 2022.

[166] Saeed Vahidian, Mahdi Morafah, Chen Chen, Mubarak Shah, and Bill Lin. Rethinking data heterogeneity in federated learning: Introducing a new notion and standard benchmarks. *IEEE Trans. Artif. Intell.*, 5(3):1386–1397, 2024.

[167] Haibo Yang, Minghong Fang, and Jia Liu. Achieving linear speedup with partial worker participation in non-iid federated learning. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2021.

[168] Yujia Wang, Lu Lin, and Jinghui Chen. Communication-efficient adaptive federated learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *Proc. Int. Conf. Mach. Learn., ICML.*, volume 162, pages 22802–22838, 2022.

[169] Nikita Dhawan, Nicole Mitchell, Zachary Charles, Zachary Garrett, and Gintare Karolina Dziugaite. Leveraging function space aggregation for federated learning at scale. *Trans. Mach. Learn. Res.*, 2024, 2024.

[170] Yae Jee Cho, Divyansh Jhunjhunwala, Tian Li, Virginia Smith, and Gauri Joshi. Maximizing global model appeal in federated learning. *Trans. Mach. Learn. Res.*, 2024, 2024.

[171] Irene Tenison, Sai Aravind Sreeramadas, Vaikkunth Mugunthan, Edouard Oyallon, Irina Rish, and Eugene Belilovsky. Gradient masked averaging for federated learning. *Trans. Mach. Learn. Res.*, 2023, 2023.

[172] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2015.

[173] Yan Sun, Li Shen, Hao Sun, Liang Ding, and Dacheng Tao. Efficient federated learning via local adaptive amended optimizer with linear speedup. 2023.

[174] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2020.

[175] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2021.

[176] Kaiwei Mo, Chen Chen, Jiamin Li, Hong Xu, and Chun Jason Xue. Two-dimensional learning rate decay: Towards accurate federated learning with non-iid data. In *Proc. Int. Joint Conf. Neural Networks, IJCNN*, 2021.

[177] Ziyao Wang, Jianyu Wang, and Ang Li. Fedhyper: A universal and robust learning rate scheduler for federated learning with hypergradient descent. 2024.

[178] Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. In *Proc. Int. Conf. Learn. Representations, ICLR*, 2020.

[179] John Chen, Cameron R. Wolfe, and Tasos Kyrillidis. REX: revisiting budgeted training with an improved schedule. In *Proc. Mach. Learn. Syst. (MLSys)*, 2022.

[180] Zexin Wang, Weidong Zhang, Xuangou Wu, and Xiujun Wang. Matched averaging federated learning gesture recognition with wifi signals. In *Proc. 7th Int. Conf. Big Data Comput. Commun., BigCom*, 2021.

[181] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit., CVPR*, 2016.

[182] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The non-iid data quagmire of decentralized machine learning. In *Proc. Int. Conf. Mach. Learn., ICML.*, 2020.

[183] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Comput. Linguistics*, 19(2):313–330, 1993.

[184] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.

[185] Yuhao Zhou, Qing Ye, and Jiancheng Lv. Communication-efficient federated learning with compensated overlap-fedavg. *IEEE Trans. Parallel Distributed Syst.*, 33(1):192–205, 2022.

[186] Shuo Yuan, Bin Cao, Yao Sun, Zhiguo Wan, and Mugen Peng. Secure and efficient federated learning through layering and sharding blockchain. *IEEE Trans. Netw. Sci. Eng.*, 11(3):3120–3134, 2024.

[187] Christian Beckel, Leyna Sadamori, Thorsten Staake, and Silvia Santini. Revealing household characteristics from smart meter data. *Energy*, 2014.

[188] Y Kiguchi, M Weeks, and R Arakawa. Predicting winners and losers under time-of-use tariffs using smart meter data. *Energy*, 2021.

[189] Marco Castangia, Riccardo Sappa, Awet Abraha Girmay, Christian Camarda, Enrico Macii, and Edoardo Patti. Detection of anomalies in household appliances from disaggregated load consumption. In *2021 International Conference on Smart Energy Systems and Technologies (SEST)*, pages 1–6. IEEE, 2021.

[190] Victor von Loessl. Smart meter-related data privacy concerns and dynamic electricity tariffs: Evidence from a stated choice experiment. *Energy policy*, 2023.

[191] Xu Cheng, Chendan Li, and Xiufeng Liu. A review of federated learning in energy systems. In *I&CPS Asia)*, 2022.

[192] Claudia De Vizia, Daniele Salvatore Schiera, Alberto Macii, Edoardo Patti, and Lorenzo Bottaccioli. A simulation framework for urban electric mobility based on limited widespread data and spatial information. *IEEE Trans. Intell. Transp. Syst.*, 2024.

