TU WIEN Informatics

Navigating the Transition from Centralized to Federated Learning with Non-IID Data: A Human Activity Recognition Case Study

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Rastko Gajanin, BSc
Matrikelnummer 11930500

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dipl.-Ing. Dr.techn. Stefan Nastić, BSc
Mitwirkung: Univ.Ass. Dr. Andrea Morichetta
Anastasiya Danilenka, MSc

Wien, 20. August 2024

Rastko Gajanin Stefan Nastić

TU WIEN Informatics

Navigating the Transition from Centralized to Federated Learning with Non-IID Data: A Human Activity Recognition Case Study

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Rastko Gajanin, BSc
Registration Number 11930500

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Ing. Dr.techn. Stefan Nastić, BSc
Assistance: Univ.Ass. Dr. Andrea Morichetta
Anastasiya Danilenka, MSc

Vienna, August 20, 2024

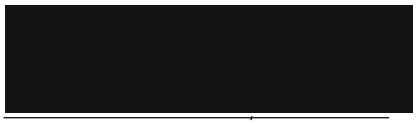
Rastko Gajanin Stefan Nastić

Declaration of Authorship

Rastko Gajanin, BSc

I hereby declare that I have written this Diploma Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, August 20, 2024



Rastko Gajanin

Acknowledgements

Firstly, I would like to thank Professor Nastic for his support and guidance throughout the past year. Our collaborations were both enlightening and enjoyable, and I am truly fortunate to have had the opportunity to work under his mentorship.

Andrea and Anastasiya, I am deeply thankful for your supervision of this thesis. Thank you for all the discussion and feedback sessions which kept me on the right track and imparted countless lessons on both federated learning and academic excellence. I found immense joy in our collaborative work, especially during the creation of the paper that resulted from this thesis.

During the entirety of my studies, my parents provided me with unwavering support and encouragement. Through their academic work and commitment, they inspired me to pursue this level of education as well. I will be forever grateful to them for providing me with the privilege to fully focus on my studies.

I wish to express my most sincere gratitude to my girlfriend, my greatest fan, Ana. Countless hours on video calls, regular visits and outings brought me so much joy and have been the main sources of strength and motivation throughout my academic journey. Your encouragement, understanding and gentle reminders to rest have been some of the most crucial factors of my success. I thank you wholeheartedly for believing in me and for always being by my side.

To my sister, Teodora, I would like to sincerely thank you for your persistent emotional support and the many conversations we had over coffee breaks and lunch. They helped me get much-needed mental rest, laughter and encouragement. Your presence kept me grounded and motivated throughout my studies and I will always remember that.

Lastly, I wish to thank my close friend, Emir, with whom I shared so many courses, coffees and rides home. Completing our studies side by side has been an immense honor and a huge source of motivation. Your caring advice and support kept me on track and helped me approach my studies with a clear mind. Our friendship made the pursuit of this degree truly unforgettable, thank you.

Kurzfassung

Die zunehmende Verbreitung von IoT-Geräten im Alltag bietet einzigartige Möglichkeiten, die kontinuierlich erfassten Daten für personalisierte Machine-Learning-Aufgaben zu nutzen. Ein prominentes Anwendungsbeispiel ist die Erkennung menschlicher Aktivitäten (Human Activity Recognition, HAR), die in Bereichen wie dem Gesundheitswesen, der Arbeitssicherheit, in Smart Homes und darüber hinaus Anwendung findet. In diesem Kontext ist der Schutz der Datenprivatsphäre von größter Bedeutung, wobei sich Federated Learning (FL) als eine vielversprechende Methode zum dezentralen und datenschutzkonformen Training von ML-Modellen erwiesen hat. Durch den Einsatz von FL können HAR-Modelle trainiert werden, ohne dass die Daten der Endnutzer über das Netzwerk übertragen werden, wodurch strenge Datenschutzstandards gewährleistet werden. Allerdings stellen die ausgeprägte Heterogenität und die nicht-unabhängig und identisch verteilten (non-IID) Daten im HAR-Bereich erhebliche Herausforderungen für FL-Methoden dar. Die Variation lokaler Label- und Feature-Verteilungen sowie die Heterogenität der Geräte in HAR-Datensätzen führen zu einer Verschlechterung der Modellleistung und zu Instabilitäten im Trainingsprozess. Während asynchrones FL die Systemaspekte der Heterogenität adressiert, verstärken sich dadurch zugleich die negativen Auswirkungen der non-IID-Daten auf das Modelltraining durch häufigere und differenziertere globale Modellaktualisierungen.

Da zentrales Lernen (Centralized Learning, CL) nach wie vor eine weit verbreitete Praxis im Bereich des maschinellen Lernens ist, gestaltet sich der Übergang zu Federated Learning (FL) aufgrund der Herausforderungen und Komplikationen, die durch extrem non-IID HAR-Daten entstehen, als nicht trivial. Dieser Übergang stellt einen häufigen Prozess in der FL-Forschung und -Praxis dar, der oft zeitaufwendig ist und zahlreiche Fallstricke birgt. Zur Bewältigung dieser Problematik schlagen wir eine neuartige Methodik vor, die den Übergang von CL zu FL bei non-IID-Daten erleichtert. Diese Methodik bietet einen strukturierten und formalen Ansatz, der potenzielle Herausforderungen explizit hervorhebt. Zur Evaluierung der vorgeschlagenen Methodik präsentieren wir eine Fallstudie, die den Übergang von CL zu FL anhand eines realistischen HAR-Datensatzes untersucht. Diese Fallstudie wird erweitert, um empirisch die Auswirkungen von Datenaugmentation, Skalierung, Optimiererwahl, Lernrate, Batch-Größe und Serverauslastung auf das FL-Training und die Evaluierung zu analysieren. Zu den zentralen Erkenntnissen gehören: Die Nutzung eines zentralen Testdatensatzes für eine gerechtere Evaluierung geht zulasten des Datenschutzes; der Stochastic Gradient Descent mit Momentum (SGD-m) zeigt

im Vergleich zu ADAM eine bessere Leistung als Optimierer; globale Skalierung birgt Datenschutzrisiken, ermöglicht jedoch eine bessere Leistung, da sie teilweise das Feature-Skew korrigiert; trotz globaler Skalierung bleibt das Feature-Skew auf Klassenebene bestehen und ist insbesondere bei Minderheitsklassen ausgeprägter; ausgelastete Server können die Konvergenz des Modells auch bei verlängerten Trainingszeiten beeinflussen. Schließlich haben wir zur Implementierung der asynchronen FL-Komponente unserer Methodik und zur Förderung weiterer Fortschritte in diesem Bereich das populäre Flower-Framework erweitert und die Lösung als Open-Source-Implementierung des asynchronen FL veröffentlicht.

Abstract

The prevalence of IoT devices in everyday life provides unique opportunities to utilize the variety of continuously acquired data for personalized Machine Learning (ML) tasks. One such use case is human activity recognition (HAR), which can be employed in healthcare, occupational safety, smart homes, and beyond. In HAR, data privacy is paramount, and Federated Learning (FL) has proven itself an excellent strategy for training ML models in a distributed and private manner. FL can be leveraged to train HAR models without passing the client's data over the network while maintaining strict privacy standards. However, highly heterogeneous and non-Independent and Identically Distributed (non-IID) HAR data introduces severe challenges for FL methods. Variations in local label and feature distributions, as well as device heterogeneity in HAR datasets, lead to poor performance and instability in model training. While asynchronous FL addresses the system aspect of heterogeneity, it amplifies the negative effects of non-IID data on model training due to more frequent and fine-grained global model updates.

Given that centralized learning (CL) is still a widespread ML practice, transitioning to FL can be non-trivial due to the challenges and complications caused by extremely non-IID HAR data. This transition is a frequent process in FL research and practice, and it is often time-consuming and contains many pitfalls. To address this issue, we propose a novel methodology for transitioning from CL to FL with non-IID data, providing a structured and formal approach that highlights potential challenges. To evaluate this methodology, we present a case study of transitioning from CL to FL on a realistic HAR dataset. We extend this case study and empirically examine the implications and effects of *data augmentation, scaling, optimizer, learning rate, batch size and busy servers* on FL training and evaluation. Our main findings include: using a centralized test set for fairer evaluation comes at the cost of privacy; better performance of SGD-m over ADAM as the model optimizer; global scaling introduces privacy risks but enables better performance as it partially addresses feature skew; even after global scaling class-level feature skew persists and is more prominent for the minority class; busy servers can significantly impact model convergence even with extended train time. Finally, to implement the asynchronous FL component of the methodology and promote further advancements in this field we extended the popular Flower framework and published the solution as an open-source implementation of asynchronous FL.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In the last decade, IoT has been able to connect millions of devices, supplying an unprecedented amount of data used by applications to provide innovative services. Considering individual use, smartphones, smart wristbands and smartwatches have become integrated into people's everyday lives. One prominent task emerging from the prevalence of these devices is the collection of real-time data from individuals to provide assistance; with the most notable application being human activity recognition (HAR). HAR rapidly became pivotal in many areas, such as healthcare, human-computer interaction, surveillance systems, entertainment, and more [1]. HAR tasks span from recognizing simple and common activities, such as walking or running, to assisting in more complex ones, such as doing laundry or preparing meals [2]. Thus, gathering data from various sensors present in smart devices, such as accelerometers, gyroscopes, and magnetometers, becomes essential to precisely model the observed activities. At the same time, the pervasiveness of such applications comes with costs. It increases the devices' computation demands, the need for real-time feedback and it amplifies privacy concerns, especially in sensitive applications [3].

Federated learning (FL) [4] has been emerging with the promise to address these challenges, offering methods for privacy, security, and scalability by decentralizing the machine learning (ML) model training and shifting it to the clients' devices, therefore making it particularly suitable for IoT systems [5]. In FL, the training of ML models is carried out on multiple individual devices that own data, i.e., clients, while a server overlooks and orchestrates the learning process. In each training round, the server selects a subset of clients for model training and sends the current global model to this subset. Clients of this subset train on the current global model. After local training, they send the updated models back to the server, where these are aggregated into the new global model. Indeed, FL serves numerous use cases spanning from training on privacy-sensitive

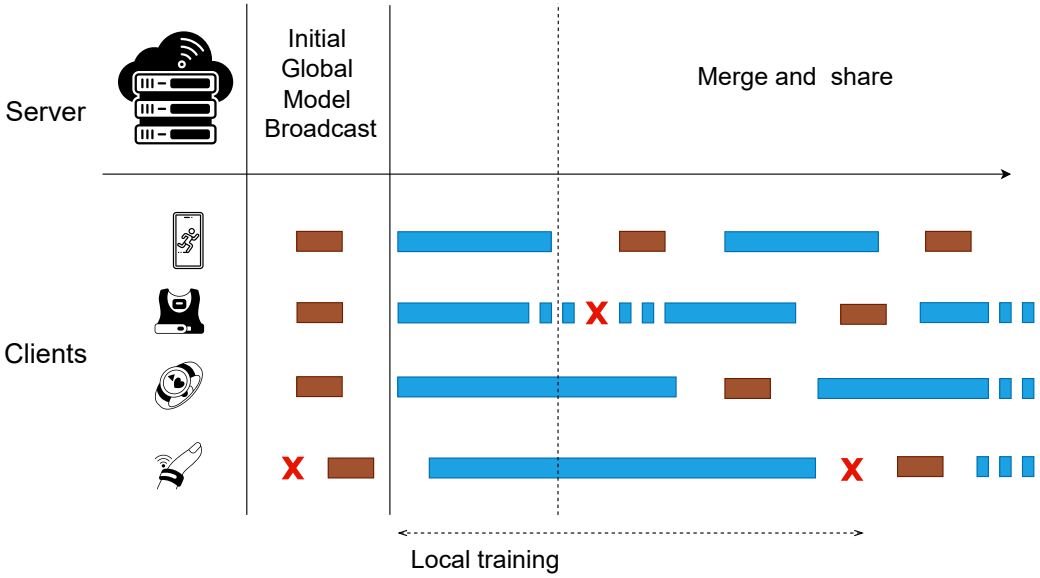


Figure 1.1: High-level perspective of the asynchronous federated approach for Human Activity Recognition.

medical data [6] to search query suggestions based on user typing patterns residing on their smartphones[7], in recent years also contributing to smart cities, industry 4.0, autonomous driving, and more [5].

Although FL's privacy-preserving and distributed nature introduces several benefits, it comes together with new challenges, especially for dynamic scenarios such as HAR. HAR typically shows *broad data heterogeneity*, caused by many individuals performing different actions in different ways and at different rates. This scenario produces high intra-class variability and inter-class dissimilarity [8], making client model training non-trivial. Indeed, it is established that having non-independently and non-identically distributed (non-IID) data across FL clients leads to divergent local model updates and, consequently, undermines global model performance [9]. Furthermore, device heterogeneity in HAR-IoT applications impacts the FL model training by introducing unreliable device connectivity and limitations in storage and computation capabilities [10].

To mitigate the challenges associated with real-time updates in such unstable scenarios, asynchronous FL (AFL) was proposed [11]. This approach allows clients to train models and send updates once ready or connected while the server continuously aggregates the received model updates. Unfortunately, AFL amplifies the problem of diverging models with non-IID data. In fact, AFL natively favors clients that train faster and communicate more frequently with the server [12]. Therefore, AFL on non-IID data further increases the global variance of model updates [13].

Figure 1.1 depicts the workflow of AFL for HAR. First, the FL server broadcasts the

global model (*brick red* rectangles); then, each device can immediately start the local training (*light blue* rectangles). Once the client completes the local training (even in case of device or network-induced delays, see red 'X'), it sends the updates to the server, which merges them with the global model and sends the new global model back. As of today, the challenge of non-IID data in asynchronous FL remains open [12], urging the development of robust methods to address it.

1.2 Problem Statement

The open issues in both HAR and FL research make transitioning from centralized learning (CL) to FL and AFL with non-IID data not straightforward. Each time this scenario arises, researchers must develop a new transition process from CL to FL. Moreover, this involves making several key design decisions and identifying pitfalls, which typically emerge only after extensive analyses, consuming additional research time. The absence of a predefined standardized and structured way of approaching this transition largely limits the adoption of FL in practice, especially when applied to highly non-IID problems such as HAR.

In this thesis, we present a novel methodology to transition from CL to FL, outlining an extensive evaluation of the main design decisions that need to be taken. We present a realistic HAR case study and exhaustive empirical analysis of our newly proposed methodology to illustrate challenges, implications and potential pitfalls arising when developing (A)FL systems with non-IID data. We aim to enable the HAR application developers to evaluate the key design decisions of the CL and federated model training and to reach optimal design choices considering the performance of ML models. Our work is part of a large Linux Foundation project, Centaurus, [1] that provides a novel open-source platform for building unified and highly scalable public or private distributed Edge, Cloud, and 3D continuum systems.

1.3 Research Questions

With this thesis, we aim to answer the following research questions:

- **RQ.1: How to uniformly structure the process of transitioning from centralized to synchronous and asynchronous federated learning in non-IID settings and what design decisions are to be considered during this process?**

  This question is answered through the proposal of a novel methodology introducing structure to the transition process from CL to SFL and AFL. This novel methodology leverages a typical ML development pipeline for CL as the base. It then introduces modifications of the base pipeline, as well as additional design decisions, parameters and steps required for developing ML models in FL settings with non-IID data.

[1] https://www.centauruscloud.io/

- **RQ.2: What are the effects and implications of different design decisions, proposed in the methodology, on FL model training with non-IID data?**

  We answer this question through a comprehensive empirical evaluation of the considered design decisions applied to a HAR use case. This uncovers the effects of these design decisions on the performance of the resulting ML models and provides key takeaways about the significance and implications of these decisions on the FL training and FL evaluation processes.

- **RQ.3: How can the asynchronous FL step of the methodology be integrated into an existing open-source FL framework?**

  To answer this question we implement the AFL method described in our novel methodology as an extension of the open-source Flower [14] framework. It is released as an open-source AFL implementation based on the works of [11, 15, 16]. The solution enables simple prototyping, reproducibility and adaptability with the ultimate goal of promoting further research in this area.

1.4 Methodology

To answer the research questions, the thesis will follow a methodology consisting of five main phases:

1. **Relevance:** Establishing the relevance of the problem of transitioning from CL to FL in realistic non-IID settings through a literature review. The result of this phase is a report on the state-of-the-art literature on the topics of non-IID data issues in Federated Learning, HAR with Deep and Federated Learning and comparison of centralized and federated learning with non-IID data.

2. **Design:** Designing a methodology that provides a unified view on the transition from CL to SFL and AFL in non-IID settings. The result of this phase is a standardized and prescriptive methodology for transitioning from CL to SFL and AFL with defined design decisions and tunable parameters.

3. **Implementation:** This phase encompasses the implementation of the methodology artifacts and, with that, the development of the flower extension for AFL. The result of this phase is the implementation of all steps in the methodology including the open-source flower integration that implements AFL. For the sake of clarity, reproducibility and traceability, the result of this phase will also encompass clear documentation of the produced artifacts and implementation details in a dedicated chapter.

4. **Case study:** This phase introduces the case study of a highly heterogeneous HAR dataset and evaluates the proposed methodology by applying it to a realistic

use case. The result of this phase is a clear description of the dataset characteristics, presentation of the data preprocessing steps, key evaluation metrics, model architecture and model hyperparameter tuning results.

5. **Evaluation:** This phase involves empirical assessment and further discussion of the challenges and common pitfalls of various design decisions proposed in the transition methodology. The results are key takeaways about the discussed design decisions that allow the community to leverage our insights and facilitate further research on this topic.

1.5 Structure

The thesis is structured as follows. Firstly, Chapter 2 provides a comprehensive overview of the foundational concepts related to Human Activity Recognition, Distributed Machine Learning and Federated Learning. Then, in Chapter 3 we present the results of our literature review on the topics of HAR with deep and federated learning, implications of non-IID data on FL and analytical comparisons of CL and FL. We proceed to introduce our novel methodology for transitioning from CL to FL in the presence of non-IID data in Chapter 4. This is followed by Chapter 5 where a case study of a transition to FL within a HAR context is presented. There we utilize and evaluate our proposed methodology. In Chapter 6 we illustrate the effects and implications of different design decisions introduced in our methodology based on an exhaustive empirical analysis. To enhance the clarity and reproducibility of thesis contributions and facilitate implementation navigation, Chapter 7 presents the implementation details of the produced software artifacts, focusing primarily on the AFL extension and its relation to flower framework. Finally, in Chapter 8 we summarize the contributions and main takeaways of this thesis and propose several avenues for future research.

CHAPTER 2

Background

Before proceeding with the primary contributions of the thesis, we wish to set a common foundation and introduce the main concepts through this chapter. Therefore, a brief primer on *Human Activity Recognition*, *Distributed Machine Learning* and *Federated Learning* is presented in the following.

2.1 Human Activity Recognition

Over the past decade, developments in sensor technologies have significantly improved the sensor quality while reducing cost, which led to their widespread adoption. Concurrently, the fields of Internet-of-Things (IoT) and edge/ubiquitous computing were challenged to follow the pace of these advancements. This caused a surge of interest in Human Activity Recognition (HAR) applications [17, 18, 19] and enabled remarkable contributions to personal healthcare [20, 21], security & surveillance [22, 23] and human-computer interaction [24, 25, 3].

HAR can be defined as the process of recognizing individuals' actions and states (e.g. walking, running, cycling, standing, laying down and sitting) based on sensor readings and an efficient learning algorithm [3, 18]. According to [26] HAR can be categorized based on the sensor type into Vision-based and Sensor-based. Sensors used in the former primarily consist of cameras with a fixed position, while for the latter, they encompass wearable devices and smartphones. Since using cameras for HAR introduces more privacy risks, carries larger implementation costs and is less pervasive than smartphones, sensor-based HAR has gained more popularity in recent years [3].

For performing sensor-based HAR, the most commonly used sensors are accelerometers, magnetometers and gyroscopes. This lies in their good performance, low cost and large prevalence (i.e. they are already built into the majority of modern smartphones) [18]. In [3] the authors describe the typical ML pipeline for sensor-based HAR and this pipeline

is depicted in Figure 2.1. Firstly, the data acquisition can be performed in a controlled environment (lab) or in a more realistic scenario, in free-living conditions (in-the-wild [27]). After sensor data is retrieved it is cleaned and noise is removed. Then, it is segmented into windows, and the features are extracted (e.g. Time domain features, Frequency domain features). Lastly, the ML model is trained on the selected subset of features, evaluated and deployed.

Each of the steps can be modified and there is a variety of modifications that already exist, however, the most commonly modified component is the learning algorithm. According to [18] ML-models leveraged for sensor-based HAR can be divided into four categories: deep learning (Deep, Convolutional and Recurrent Neural Networks), ensemble learning (Random Forests, Gradient Boosting), shallow models (K-Nearest-Neighbors, Support-Vector-Machines) and Fuzzy Logic Systems. Another category in this classification, inspired by the categories provided by [26], are probabilistic models (Naive Bayes, Hidden Markov Models). While all of these approaches address certain shortcomings and improve the HAR performance there are still several persistent challenges in this field: 1) intra-class variability and inter-class dissimilarity - different individuals perform the same actions differently which complicates model training [8], 2) identifying composite activities consisting of multiple atomic activities (e.g. playing basketball consists of walking, running and sitting) [19] 3) identifying concurrent activities (e.g. talking and walking) [19, 3], 4) ensuring sufficient data privacy (the majority of these approaches are centralized implying that sensitive data is communicated over potentially unsafe networks).

2.2 Distributed Machine Learning

As the amount of data generated across the edge-cloud continuum continues to grow, centralized machine learning solutions became processing and storage bottlenecks. This encouraged the researchers to explore distributed processing options that would parallelize the training process [28]. Distributed Machine Learning can be seen as an umbrella term for all solutions that distribute ML model training and/or inference to a certain extent.

2.2.1 Categorization

Verbraeken et al. recognize several classifications of Distributed Machine Learning in an extensive survey on this topic [28]. The first major discriminant is the parallelism type: *data parallelism* refers to a setting where different datasets are used for training the same model, while *model parallelism* describes a reversed context, the same dataset being used to train different models, whose results are typically aggregated by ensembling [28]. In certain cases, these models are not aggregated but only kept on the clients. This leads to distributed multi-task learning [29, 30, 31, 32, 33, 34], where multiple clients attempt to locally train different models that are *related* and their outputs are usually not aggregated.

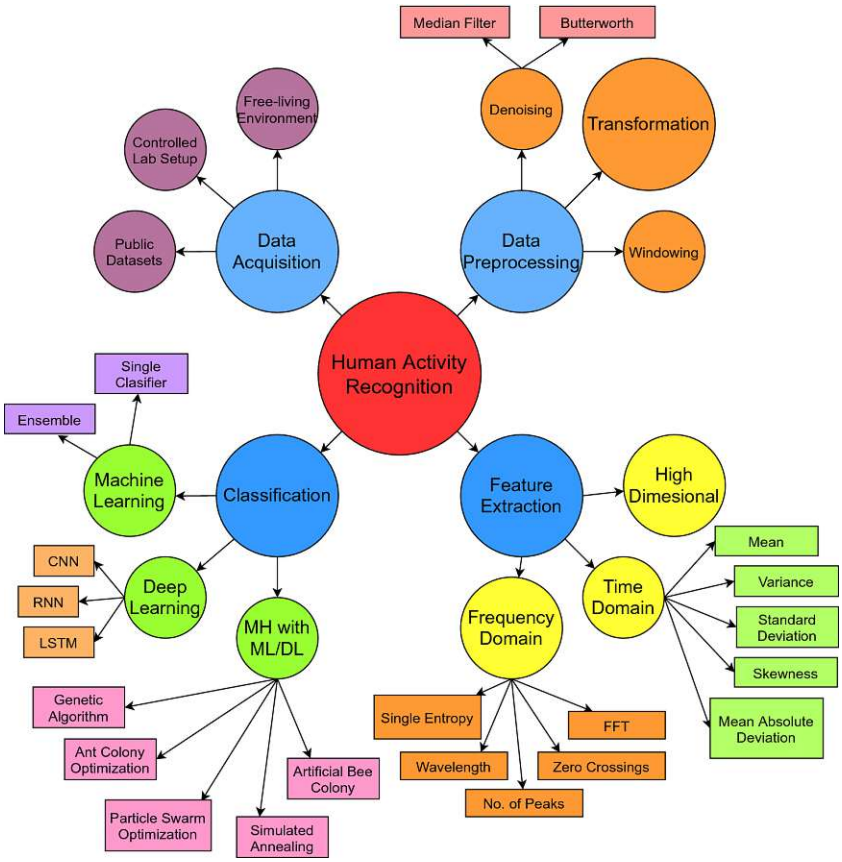


Figure 2.1: Typical ML model development pipeline for sensor-based HAR [3]

According to [35] the communication network topologies can be classified into three primary groups:

- *centralized* - Hierarchical with one level (one central node connected to all other nodes)
- *decentralized* - Multi-level hierarchical (Multiple intermediary nodes between the edge nodes and the central node)
- *fully distributed* - No central node, each node is connected only with its neighbors

Verbraeken et al. extend this classification by adding the following categories that are specific to distributed ML [28]. Visualization of these topologies created by the authors is provided in Figure 2.2:

- *Trees* - Typical hierarchical structure

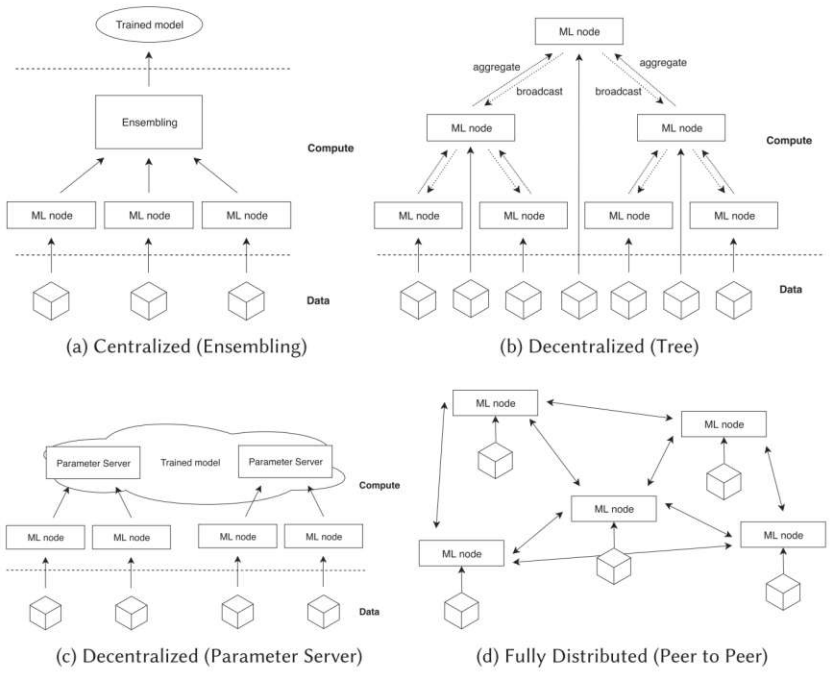


Figure 2.2: Different Distributed Machine Learning topologies [28]

- *Rings* - Usually employed to minimize the large communication cost of the broadcast operations
- *Parameter servers* - The cenralized nodes only maintain the state of the models, i.e. their parameters
- *Peer to peer* - Each node (client) has its own copy of the model parameters

Lastly, the authors classify the solutions based on the synchronization behavior of the training process and propose the following categories [28]:

- *Bulk Synchronous Parallel (BSP)* - Training is performed in strictly synchronized communication/computation rounds. Clients of one round wait until other clients in the same round finish before being able to proceed.
- *Stale Synchronous Parallel (SSP)* - Several faster clients may proceed even if stragglers (slower clients) are still training. This behavior is allowed until the maximum staleness is reached and all clients have to wait for the slowest one.
- *Approximate Synchronous Parallel (ASP)* - Instead of constraining the client contributions by staleness, this method constrains them by establishing if they are significant or not.