[193] Claudia De Vizia, Alberto Macii, Edoardo Patti, and Lorenzo Bottaccioli. A hierarchical and modular agent-oriented framework for power systems co-simulations. *Energy Informatics*, 2022.

[194] Mert Pekey, Yiğit Deniz Çelebi, Ceren Anıl, and Albert Levi. Private information inference of households from electricity consumption data. In *BalkanCom*, 2021.

[195] Abhishek Roshan and D Ganga. Intelligent categorization and interactive mechanism for smart demand side management of residential consumers. *Sustain. Energy Grids Netw.*, 2024.

[196] Jun Lin, Jin Ma, and Jian Guo Zhu. Estimation of household characteristics with uncertainties from smart meter data. *Int. J. Electr. Power Energy Syst.*, 2022.

[197] Hongliang Fang, Jiang-Wen Xiao, and Yan-Wu Wang. Self-training convolutional autoencoder for consumer characteristics identification with imbalance datasets. *Eng. Appl. Artif. Intell.*, 2023.

[198] Hanguan Wen, Xiufeng Liu, Ming Yang, Bo Lei, Cheng Xu, and Zhe Chen. A novel approach for identifying customer groups for personalized demand-side management services using household socio-demographic data. *Energy*, 2024.

[199] Rakesh Shrestha, Mohammadreza Mohammadi, Sima Sinaei, Alberto Salcines, David Pampliega, Raul Clemente, Ana Lourdes Sanz, Ehsan Nowroozi, and Anders Lindgren. Anomaly detection based on LSTM and autoencoders using federated learning in smart electric grid. *J. Parallel Distributed Comput.*, 2024.

[200] Mohamed Abdel-Basset, Nour Moustafa, and Hossam Hawash. Privacy-preserved generative network for trustworthy anomaly detection in smart grids: A federated semisupervised approach. *IEEE Trans. Ind. Informatics*, 2023.

[201] Mi Wen, Rong Xie, Kejie Lu, Liangliang Wang, and Kai Zhang. Feddetect: A novel privacy-preserving federated learning framework for energy theft detection in smart grid. *IEEE Internet Things J.*, 2022.

[202] Mohammad Navid Fekri, Katarina Grolinger, and Syed Mir. Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks. *Int. J. Electr. Power Energy Syst.*, 2022.

[203] Mahmoud M. Badr, Mohamed M. E. A. Mahmoud, Yuguang Fang, Mohammed J. Abdulaal, Abdulah Jeza Aljohani, Waleed Alasmary, and Mohamed I. Ibrahem. Privacy-preserving and communication-efficient energy prediction scheme based on federated learning for smart grids. *IEEE Internet Things J.*, 2023.

[204] Yi Wang, Mengshuo Jia, Ning Gao, Leandro Von Krannichfeldt, Mingyang Sun, and Gabriela Hug. Federated clustering for electricity consumption pattern extraction. *IEEE Trans. Smart Grid*, 2022.

[205] Weilong Chen, Shengrong Bu, Xinran Zhang, Yanqing Tao, Yanru Zhang, and Zhu Han. Semi-supervised federated analytics for heterogeneous household characteristics identification. *IEEE Trans. Smart Grid*, 2024.

[206] Caspar Meijer, Jiyue Huang, Shreshtha Sharma, Elena Lazovik, and Lydia Y. Chen. Ts-inverse: A gradient inversion attack tailored for federated time series forecasting models. In *SaTML*, pages 110–124. IEEE, 2025.

[207] Chenghao Hu and Baochun Li. Maskcrypt: Federated learning with selective homomorphic encryption. *IEEE Trans. Dependable Secur. Comput.*, 2025.

[208] Eurostat. Harmonised European Time Use Surveys (HETUS) 2018 Guidelines. Technical report, Publications Office of the European Union,, 2020.

[209] Jacques Janssen and Raimondo Manca. *Semi-Markov risk models for finance, insurance and reliability*. Springer Science & Business Media, 2007.

[210] Joshua D. Rhodes, Wesley J. Cole, Charles R. Upshaw, Thomas F. Edgar, and Michael E. Webber. Clustering analysis of residential electricity demand profiles. *Applied Energy*, 2014.

[211] ISTAT. Multiscopo sulle famiglie: uso del tempo, 2013.

[212] ISTAT. Annuario statistico italiano. Technical report, ISTAT, 2022.

[213] ARENA. Demand response customer insights report, 2018.

[214] Timur Yunusov and Jacopo Torriti. Distributional effects of time of use tariffs based on electricity demand and time use. *Energy Policy*, 2021.