- *Total Asynchronous Parallel (TAP)* - Allows complete asynchrony and maximal speedup with the risk of slower convergence and even potential model drift [28].

2.2.2 Benefits and Applications

Besides the primary goal of alleviating the processing bottleneck imposed by centralized ML solutions, several other justifications make distributed ML an advantageous way of training ML models [36]. Firstly, modern mobile devices have a considerable amount of processing power (CPU and RAM) [37], making them viable options for computation offloading. Not harnessing such immense aggregated compute power by simply sending the data to a centralized server for training attributes inefficiency to such solutions. Secondly, as the amount of collected data on edge devices grows, transmitting such large quantities of data over the network is significantly more expensive than simply sending model parameters. This makes the distributed ML solutions appropriate for big data and training of large-scale models, as these typically require proportionally more training samples [28]. Lastly, the privacy of the data is one of the great concerns alleviated with distributed ML and is mainly addressed by not transmitting the client-local data over the network, homomorphic encryption or differential privacy techniques [38].

Distributed Machine Learning has found its application in a variety of fields, from natural language processing [7], medical data [6] and inherently distributed transactional [39] and astronomical [40] data of large magnitudes [28]. One prominent example would be the Google keyboard query prediction model that was trained in a federated (distributed) fashion [41, 7]. During training, the sensitive data (queries) remained on the clients' devices and only the updated models were sent to the server, thereby addressing the privacy concerns. Closely related to user text data is the data collected from smartwatches and smartphone sensors. This data is typically used for recognizing user activity [42, 43] and can be observed and processed online. Moreover, as it is generated closer to the edge it might contain identifiable information. Hence it would be suitable to employ privacy-preserving distributed ML (e.g. federated learning) as the training strategy for the models relying on this data. Likewise, federated learning has been a popular choice for many models trained on medical data [6] where the privacy of electronic medical record (EMR) data carries great significance. Common applications encompass EMR Data classification, EMR data extraction, disease diagnostics and medical image analysis [6].

2.2.3 Limitations & Challenges

As with any other problem that is being adapted for distributed processing, ML algorithms must be accordingly adjusted as well when transitioning to DML. This brings significant design/formulation overheads, especially when coupled with further constraints such as performance trade-offs (e.g. energy consumption, per resource efficiency, provider costs) [28]. The second major concern according to Verbraeken et al. is *fault tolerance*, which is significantly more problematic in synchronous solutions [28]. This is primarily because,

in case of a single client crash, other clients cannot make progress and are forced to wait, incurring significant overheads. Verbraeken et al. further argue that current frameworks lack reliable privacy support, even when the data is not transmitted over the network [28]. Lastly, the issue that also lies at the focus of this thesis is the inherent heterogeneity of the clients across the edge-cloud continuum. This heterogeneity does not only refer to the resource heterogeneity in terms of compute power and memory, but it refers to the *statistical heterogeneity* of the data as well. Different clients might not only have different amounts of CPU and memory resources available but different amounts, distributions and types of data as well. Such settings pose significant issues when training ML models, as they usually lead to unnecessary waiting, straggler issues, model drift and overall slower model convergence.

2.3 Federated Learning

The concept of federated learning (FL) was popularized through the work of McMahan et al.[4] published in 2017. As the amount of *privacy-sensitive* data grows on mobile (edge) devices, there is a need to process these data (train ML models) securely and efficiently. Fundamentally, federated learning can be observed as a means of achieving these goals. It is an encrypted [38] distributed model training approach consisting of multiple rounds. At the beginning of each round, the centralized parameter server selects a subset of clients and sends the current global model parameters to these clients. Each selected client then trains on its own data samples and only sends the updated model *parameters* back to the server. When all selected clients send their updates, these are then aggregated on the server and the round is finished [4]. The aggregation is usually the mean of all updated model parameters weighted by the number of samples the respective client used to produce that update. This basic outline of the federated learning algorithm is illustrated in Figure 2.4. According to the classification provided by Verbraeken et al., this method can be considered as a *block-synchronous data-parallel* form of distributed ML, with a *centralized parameter server* topology. The server is also responsible for the orchestration of local training processes, storage and distribution of the global model. The method ultimately ensures data privacy by keeping data at the client's premises, while shifting the computation to the data. Furthermore, by sending only model parameters over the network and not entire datasets a lower communication overhead is achieved [4].

Federated learning can be categorized based on multiple criteria. Firstly, the clients' datasets can differ in both dimensions: *samples* and *features* [44, 38]. If all clients possess the same features, but different samples, the setting is referred to as *horizontal FL*. On the other hand, *vertical FL* represents a setting where clients represent the same samples with different features. A visual depiction of these two settings created by [44] is provided in Figure 2.3. The overlapping of samples and features, as well as the combination of the two settings also represent potential contexts. These, however, introduce a high degree of complexity and will not be discussed further. Across this thesis when we refer to *federated learning*, we refer to the *horizontal FL*, where the clients hold different samples of the same set of features.

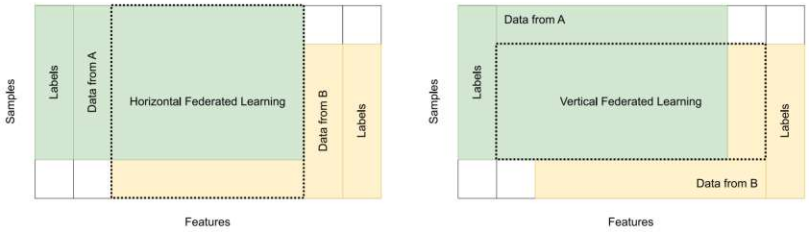


Figure 2.3: Illustration of horizontal (left) and vertical FL (right). In *horizontal* FL all clients possess the same set of features representing different sets of samples, while in *vertical* FL the clients represent the same set of samples with different sets of features. [44]

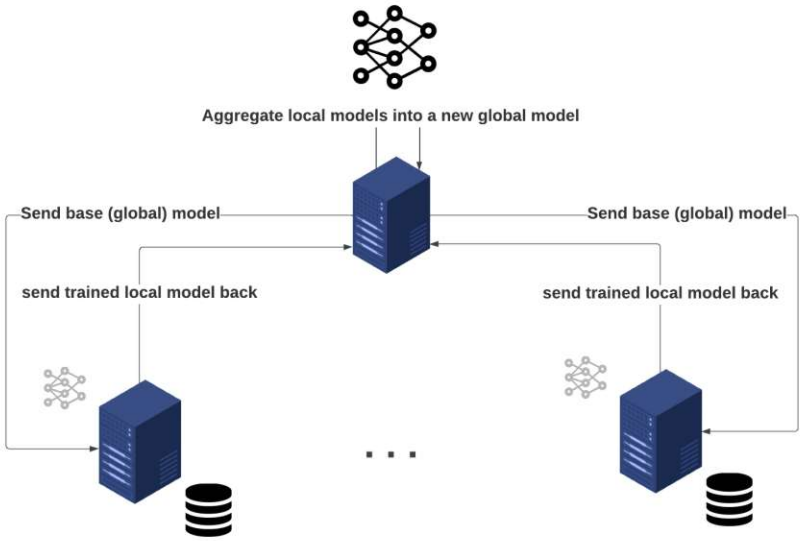


Figure 2.4: Depiction of Federated Learning. At the beginning of each round, the server sends curremnt global model parameters. Each of the clients continues training the model on their local dataset and, after a specified amount of local epochs, sends the updated model back to the server. The individual updates are then aggregated and this aggregate forms the new global model.

Since the previously described federated learning procedure is carried out in rounds, it is commonly referred to as *synchronous federated learning*, because the server waits for the responses from all clients. In the following, we discuss the *asynchronous* variant of FL, where the training is performed without such synchronization.

2.3.1 Asynchronous Federated Learning

Asynchronous Federated Learning was introduced as a means of alleviating the straggler issues caused by client heterogeneity (concerning both resources and data) [11]. This

adaptation is essentially characterized by the absence of training rounds that are standard for vanilla FL. Instead, as soon as the server receives a client update, it is immediately merged into the global model. The client can either immediately resume training with the newest global model, or be scheduled to train, where it will pull the newest global model after being triggered [11]. The model aggregation typically uses a staleness coefficient that regulates how much impact an update has, based on its training time (staleness) [11, 45].

Asynchronous Federated Learning also possesses several drawbacks. The large number of clients might lead to frequent model updates, incurring significant overheads and making the centralized server the bottleneck. This would especially be evident when training bigger models as many parameters are to be updated. Besides, aggregating individual updates into the global model violates secure aggregation protocols, weakening the privacy properties of the solution [46, 47]. To address these limitations, semi-asynchronous methods have been developed [44]. For instance, in works by So et al. and Nguyen et al. K local updates are buffered and applied at once on the server [47, 46].

CHAPTER 3

Related Work

This chapter encompasses a literature survey on the following three topics: 1) Human Activity Recognition with Deep and Federated Learning, 2) Establishing the negative effects of the data skew in Federated Learning and 3) Comparison of centralized and federated learning. The goal of this chapter is to identify current gaps and establish the context for the contributions presented in this thesis.

3.1 Human Activity Recognition with Deep and Federated Learning

Thanks to the advancements in both the processing as well as sensing power of smartphones and similar wearable devices (e.g. smart watches) the amount and quality of data collected on such devices started carrying great potential for the development of ML models. Several groups of researchers have successfully aggregated the readings from smartphone and wearable device sensors into insightful datasets ready to be further incorporated in the ML pipeline. For instance, a very popular and widely-used dataset, published at the UCI repository is presented in [48]. It contains inertial sensor readings of 30 subjects performing typical daily actions: standing, sitting, laying down, walking, walking downstairs and upstairs. Data collection was conducted in a lab and the measurements were taken with the same device for all users. Another prominent dataset, WISDM [49], also collected accelerometer data from 29 subjects while they were performing their daily activities. The dataset however significantly limits the users to keeping the phone in a specific position while monitoring the data. Contrary to the previous datasets the Extrasensory dataset presented by Vaizman et al. [27] was collected *in-the-wild* by leveraging a large variety of devices and without constraining the users on how to use them, thereby representing a very reliable real-world setting. It also encompasses more labels than the six base ones (sitting, standing, laying down, walking, running, cycling) including location (e.g. at work, at home) and compound activity labels (e.g. playing

basketball). Based on its strong alignment with real-world scenarios, we selected this dataset for our case study presented in Chapter 5.

In order to predict human activity from such an abundance of sensor data, the temporal, spectral and statistical features are usually extracted first and then fed into a classifier model. There exists an entire spectrum of solutions modifying both the feature extraction step as well as the model design step. The authors of the Extrasensory dataset have also provided a simple MLP baseline [50] with the features they extracted. Several other authors also used this dataset for the evaluation of their approaches. For instance, in [51], the authors propose a hierarchical neural network model that firstly classifies the sample into two super-groups of the six primary labels: stationary (sitting, laying down and standing) and non-stationary (walking, running, cycling). Furthermore, [42] have trained convolutional neural networks on multiple HAR datasets including Extrasensory and found out that the dataset is significantly more difficult to model compared to other HAR datasets, primarily due to the unconstrained nature of data collection.

As smartphone devices grew more powerful, they also developed the ability to withstand ML model training processes previously reserved only for stronger, centralized, servers. This, coupled with the privacy-sensitive sensor data collected through these devices, makes training HAR models in a federated manner a natural progression in this field of research. In their work [52] leverage FL to train a deep neural network to model the Heterogenous HAR data collected by [53]. They prove that the model performance in both balanced and simulated imbalanced settings is acceptable for this dataset within the federated context. However, the applicability of their insights to the Extrasensory dataset is limited as the actions performed by the users were scripted, leading to a more uniform label distribution. Authors of [31] introduce MOCHA, a federated multi-task learning framework for addressing both statistical as well as system-level heterogeneities in FL, which used the UCI-HAR dataset for evaluation [48].

Moving towards asynchronous FL, authors of ASO-Fed [15] propose learning shared feature representations on the server with the goal of addressing non-IID-ness issues. This, paired with a decayed update coefficient for managing the influence of the new updates on the global model and the modification of the local loss function, similar to [54], led to improved results on the Extrasensory dataset. In their further work, related to asynchronous FL with sensor data [55], the same authors develop a drift detection and drift correction scheme to address this issue in asynchronous FL and evaluate the performance of their solution on multiple datasets, including Extrasensory as well.

While many works approach HAR with deep and federated learning, many of these solutions are evaluated on relatively balanced datasets where data was collected in controlled environments. Moreover, as local training updates can be delayed or dropped due to imbalances in data, system heterogeneity and network partitions, synchronous FL methods have only limited applicability to such real-world settings. While there exist approaches that model HAR with asynchronous FL [15, 55], these contributions solely use the HAR dataset as an evaluation dataset and do not dive into the challenges of federated HAR. **Our work** instead proposes a HAR-oriented analysis approach while

focusing on the transition from CL to FL in the presence of highly non-IID data. We offer a thorough examination of the HAR use case from both IoT and FL perspectives. We further aim to provide an in-depth analysis of the causes of performance degradation when designing FL systems that model non-IID (HAR) data.

3.2 Non-IID Data Implications on Federated Learning

In CL, data is kept centrally, meaning the model will learn from all clients simultaneously within one training epoch, dampening the negative effects of non-IID data on model training. However, placing the non-IID dataset in a federated context makes model training significantly more challenging, as the models train only on a single client's data within one epoch, which carries a great risk of model divergence.

A survey on implications of Non-IID data on federated learning presented by Zhu et al.[56] identifies *three* primary types of data skew:

1. **Feature (attribute) skew** - Describes settings where different clients have different distributions of the same features.
2. **Label skew** - Different clients have different label proportions/distributions.
3. **Temporal skew** - Describes a skew in the distribution of temporal (time-series) data. These differences appear in the correlation of the time-series samples, e.g. different frequencies (period lengths) and/or signal amplitudes (in IoT sensors).

Quantity skew is identified as orthogonal to the prior classification and describes the variation in the number of samples among the clients.

Many authors have confirmed that Non-IID data leads to degraded performance of parametric models (neural networks)[57, 58, 4, 56, 59] and always leads to model divergence in horizontal FL[56]. In [56] the authors highlight that deep neural networks are more prone to the negative effects of data skew than shallow networks. They further state that Non-IID data does *not* influence the performance of non-parametric models (such as decision trees) in both horizontal and vertical FL.

Zhao et al. [57] perform a more in-depth analysis of this issue. Besides empirically measuring the negative effects of data skew through test accuracy, they also monitor the model parameter differences among the client updates, even for each model layer. Through experimentation, they confirm their hypothesis that model divergence is a direct cause of reduced accuracy. The authors also visualize model parameters in IID and non-IID FL settings within one training round consisting of multiple local training epochs. This visualization is provided in Figure 3.1. One can observe that in IID FL settings, the model parameters (weights) advance in similar directions, while in non-IID FL settings, the discrepancy in model parameters between two clients (Client 1 and Client K) within one round is much larger and continues to grow as the local epochs progress. This implies

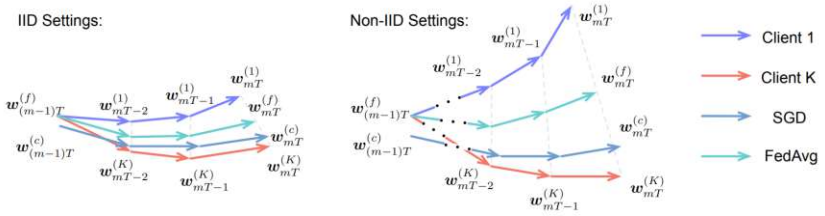


Figure 3.1: Visualization of model divergence within one training round in IID and Non-IID settings provided by [57]. Clients start with the same initial parameters (after synchronization). In IID setting, the difference between the model parameters of each client, averaged clients (FedAvg) and centralized training (SGD) is relatively small compared to the non-IID setting. It can further be observed that model divergence increases together with the number of local epochs when data skew is present.

that keeping the number of local epochs lower in non-IID FL settings can be beneficial to avoid severe model divergence.

In their work, Nguyen et al. [60] identify another specific type of data skew which they name *cluster-skew non-iid data*. They argue that, in modern real-world applications, the distributed data typically has a cluster structure. This implies that clients can be grouped into clusters that have similar data distributions.

Current literature that establishes and analyzes the negative effects of data skew in FL focuses primarily on *synchronous* FL and the evaluations are usually performed on synthetic datasets. In contrast, we aim to extend the state-of-the-art by examining the effects of data skew in *asynchronous* settings and using a realistic dataset containing naturally distributed non-IID data for evaluation.

3.3 Differences between Centralized and Federated Learning

A similar contribution to our work is presented by Drainakis et al.[61]. They evaluate the effects of transitioning from centralized (CL) to FL settings in non-IID data scenarios. While our focus lies on defining a structured methodology to transition from CL to FL and the implications of various design decisions within this methodology, their focus is on the comparison of model convergence for CL and synchronous FL (SFL) in terms of network resources and energy consumption, as well as establishing the trade-off between model performance and resource footprint during training. Furthermore, while they compare CL and SFL performance, they do not consider the asynchronous FL (AFL) setting which is the primary setting of our evaluation.

Another work that compares the performance difference between centralized and federated settings is presented in [62]. Here the authors simulate non-IID-ness by assigning an equal amount of samples from two out of ten labels to each client. In their evaluation,

they consider FedAvg as the representative of SFL and CO-OP [63] as the representative of AFL. They found that, in IID settings, the performance of SFL is similar to CL, while AFL performance is slightly degraded. In the simulated non-IID setting CL performs best, followed by SFL and AFL as least performant. Compared to their work, our evaluation differs in two aspects: 1) we consider what design decisions lead to degradation instead of just establishing it, and 2) we use a realistic dataset for our evaluation instead of a simulated one.

CHAPTER 4

From Centralized to Federated Learning with Non-IID Data

The process of developing centralized ML models is quite standardized and can be summarised (with various levels of detail) into 4 main steps: (1) data analysis and preprocessing (includes activities such as data partitioning, filtering, imputation, augmentation, feature selection, engineering, and scaling), (2) model training and tuning (includes evaluation metrics definition, tuning of the model architecture and hyperparameters, such as optimizer, batch size, learning rate, and stopping conditions), (3) performance evaluation and (4) model deployment. In addition to these tasks, steps 1 and 2 require an iterative process of choosing the right techniques for data processing and model configuration. Through these steps, a solid centralized learning (CL) baseline can be built and used as the foundation for transitioning to FL. In the following, we describe how this CL baseline is established and propose the structured approach to transitioning from CL to FL highlighting the relevant design decisions and considering the caveats that non-IID data might bring.

4.1 Establishing the CL Baseline

Firstly, a CL baseline is built and it will be used as the foundation for the transition to the FL setting. This process encompasses two major phases consisting of multiple steps: **data analysis & preprocessing** and **model training, tuning and validation**. This sequence of steps is the standard procedure for ML model training in CL pipelines, therefore it will only be briefly outlined in the following.

4.1.1 Data Analysis and Preprocessing

The aim of this phase is to gain an understanding and overview of the data at hand and prepare the data before feeding it into the model. This is achieved through:

1. *Exploratory Data Analysis (EDA)* - Observing the data in the dataset (e.g. feature and label distributions, missing values, relationships among features...)
2. *Data Partitioning* - Splitting the data into train, validation and test sets
3. *Feature Selection* - Extracting only the features considered useful
4. *Filtering / Imputation* - This involves removing or imputing missing or invalid values
5. *Scaling* - Bringing the features to the same scale (either min-max or through standardization)
6. *Augmentation* - Adding samples through various oversampling techniques typically to increase the representation of minority classes

4.1.2 Model Training, Tuning and Validation

After the data has been examined and prepared, the *evaluation metrics* (e.g. accuracy, mean squared error, F1-Score) are determined depending on the goal the model will try to achieve. Then, the *model architecture* is defined based on the insights about the problem complexity gained through EDA. With model architecture in place, *hyperparameter tuning* is performed on the model in order to find optimal values of hyperparameters (such as learning rate or batch size). Lastly, the model is validated on a test set and if the target performance is reached it is deployed, otherwise data preprocessing or model architecture are adjusted and the entire process is repeated.

4.2 Transitioning to FL

After the CL baseline is established, insights gained through EDA, adjustments and tuning can be leveraged in FL modeling as well. In the following, we discuss additional design decisions and adjusted steps of the typical ML pipeline that enable the transition to FL in non-IID settings.

4.2.1 Defining the Test-Set Generation Strategy

In order to have a partitioning plan in place before performing other data pre-processing steps, it must be defined how the test set is generated. In CL, due to the centrally located dataset, partitioning of data can be done using random shuffling and performance evaluation can further be reliably tracked throughout the whole training. However, the

absence of a centralized dataset affects the formation of train/validation/test sets in FL, allowing for three evaluation scenarios:

- **Hold-out-clients (HOC)** - Dividing *clients* into two subsets: 1) clients used for training and validation and 2) clients used for testing. The first group of clients splits local datasets only into train and validation partitions while, in the case of the second, the entire local datasets are reserved exclusively for testing.
- **Distributed evaluation (DE) -** Dividing *local datasets* into train, validation and test partitions, with sending and aggregating test-performance metrics on the server
- **Centralized Test Set (CTS) -** Applying *hybrid techniques*, such as maintaining a test dataset on the server while validating model performance on client's subset or parts of local datasets

The three partitioning settings influence the *semantics* of model evaluation. If HOC is used, for instance, it evaluates global model performance on "new" clients and entirely unseen feature/label distributions, whereas the CTS approach evaluates how well the global model performs on unseen data from known clients/distributions. The former offers a more stringent evaluation of the model's generalization capabilities and is appropriate for settings where model inference will be performed on clients who did not participate in training, while the latter is more applicable to a setting where the fixed set of clients is used for both training and inference.

Without detailed data distribution statistics, ensuring representative data partitions in non-IID settings is challenging, leading to skewed performance indicators and suboptimal model performance, further emphasizing the importance of the proper performance evaluation scheme. More details on the implications of the test scheme in FL and their empirical analysis through the HAR use case are provided in Section 6.3.1.

4.2.2 Data Analysis & Preprocessing

As data analysis and EDA have already been performed while establishing the CL baseline, these are not required in FL. Therefore, after defining the partitioning strategy, the train, validation and test *partitions* can already be made.

It is clear that, in CL, full access to the dataset enables the use of sophisticated techniques for data analysis and preprocessing. However, to properly federate these processes, it is necessary to establish the expected level of data heterogeneity and willingness to share local data statistics, potentially introducing additional privacy risks. Thus, one can choose from three main approaches for data analysis and preprocessing in FL:

- **Generalized approach** – employing techniques, insensitive to the diversity of local datasets (e.g., image processors that scale images based on a set mean and standard deviation).

- **Local approach** – employing techniques that preprocess data based solely on local dataset statistics, for instance, scaling data with local mean and standard deviation or performing data augmentation considering local data label imbalance statistics.
- **Federated analytics (Global) approach** – allowing for aggregation of data statistics from client devices on the server to form global statistics, mimicking CL data analytics, sharing it among all clients, and using it to guide local preprocessing.

After making this decision, several steps of the pre-processing pipeline remain the same as in the CL baseline. These include *Feature Selection*, *Filtering / Imputation* and *Data Augmentation* to a certain extent.

Scaling of the features is performed based on the previous decision. Either *local* or *global* scaling can be employed. Within the former, clients scale their features individually and only based on the data they have, whereas for *global* scaling, feature means and standard deviations (or other statistics) are shared either among the clients or with the server, where these are aggregated. These aggregated global means and standard deviations of all features are transmitted to the clients so they can scale their features accordingly.

Data augmentation can either mirror the approach used in CL, characterized by a fixed oversampling rate, or deviate from it by dynamically adjusting the augmentation process based on the client's local data distributions or other relevant inputs.

We illustrate the effects local and global analytics approaches have on scaling and on the resulting model performance for the non-IID HAR use case in Section 6.3.3. We further federate data augmentation strategies in Section 5.2.4 and show their effect in FL with non-IID data in Section 6.3.2.

4.2.3 Model Design and Tuning

In CL, the model is trained on a single device, which has consistent access to the entire training dataset. In contrast, in FL training occurs on multiple devices and involves recurrent broadcasting of the global model to devices and aggregating updates from them. Introducing FL affects several model design and tuning decisions described in the following.

Defining the training strategy

As noted in Section 2.3, FL training can advance *synchronously* or *asynchronously*. This decision depends on the reliability and heterogeneity of devices participating in FL, with the latter being more suitable for failure-prone scenarios such as those in IoT as it avoids waiting for slower devices (stragglers). Depending on whether synchronous or asynchronous FL is chosen, the process of broadcasting and aggregating the model differs, introducing more hyperparameters for training to consider. In the following, the typical workflows of synchronous and asynchronous FL are introduced.

Synchronous Federated Learning In **synchronous** FL (SFL) the step of model broadcasting involves a set of clients being chosen for the current round of training, parametrized by the S - number of clients to choose from the available pool of clients and a specific client selection strategy (random by default). The server sends them the current global model x_t and instructs them to train the passed global model further with their local data. After local training, each client i sends their updated model x_{t+1}^i back to the server where the server aggregates the resulting client models and updates the global model with this aggregate. The process repeats for a predefined number of rounds or until a stopping condition is satisfied. Equation 4.1 formally describes the synchronous update procedure where N is the total number of samples and n_i is the number of samples present on client i:

$$x_{t+1} = x_t + \sum_{i \in S} \frac{n_i}{N}(x_{t+1}^i) \tag{4.1}$$

Asynchronous Federated Learning In **asynchronous** FL (AFL) the process of broadcasting and aggregating model updates does not proceed in rounds; contrary to that, clients start training as soon as they merge their newest update with the global model and receive the new merged global model from the server [11, 16]. Thus, the model aggregation step merges the individual client's model update into the global model immediately. This step is parametrized by *mixing ratio/fedasync mixing alpha* which dictates the averaging weights (e.g. 50-50 or 30-70) of the global model and the current client's update. To normalize the update magnitudes similarily to SFL, the mixing ratio is multiplied by the proportion of samples held by the client that sends the update ($\frac{n_i}{N}$). The client updates are merged until the stopping criterion is met or the maximum training duration is reached.

To formally express the update rule of the asynchronous federated baseline we provide the following Equation 4.2. Assume that client i sent its gradients Δ_i after local training and the server is about to incorporate these gradients into the global model. The current global model is marked with x_t, the number of samples available at client i is marked with n_i and the total number of samples across all clients is marked with N. α_{FA} is the above-defined mixing ratio (Fedasync mixing alpha).

$$x_{t+1} = x_t + \alpha_{FA}\frac{n_i}{N}(x_t + \Delta_i) \tag{4.2}$$

If, however, it is decided that instead of gradients the clients send updated model parameters x_{t+1}^i (as in SFL), the update rule changes slightly and it is formalized in the Equation 4.3.

$$x_{t+1} = x_t + \alpha_{FA}\frac{n_i}{N}x_{t+1}^i \tag{4.3}$$

Evaluation, Architecture & Tuning

Similar to CFL, appropriate *evaluation criteria* must be established. These criteria typically build upon the existing evaluation metrics of the CFL baseline by incorporating additional FL-specific metrics, such as communication efficiency. The *model architecture* can either mirror the CFL baseline or be adapted to FL, taking into account various constraints like memory or network limitations. Once the model architecture is determined, we can distinguish between two categories of *hyperparameters* that will be tuned:

- *Base Hyperparameters* - These are the same as in CFL (e.g. batch size, learning rate, optimizer)
- *FL Hyperparameters* - These are specific to FL (e.g. number of local epochs, number of rounds, number of clients per round, mixing ratio in AFL)

Further, the choice of both the *model architecture and hyperparameters*, such as batch size and learning rate, due to the data and resource heterogeneity across devices, is influenced by memory constraints, convergence speed, overfitting tracking, and more. We talk more about the effects of hyperparameters in Section 5.4 and Section 6.4.1.

4.2.4 Performance Validation & Testing

While in CL the model is validated after each epoch and tested at the end, this procedure differs in FL. This phase encompasses monitoring the convergence (validating) and evaluating the tuned model with the predefined test set. Test set generation strategies are described in Section **??**. Model validation in SFL usually happens after the round is finished and before the start of the next round, therefore the model validation is *synchronized* with the training procedure. In AFL, however, the validation has to consider both the server and client load. Performing model validation after each model update is not recommended as it overloads the server with constant computation and the influence of these additional computations are discussed in more detail in Section 6.4.2. Therefore periodic validation initiated by the server might be a better option. Furthermore, in case the distributed evaluation (DE), in this context *distributed validation (DV)*, strategy is used, the clients selected for validation might already be busy with training, which leads to additional overheads. To address this, DV can also happen *asynchronously*, but this modification comes at the cost of a larger orchestration overhead.

Due to the distributed nature of model training coupled with non-IID data distributions the issue of **biased local validation/test sets** becomes more prominent. In other words, asking the client with a skewed view of the data to validate/evaluate a model will certainly produce skewed results as well. To address this a fair, and typically centralized, validation/evaluation scheme can be used.

The influence and trade-offs related to the decision of how the test set is generated are described in more detail in Section 6.3.1. If the established validation and evaluation

performance suffices, the model can be deployed. Model deployment may be considered the only step in the pipeline where FL has some sort of advantage over CL as in order to deploy the model one should only ensure proper final model broadcasting to all clients, allowing them to start using it for inference.

On the other hand, if the performance is insufficient, the issue is *diagnosed* and to addressed by either adjusting data preprocessing, model design & tuning steps, or introducing an *adaptation* of the FL workflow, described in the following.

4.2.5 FL Adaptations

Based on the discovery of issues caused by distributing the ML pipeline with non-IID data, the need for the modification of the standard FL workflow arises. The work of Zhu et al. [56] already provides a classification of modifications applied for non-IID data issues: *Data-Based* - modifying the data distributions (e.g. data sharing, adaptive data augmentation), *Algorithm-based* - personalizing the global model (e.g. adapting loss functions, adding personalized layers, knowledge distillation) and *System-based* (e.g. client clustering). Further, in their survey, Lu et al. [64] also name *dynamic client selection, adaptive aggregation and adaptive update weight adjustment* as potential methods of solving the non-IID data issue in FL. Based on the previous classifications we propose the following locations in the methodology where they are applied:

1. **Data-level modifications** - Such adaptations modify the data preprocessing/partitioning steps.
2. **Model-level modifications** - These modifications encompass changing the model architecture (e.g. modifying the layers), the hyperparameters or the loss function.
3. **Client-level modifications** - Such modifications happen on the clients and they can be performed on two points: 1) before local training starts or 2) after local training ends (before the model update is sent back to the server). These two points serve to 1) process the received up-to-date global model before local training and 2) process the gradients/resulting models before sending them back to the server.
4. **Server-level modifications** - These modifications take place on the server. Two common locations for these adaptations are:
    a) Client selection step - The adaptation aims at selecting the clients dynamically to reach a more balanced pool of client distributions
    b) Model aggregation step - This adaptation encompasses changing the global model update rule or re-weighing the clients to a more informed aggregation.

Among these four, the first modification affects the *Data preprocessing* step of the model development pipeline (Section 4.2.2), while latter three modify the *Model Training* step (Section 4.2.3).

4.3 Summary

The design decisions and additional parameters introduced to the ML pipeline by transitioning to FL in non-IID settings are illustrated in Figure 4.1. To sum up, although we can roughly apply the classical CL steps in FL, this distributed setting complicates their proper execution and poses additional design decisions, requiring more ingenuity, especially when facing non-IID data. These additional considerations encompass the preprocessing approach, mode of training, mode of evaluation and, optionally, adaptations to the FL workflow. CL baseline can provide useful insights for multiple decisions in the FL pipeline such as data preprocessing and model design. Therefore establishing this CL baseline is performed before considering the federated context. Chapter 5 presents the inherently heterogeneous HAR use case and introduces the methods of how the models were developed for CL, synchronous, and asynchronous FL. Later, in Chapter 6, we examine the effect of the presented design decisions on model convergence within the HAR context.

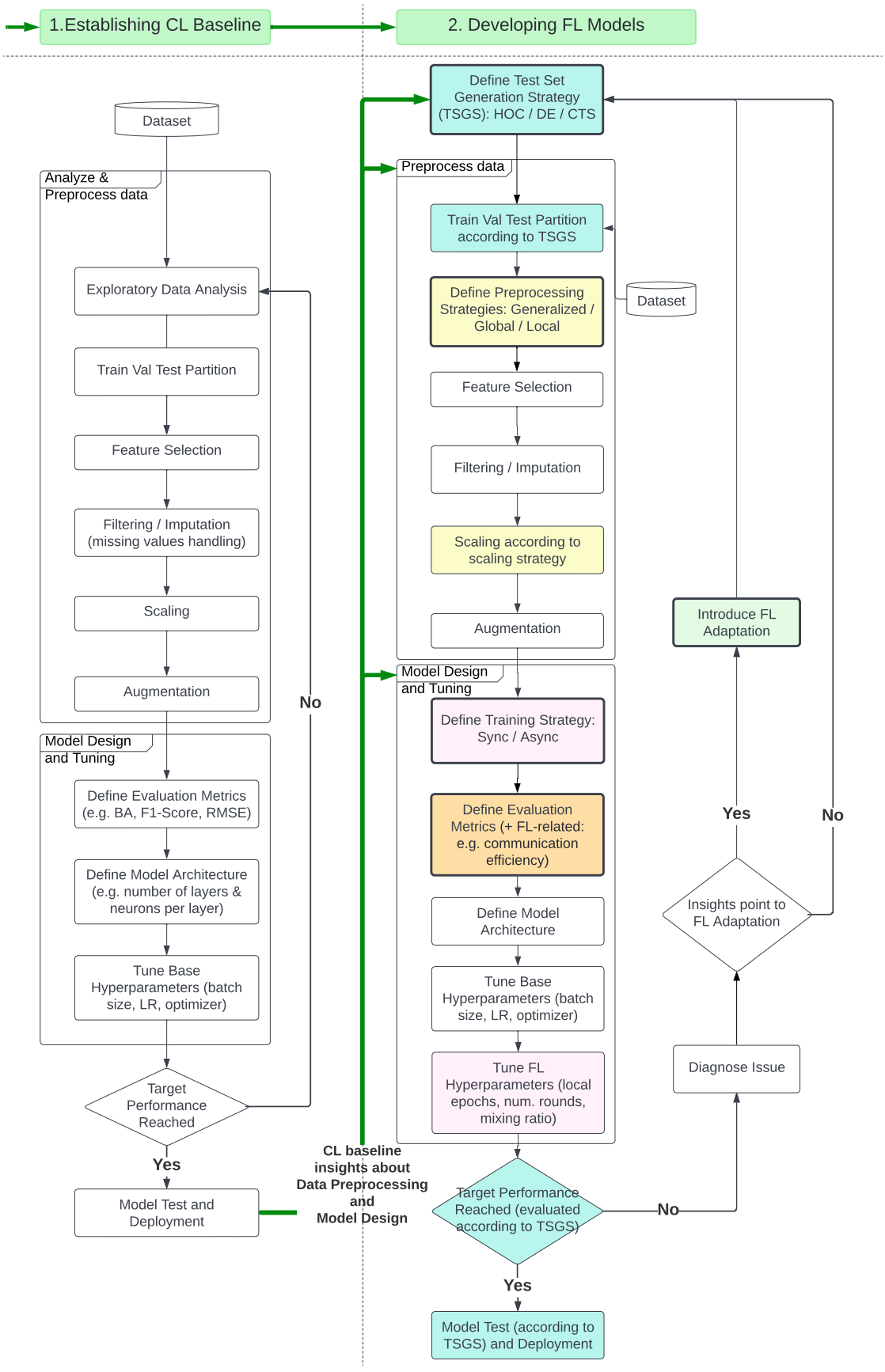


Figure 4.1: Methodology for transitioning from CL to FL with non-IID data

CHAPTER 5

Case Study: Federated Learning for Human Activity Recognition

Next, we introduce our use case scenario. First, we describe the used dataset and its characteristics. Then, we present the main data preprocessing adaptations to prepare the data for model tuning. We then introduce the key quality assessment metrics that will be used to evaluate the effectiveness of models in this specific non-IID setting. Lastly, we introduce the model architecture together with the results of hyperparameter tuning.

5.1 Extrasensory Dataset

Our work leverages the *Extrasensory* dataset [27]. This source contains sensor readings from smartphones belonging to 60 different individuals including sensors such as accelerometer, gyroscope, audio, etc. Time-series-related signal features encompassing various statistical and spectral properties were already extracted by the authors of the dataset. The original dataset contains in total 225 features from 11 sensors. As labels, the authors presented 6 primary mutually exclusive labels that describe the individual's current status: *standing, walking, sitting, laying down, running, and cycling*. The individuals themselves reported the labels through a dedicated smartphone app, during or immediately before starting an action/changing status. Additionally, the app reminded individuals to track the labels if they have not done so in a while. In addition to this set, there exists an expanded set of labels (non-mutually exclusive) that encompass complex actions (e.g. cleaning, playing basketball) and locations (e.g. at work, at school). This dataset is ideal for our work because it excellently reflects the real-world IoT setting. The data was collected in-the-wild, thus guaranteeing naturally occurring heterogeneities among different clients: individuals have different devices (sensor heterogeneity), different behavior (means of performing actions or being in a certain body state), different habits (certain individuals tend to run more, while others cycle more). The non-IID property of

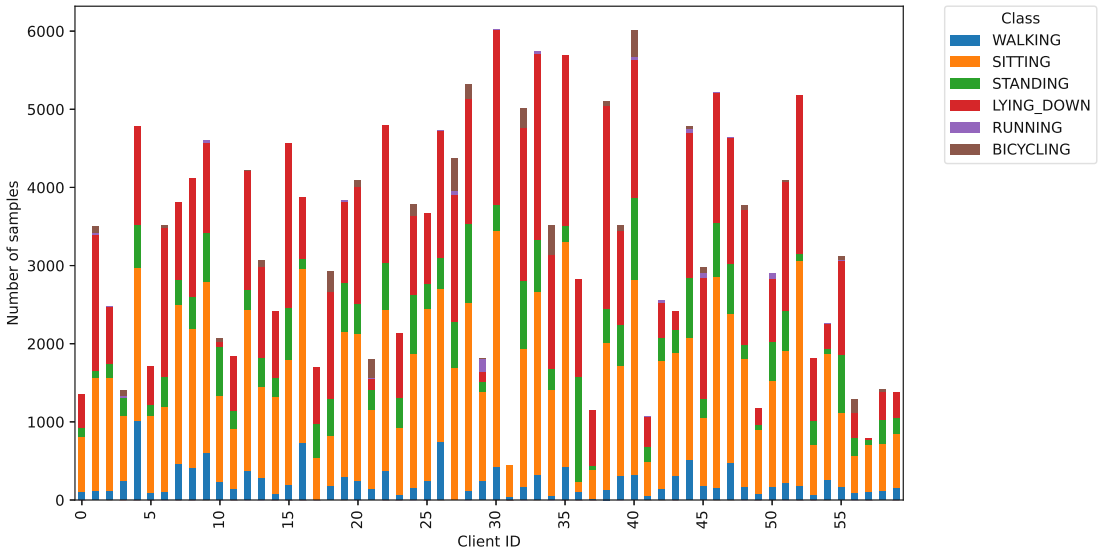


Figure 5.1: Stacked bar plot of label counts for each client. For each client (x-axis) the height of a bar (y-axis) indicates how many samples of the given label (color) are present in that client.

data is simulated in most federated learning research [44], whereas we use a real case. Furthermore, only a few authors [65, 66, 15] evaluated their FL adaptations with the Extrasensory dataset, however, without a detailed analysis.

5.1.1 Dataset Characteristics

The dataset illustrates different types of skew among the clients, namely, *label*, *quantity*, and *feature* skews [56]. Label and quantity skew among the clients is visualized in Figure 5.1, where we can observe how many samples of each label (class) each client has. We observe that *sitting* and *lying down* are the most common, while *running* is very scarce. It is clear that label distributions as well as the total number of samples significantly vary across the clients. Figure 5.2 depicts the quantity and label skew more compactly, through a violin plot. The x axis reports each activity, while the y axis shows how many occurrences of that activity are present in each client. The width represents the commonness of having y occurrences for that activity in a client. E.g., if 20 clients have roughly 3000 sitting samples, then for the sitting violin plot, at y = 2000 large violin width is expected. This plot allows us to see how certain activities are underrepresented, especially running and bicycling, as well as, how each activity is unequally distributed across clients (violin width). These label and data quantity skews are further complicated by the natural feature skew that is present in the HAR datasets [8] due to individual behavioral patterns expressed by humans, leading to the *variability of data representing the same activity* as well as making similar activities even harder to distinguish. This combination of several data skews materialized in a real-life

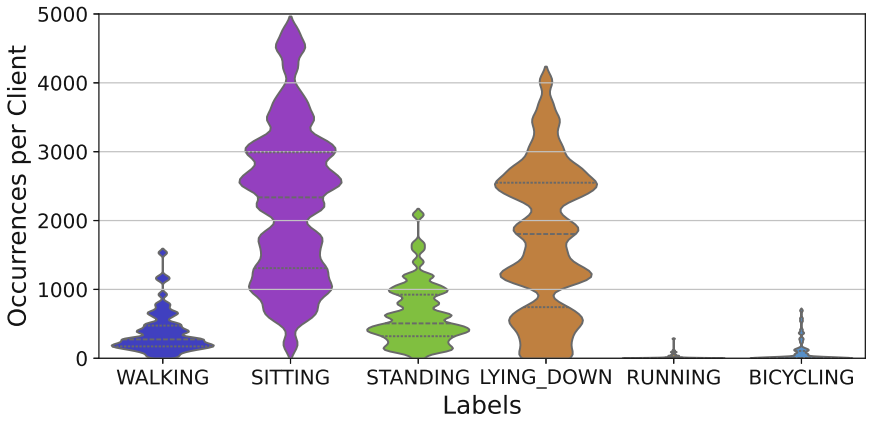


Figure 5.2: Violin plot highlighting the data and label skew. We can see how certain activities are underrepresented, as well as, from the violin *width*, how each activity is not equally distributed across clients.

IoT scenario makes the Extrasensory Dataset a challenging yet valuable candidate for testing solutions for heterogeneous FL.

5.2 Data Preprocessing

In the following the data preprocessing steps described in Sections 4.1.1 and 4.2.2 of the proposed methodology are described.

5.2.1 Data partitioning

As the data is originally partitioned by individuals we proceeded with this predefined split among the 60 clients. Each client's local dataset was split into three parts following the 64-16-20 % shuffled split among train, validation and test sets (applying the 80-20 rule consecutively). Each client's test set was sent to the server to create a *fair* centralized test set (CTS) for evaluation. In the CL setting the local train and validation sets were merged into the respective centralized train and validation sets.

5.2.2 Feature selection

For our experiments, we focused on the original set of 6 labels: *walking, sitting, standing, lying down, running, bicycling*. Figure 5.3 illustrates how many missing values are to be expected for each sensor in the worst case. We discarded the sensors with more than 60% of missing values in any feature belonging to the features of the respective sensor. We further discarded the magnetometer sensor to make the model input smaller without significantly impacting the performance. The final sensors used for our models are: accelerometer (26 features), gyroscope (26), watch accelerometer (46), watch compass (9),

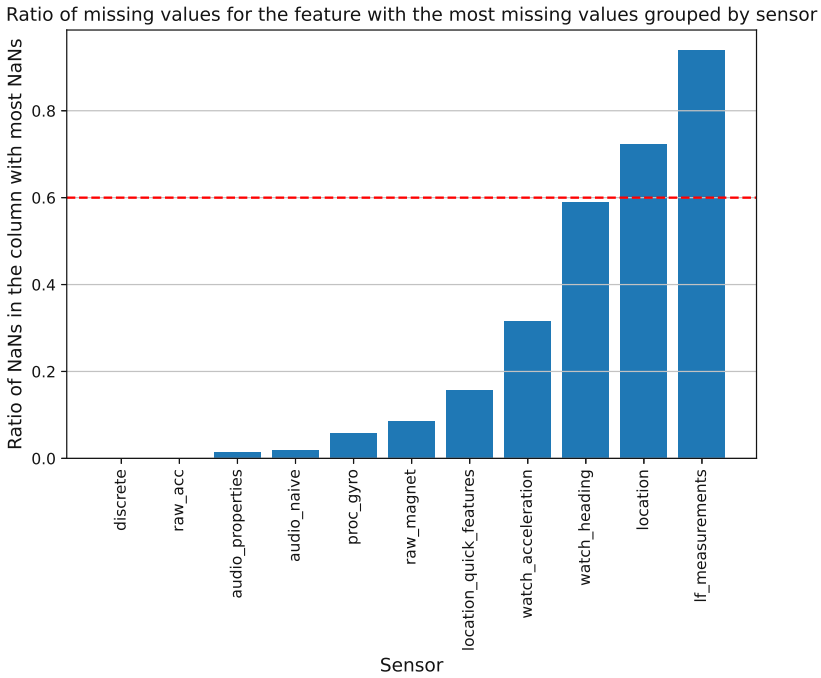


Figure 5.3: Ratio of missing values for a column (feature) with the most missing values grouped by sensor. E.g. the feature with most missing values within the watch accelerometer sensor features has approximately 30% missing values.

audio (26), audio properties (2) and phone state as one hot encoded discrete measurements (32), resulting in 175 input features in total. This subset of sensors was selected to avoid handling too many missing values from other sensors and to keep the input dimension smaller.

5.2.3 Standardization & Cleaning

In our preprocessing pipeline, we apply *global standardization*, a process in which feature means and standard deviations of all clients are sent to the server in order to create a globally scaled view of the data. We impute the missing values with the feature means.

5.2.4 Data Augmentation

As the classes are already severely imbalanced without any augmentation (See Figures 5.1 and 5.6), we employ and evaluate two different data augmentation strategies *after* standardization and missing value imputation.

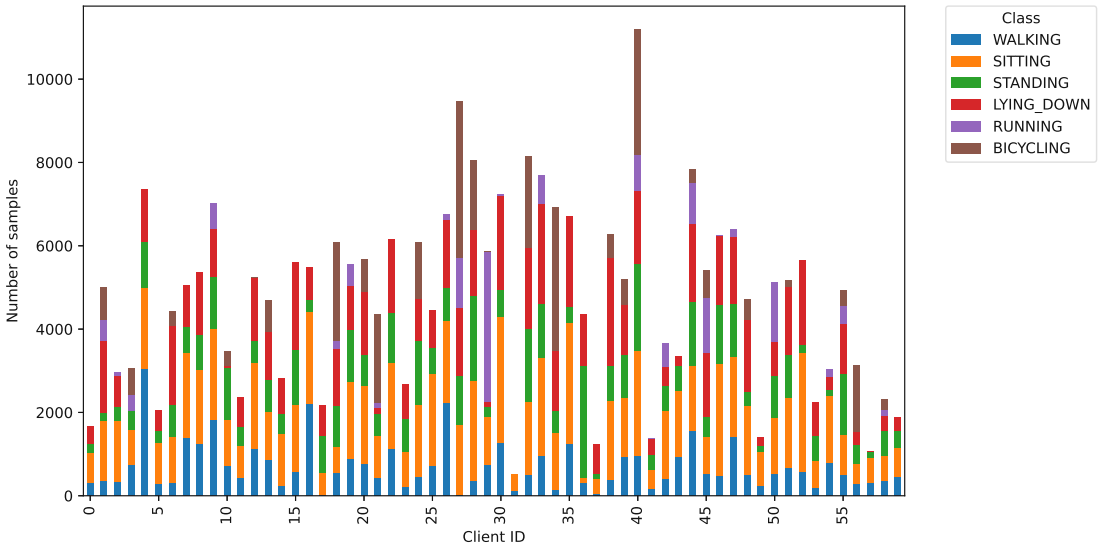


Figure 5.4: Stacked barplot of the per-client label distributions with *base* data augmentation setting.

Base Augmentation

We extract and replicate samples of each class an empirically defined number of times (Running 20 times, Cycling 8 times, Standing 1 time, Walking 2 times). These values aim to improve the representation of severely underrepresented classes, such as *running* and *cycling*, more compared to the less underrepresented classes *standing* and *walking*. The representation of a class can be extracted from the height of the violins in the violin plot illustrated in Figure 5.2.

After extracting the replicas, we use Gaussian noise (Mean 0 and std 10^{-4}) to augment the features of each replica. We call this augmentation strategy *"base'"* and the resulting per-client label distributions are presented in Figure 5.4. We observe that the *running* class is significantly better represented in those clients who possess it.

Balanced Augmentation

We also examine the *balanced* augmentation setting; here, all existing labels are balanced *on each client separately*. In this case, the number of replicas separately created for each sample dictates the balanced augmentation result. This number is the ratio of the number of samples of the most common label over the number of samples of the currently augmented label. (e.g. if sitting is the most common label on the client i and has n_s^i samples, then the number of replicas of the running samples on the same client will be $\lfloor \frac{n_s^i}{n_r^i} \rfloor$. To make the process more understandable, we visualize it in Figure 5.5. In the plot we see that each client now has approximately a balanced distribution of all classes that are available to that client.

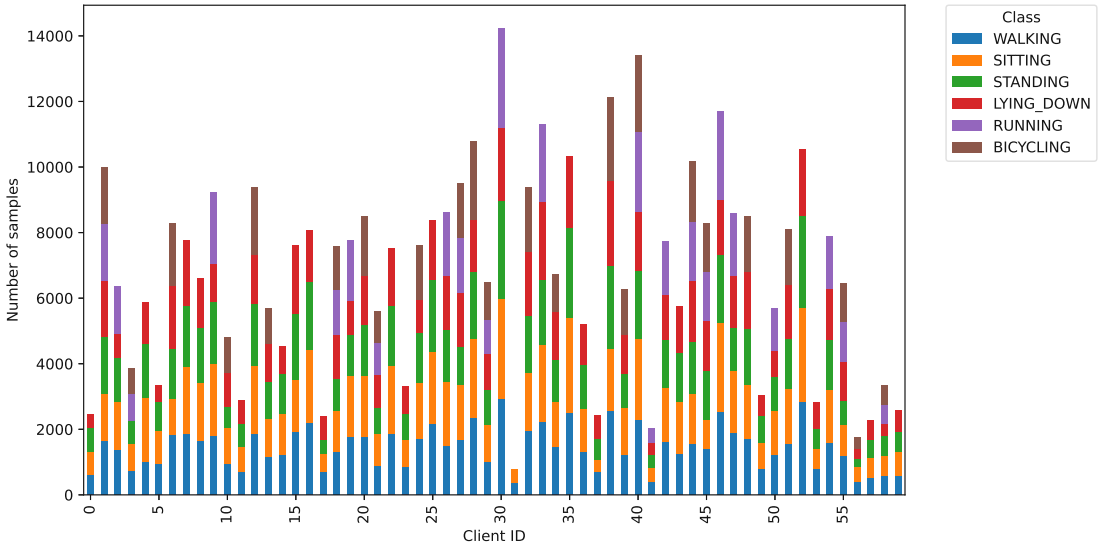


Figure 5.5: Stacked barplot of the per-client label distributions with *balanced* data augmentation setting.

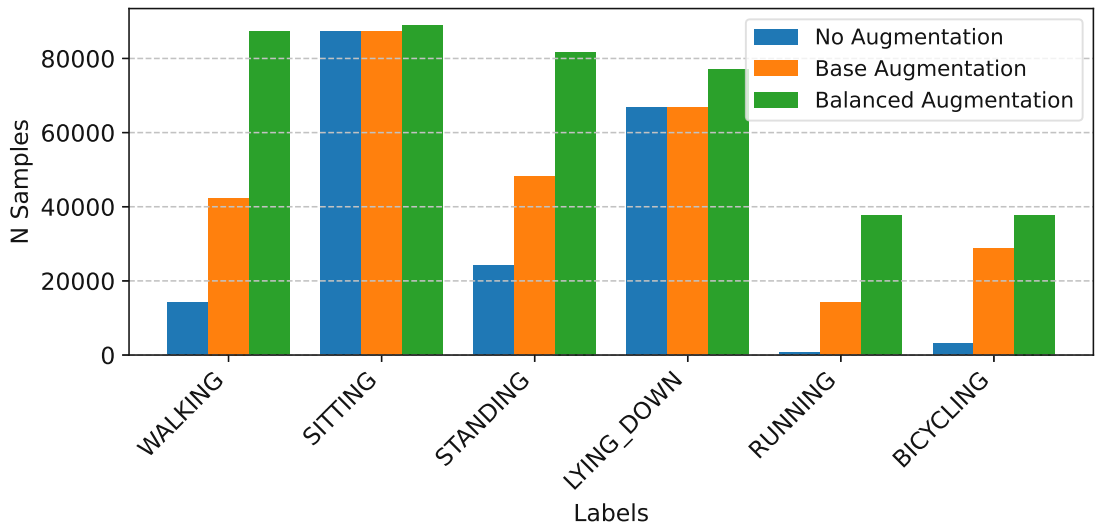


Figure 5.6: Label distribution with different data augmentation settings: *none*, *base* and *balanced*. Note that in the balanced setting *running* and *cycling* are still *globally underrepresented*.

The resulting label distributions are depicted in Figure 5.6, which shows how, previously negligible labels, like *running* and *cycling* have a better representation and better compensate for the data skew. In particular, the balanced augmentation brings labels like *walking* and *standing* to have almost the same volume as the most prominent ones, i.e., *sitting* and *lying down*.

5.3 Quality Assessment Metrics

The next essential step is to define a clear set of quantitative measures of the model's quality. In particular, we want metrics that aid the understanding of the goodness of each model in their training phase while being general enough to allow the comparison across CL and FL implementations. Therefore, we choose the following set of metrics:

1. *Balanced Accuracy (BA)* - a commonly used metric in label-imbalanced settings [50]. It is defined as the macro-averaged recall across all labels.

2. *Macro-averaged F1-Score* - similarily to BA, in class-imbalanced settings, this metric can assess the model's predictive power across all classes, without the bias toward the majority class introduced by the label imbalance.

3. *F1-Score* on the *minority class (running)* - showcases the model's capability to predict severely underrepresented classes.

4. *F1-Score* on the *majority class (sitting)* - shows how the model works when there is enough information.

We use these metrics for tuning, comparing and selecting the best setting for our CL and FL models. Furthermore, they work as a reference during the evaluation of the design decisions, in Chapter 6.

5.4 Model Design & Tuning

We model the problem with a multi-layer perception (MLP) with 64, and 16 neurons in each layer. *Leaky ReLU* was used as the activation function of the hidden layer and *softmax* as the output activation. The model architecture (activation functions and number of neurons) is inspired by the work of the Extrasensory dataset authors [50], allowing us to have a direct comparison. The main extension to their work is that we perform multi-class classification on the mutually exclusive labels, whereas they focus on multi-label distribution on many different subsets of labels and also experiment with different layer/neuron number combinations. As we employ softmax as the activation, we use *categorical cross entropy* as the loss function. Stochastic Gradient Descent (SGD) was used as the optimizer used with momentum set to a commonly selected value of 0.9.

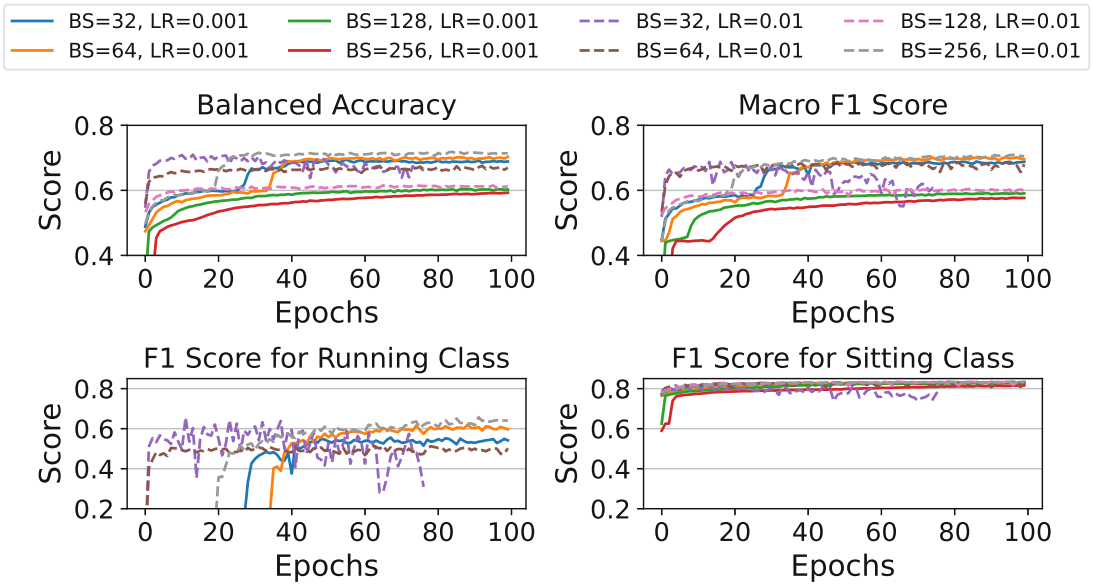


Figure 5.7: Zoomed-in convergence plots of the evaluation metrics for different batch sizes (BS) and learning rates (LR) in **centralized learning**.

First, we focus on the CL model where we tune the batch size and learning rate to find the optimal configuration, using the *base-augmented* dataset. We evaluate the model with the following hyperparameter grid: *batch size* $\in \{32, 64, 128, 256\}$ and *learning rate* $\in \{0.01, 0.001\}$. We monitor the *convergence* as well as the performance of the models on the *fair test set* using the metrics defined in Section 5.3. We display the convergence of the four relevant metrics in Figure 5.7 and present the average of the last 5 epochs for each of the four metrics in Table 5.1.

The four subplots in Figure 5.7 show on the x axis the training epochs, and on the y axis the score for the observed metric. Going left-to-right from the top row, we can inspect the results for BA, $Macro\ F1$, $F1\ Score$ for $Running$ ($F1_{running}$), and $F1\ Score$ for $Sitting$ ($F1_{sitting}$). At a glance, the results indicate a plateauing of the metrics improvement between the first 20 to 40 epochs. Overall, the $[BS = 256,\ LR = 0.01]$ configuration outperforms the others on almost all the metrics. These results tell us that the CL model can accurately perform HAR, especially with the aforementioned configuration, reaching up to ≈ 0.7 for BA and $Macro\ F1$ and ≈ 0.64 on $F1_{running}$. We can also see how, as expected, the larger data volume for the *sitting* label leads to better classification of that label ($F1_{sitting} \approx 0.82$), especially if compared to the underrepresented *running* class ($F1_{running} \approx 0.65$).

As a result of multiple preliminary experiments, our configuration for synchronous FL (SFL) is as follows: we train each model for a maximum of 100 rounds with 2 local epochs on all 60 clients, as the FL model requires, overall, more epochs for converging than in CL. We varied the number of clients per round $S \in 20, 40, 60$ and found that they all

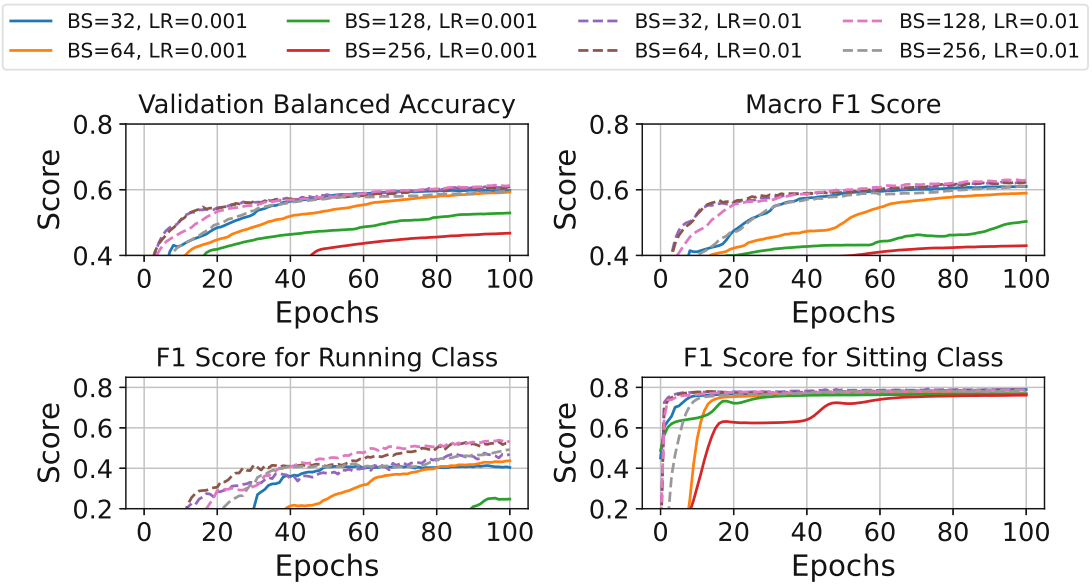


Figure 5.8: Zoomed-in convergence plots of the evaluation metrics for different batch sizes (BS) and learning rates (LR) in **synchronous FL**.

achieved comparable performance, where the larger values implied fewer fluctuations in the observed metrics between each round. Therefore we select all 60 clients in each round to make the results comparable to the AFL setting where all 60 clients run continuously. To save computation time, we use early stopping; the training process is aborted if the performance does not improve after 50 rounds. On top of this setup, we tune the BS and LR, using the same values as in CL. The convergence results of different hyperparameter configurations across the four tracked metrics evaluated after each round are presented in Figure 5.8. We may observe that a larger learning rate performs significantly better across BA, $Macro\ F1$ and $F1_{running}$ metrics and that medium batch sizes (64 and 128) are preferred. Table 5.1 contains the average of the four metric values calculated over the last 5 evaluation rounds for each hyperparameter configuration. We can immediately notice that the overall scores for BA and $Macro\ F1$ are lower than in CL (≈ 0.6 vs. ≈ 0.7). This degradation seems to impact the less-represented labels more than the majority labels such as *sitting*, as the F1 Score on the *running* class was lower in SFL than in CL setting by approximately 0.11, while the F1 Score on the *sitting* class reduced only by 0.04.

In asynchronous FL (AFL), for hyperparameter tuning, we train the models for 40 minutes. The goal is to select the model with the largest macro F1-Score as the baseline defined by batch size and learning rate. We, again, plot the convergence results of the four metrics evaluated periodically (every 20 seconds) with a fair test set and these results are illustrated in Figure 5.9. Similarly to SFL, we observe that larger LR yields better results on average across the four metrics with the smallest differences in $F1_{sitting}$ as in

Figure 5.9: Zoomed-in convergence plots of the evaluation metrics for different batch sizes (BS) and learning rates (LR) in **asynchronous FL**.

the previous two settings, CL and SFL. Table 5.1 summarizes the performance of the models trained with different hyperparameters, which are comparable to the SFL tuning results presented in the same Table. As the centralized evaluation happens periodically in AFL the values in the Table represent the mean over the last 5 periodic centralized evaluations. Ultimately, we select the $[BS = 128,\ LR = 0.01]$ configuration as it achieves the best performance across the majority of tracked metrics. Likewise, we performed the experiments for various FedAsync mixing alpha settings $\alpha_{FA} \in \{0.2, 0.4, 0.8\}$. We discovered that this only influences the speed of convergence. Therefore, to reach convergence the fastest, $\alpha_{FA} = 0.8$ was selected.

To summarize the results of hyperparameter tuning across all three paradigms of training: CL, SFL and AFL, we present the scores of each metric averaged across the last five convergence observations (epochs in CL, rounds in SFL and periodic evaluations in AFL) in Table 5.1. CL consistently outperforms both SFL and AFL across all four metrics, while SFL and AFL reach similar performances. A comprehensive discussion and empirical comparison of the established baselines representing the three training paradigms (CL, SFL and AFL) is provided in Section 6.2 of the subsequent Chapter. There the comparability of SFL and AFL is considered together with the extended set of evaluation metrics such as communication costs & efficiency.

	LR				0.001				0.01
	BS	32	64	128	256	32	64	128	256
CL	m-F1	0.69	0.70	0.59	0.58	0.60	0.68	0.60	**0.71**
	BA	0.69	0.70	0.60	0.59	0.66	0.67	0.61	**0.71**
	F1-R	0.54	0.60	0.00	0.00	0.45	0.49	0.00	**0.64**
	F1-S	0.83	0.83	0.83	0.81	0.77	0.82	0.83	**0.83**
SFL	m-F1	0.61	0.59	0.50	0.43	0.62	0.62	**0.63**	0.61
	BA	0.60	0.59	0.53	0.47	0.61	0.61	**0.61**	0.59
	F1-R	0.41	0.43	0.25	0.00	0.46	0.52	**0.53**	0.49
	F1-S	**0.79**	0.77	0.77	0.76	0.79	0.79	0.79	0.78
AFL	m-F1	0.58	0.56	0.46	0.44	0.62	0.62	**0.63**	0.61
	BA	0.59	0.56	0.50	0.47	0.60	0.61	**0.61**	0.60
	F1-R	0.36	0.29	0.00	0.09	0.48	0.46	**0.54**	0.47
	F1-S	0.79	0.77	0.77	0.76	**0.79**	0.79	0.79	0.79

Table 5.1: Summary of the batch size (BS) and learning rate (LR) hyperparameter tuning for the three settings: Centralized Learning (CL), Synchronous Federated Learning (SFL) and Asynchronous Federated Learning (AFL). The monitored metrics are macro-averaged F1 score (m-F1), balanced accuracy (BA) and F1 score on the underrepresented running class (F1-R). As the performance on F1 score on the sitting class did not significantly vary, it was omitted from the table. In the CL setting, the values are calculated as the mean over the last 5 epochs, while in SFL, they represent the mean over the last 5 rounds, and in AFL, they correspond to the mean over the last 5 periodic centralized evaluations.

CHAPTER 6

Evaluation

In this section, we evaluate the challenges and complications that may arise when transitioning from CL to FL models in non-IID settings. We first describe our evaluation framework together with a brief implementation description of our AFL solution. We then provide the performance and communication efficiency comparison of CL, SFL and AFL baselines in the context of the Extrasensory use case. We group the relevant design decisions, presented in Chapter 4, into two groups: 1) data-related decisions and 2) system/model-related decisions and discuss their effects and implications on FL with non-IID data based on empirical measurements.

6.1 Evaluation Testbed Setup

We need to consider various aspects when setting up the *evaluation testbed*. First, we need to guarantee that the results produced for AFL are comparable with the other methods and other AFL variants. To do so, we:

1. establish the final performance metrics scores after a fixed number of updates per client (instead of a fixed training duration) and
2. consider two evaluation vantage points (VP):
   a) *central* - In this VP, we perform the evaluation on the centralized test set on the server. In synchronous settings, this step is performed before starting the next round, while in AFL, we perform it periodically (every 20s). All of the results presented in this evaluation relate to this, *central* VP.
   b) *distributed* - In the this VP, the evaluation happens locally on each client before the local training starts. In this case, we use the local client validation set. As the results of this VP are usually skewed towards the client's own distribution, they are not discussed further in our evaluation.

Finally, we need to specify the *performance metrics*. As model quality indicators, we leverage the metrics defined in Section 5.3, i.e., BA, $Macro\ F1$, $F1\ Score$ for *Running*, and $F1\ Score$ for *Sitting*. Furthermore, we introduce a FL-specific metric, i.e., *communication cost*. This metric measures the number of model transfers between the clients and the server; specifically, it evaluates how many times the model has iterated over the samples.

6.1.1 Asynchronous Flower

To distribute the training process in a federated manner we leverage the Flower framework [14]. One major shortcoming of this framework is the absence for AFL support, hence we implement this extension. We modify the thread pool executor to prevent the server from waiting for clients to finish training. Instead, a callback is triggered upon arrival of each client result that updates the global model according to Equation 4.2. If the maximum train duration is not exceeded it re-submits the client for training with the updated global model. This adaptation is transparent for the participating clients. The server still contains one loop that periodically executes central evaluation until the maximum training duration is exceeded. Our implementation of AFL is available in the following GitHub repository [1]. A more extensive discussion of the Flower framework and our AFL implementation is provided in Chapter 7.

For the *execution setup*, we simulate the clients and the server by running a ray simulation provided by the Flower framework [14]. We perform our experiments on a Ubuntu 24.04 LTS VM with the AMD EPYC 7742 64-Core CPU and 384GB of memory.

6.2 Empirical comparison of the CL, SFL and AFL baselines

We now compare the performance of CL, SFL and AFL across the four evaluation metrics: macro F1 score, balanced accuracy, F1 on running and F1 on sitting. Additionally for FL settings we track the number of communication rounds (NCR), total train time (TTT) in seconds and rounds per hour (RPH) to evaluate their communication efficiency. To make the results of AFL more comparable to SFL we introduce shortened versions of the AFL experiment: AFL-T and AFL-C. The former ensures that AFL-T and SFL have approximately the same TTT, while the latter ensures approximately equal NCR. Results of this evaluation are presented in Table 6.2 and visualized in Figures 6.1 (four quality assessment metrics) and 6.2 (three communication efficiency metrics).

The hyperparameters of the three evaluated baselines are stated in Table 6.1 again, for convenience. The test set was built using the *fair centralized test set* strategy (FTS), the features were scaled *globally* for all three baselines and *SGD-m* was used as the optimizer with a momentum of 0.9.

[1] `https://github.com/r-gg/flower-async`.
A pull request to merge our approach into the Flower framework is currently under review.

Baseline	LR	Batch Size	Paradigm Specific Parameters
CL	0.01	256	Epochs: 200
SFL	0.01	128	Rounds: 100, Clients per round: 60 (all), Local Epochs: 2
AFL	0.01	128	Local Epochs: 2, Max. train time : 40min, Mixing ratio: 0.8

Table 6.1: Overview of the hyperparameters of the selected baselines

Figure 6.1 illustrates the performance of the three training methods across the four quality assessment metrics. We observe that, naturally, CL performs best across all metrics. SFL, AFL and AFL-T have similar performance while AFL-C falls behind, implying that by fixing the number of communication rounds, SFL will perform better than AFL. This is expected, as upon each global model update SFL averages the updates from all clients, while in AFL each global model update has a very limited view of the problem (i.e. only one client's model update). We observe that if training time is equal (AFL-T) then both SFL and AFL perform similarly.

Further, in Figure 6.2 we observe the communication cost and efficiency of SFL and AFL. CL naturally has the lowest runtime as there is no communication and aggregation overhead. This plot is a very illustrative example of the primary benefits of AFL: by fixing the train time (AFL-T) we observe a larger number of communication rounds in AFL. This happens because AFL does not wait for slow clients (stragglers) contrary to SFL. As a consequence, the number of rounds per hour increases as well.

Takeaway: *CL offers the best performance across all metrics while compromising users' data. SFL and AFL have comparable performance if trained for an equal amount of time. AFL requires more rounds to reach the same performance compared to SFL but compensates for this issue with a larger rounds per hour rate.*

6.3 Data-related Decisions

We now evaluate the effects of data preprocessing decisions and which challenges may arise when transitioning from CL to FL in non-IID settings.

6.3.1 Fair vs Hold-out Test set

As discussed in Section 4.2.1 there are several ways of generating the test set in federated learning settings with IoT data, i.e., with multiple clients. This decision significantly impacts the semantics of evaluation as well. While distributed evaluation (DE) offers a way to privately evaluate model performance on the participating clients considering all label distributions, it poses the same issue as training with non-IID data. More precisely, the evaluation on individual clients is biased towards the client's own feature and label

	m-F1	BA	F1-R	F1-S	NCR	TTT (s)	RPH
CL	**0.71**	**0.71**	**0.64**	**0.83**	-	512	-
SFL	0.63	0.61	0.53	0.79	100	2298	157
AFL	0.63	0.61	0.54	0.79	141	2400	**212**
AFL-T	0.61	0.63	0.53	0.79	134	2301	210
AFL-C	0.59	0.61	0.47	0.78	100	1707	**212**

Table 6.2: Comparison of the CL, SFL and AFL baselines across four quality assesment metrics: macro-averaged F1 (m-F1), balanced accuracy (BA), F1 on *running* class (F1-R), F1 on *sitting* class (F1-S) and three communication efficiency related metrics: average number of communication rounds (NCR), total train time in seconds (TTT (s)), number of communication rounds per hour (RPH). AFL-C denotes the scores of the AFL baseline with a convergence cutoff to ensure the equal number of communication rounds as in SFL. Column wise "best" values are bolded for all columns except NCR and TTT as these are fixed for SFL (NCR is fixed) and AFL (TTT is fixed).

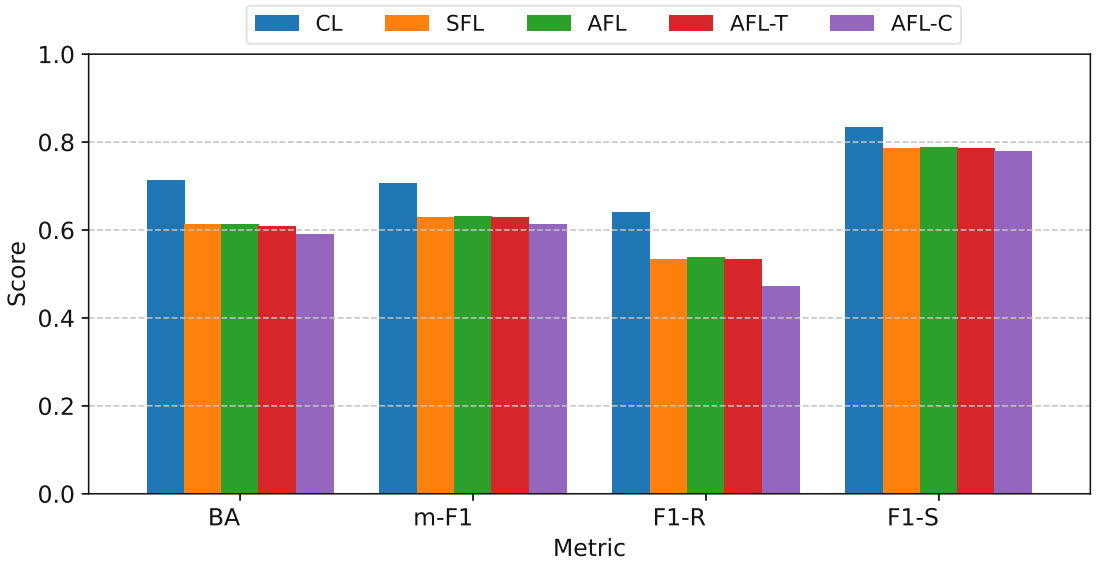


Figure 6.1: Visual comparison of CL, SFL and AFL baselines across four quality assesment metrics: macro-averaged F1 (m-F1), balanced accuracy (BA), F1 on *running* class (F1-R), F1 on *sitting* class (F1-S). AFL-C denotes the scores of AFL baseline with a convergence cutoff to ensure the equal number of communication rounds as in SFL.

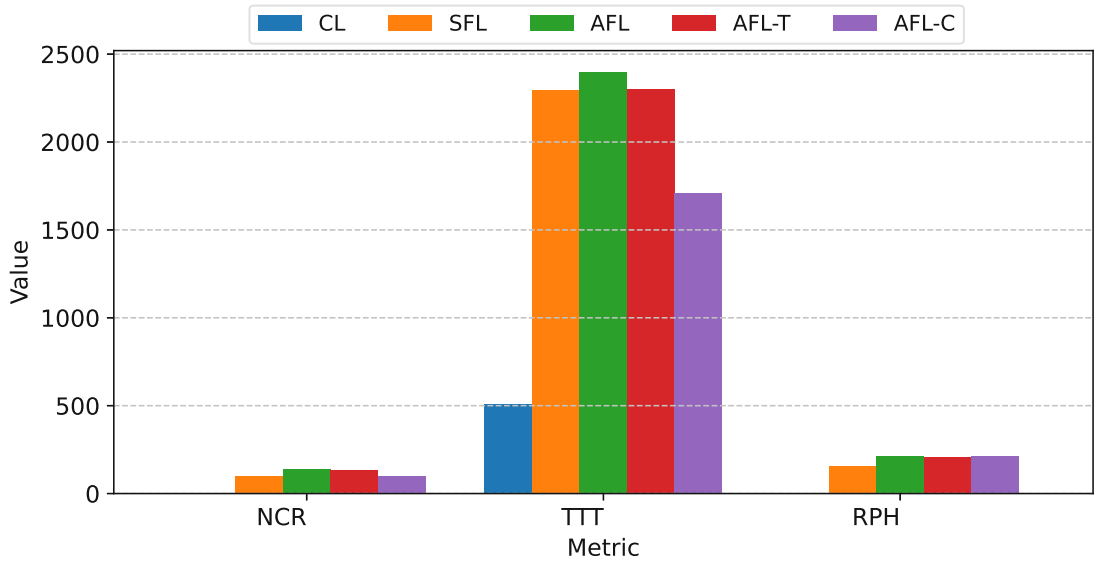


Figure 6.2: Visual comparison of CL, SFL and AFL baselines across three communication efficiency related metrics: average number of communication rounds (NCR), total train time in seconds (TTT (s)), number of communication rounds per hour (RPH). AFL-C denotes the scores of AFL baseline with a convergence cutoff to ensure the equal number of communication rounds as in SFL.

distributions. Without the disclosure of label distributions, the aggregation of evaluation scores returned from clients after DE becomes non-trivial and using Federated Average for this purpose is not effective for non-IID data.

Therefore, we decided to evaluate the two remaining ways of test-set generation:

1. **Hold-Out Clients** (HOC), which reserves all samples of a *subset of clients* only for testing

2. **Fair (Centralized) Test Set** (FTS), where *each client selects a (fixed) portion* of data to send to the server to create the test set. This can be seen as a variant of the *Centralized Test Set* (CTS) approach introduced in Section 4.2.1 because there are multiple methods of making the centralized test set, for instance, the "golden dataset" could be used, which does not contain samples from clients participating in training but rather external data.

While *HOC* offers better privacy guarantees, the generated test set will be biased toward the distributions of selected clients (this is also referred to as *client-selection bias*). This effect is even more prominent in severe non-IID settings, due to high levels of quantity, feature, and label skews. On the other hand, *FTS* carries more privacy risks but enables

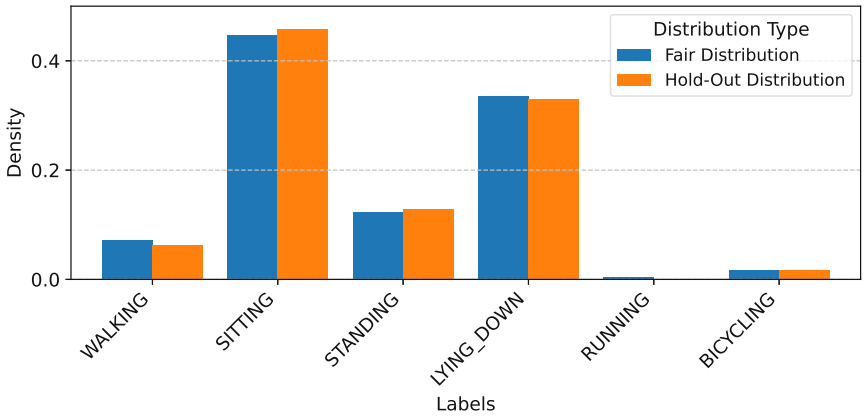


Figure 6.3: Label distribution in the fair vs hold-out test set.

a fairer evaluation of the global model performance by sampling data (distributions) from all clients.

Our work targets **fair examination** of the models; therefore, we decided to use *FTS* for all our experiments. Especially considering our use case, the skew in label distribution is so accentuated that with *HOC*, a subset of clients could be produced that does not contain any samples from the minority class, leading to a partial evaluation. In Figure 6.3 we observe that *running* label is severely underrepresented in both types of test set generation, however, in case of *HOC* the class *running* is **not present** in the test set.

Takeaway: *The client-selection bias present in HOC is amplified by the high levels of heterogeneity typical for IoT data. FTS can be used to gain a clear and stable view on the global data distribution enabling fair evaluation while carrying privacy risks.*

6.3.2 Data augmentation and its effects in non-IID FL

Here we re-visit augmentation schemes from Section 5.2.4 and consider them within the *federated learning context*. Figure 6.4 depicts the convergence results of training the AFL baseline model with varied data augmentation. Likewise, a summary of the effects of different data augmentation schemes for CL, SFL, and AFL is presented in Table 6.3. The shorter lines in this and the following Figures are the result of the evaluation approach introduced in Section 6.1, where the number of communication rounds is fixed instead of the total train time. For instance, in Figure 6.4 there are 108 communication rounds per client (as visible in the legend title).

Across all three settings, the *running* label is almost always ignored by the models if no augmentation is performed. The model trained without augmentation performs comparably on the majority label *sitting* as the one trained with base augmentation because sitting is the most common class. On the other hand, excessive augmentation can be problematic. The *balanced* setting, despite equal class representation, introduces

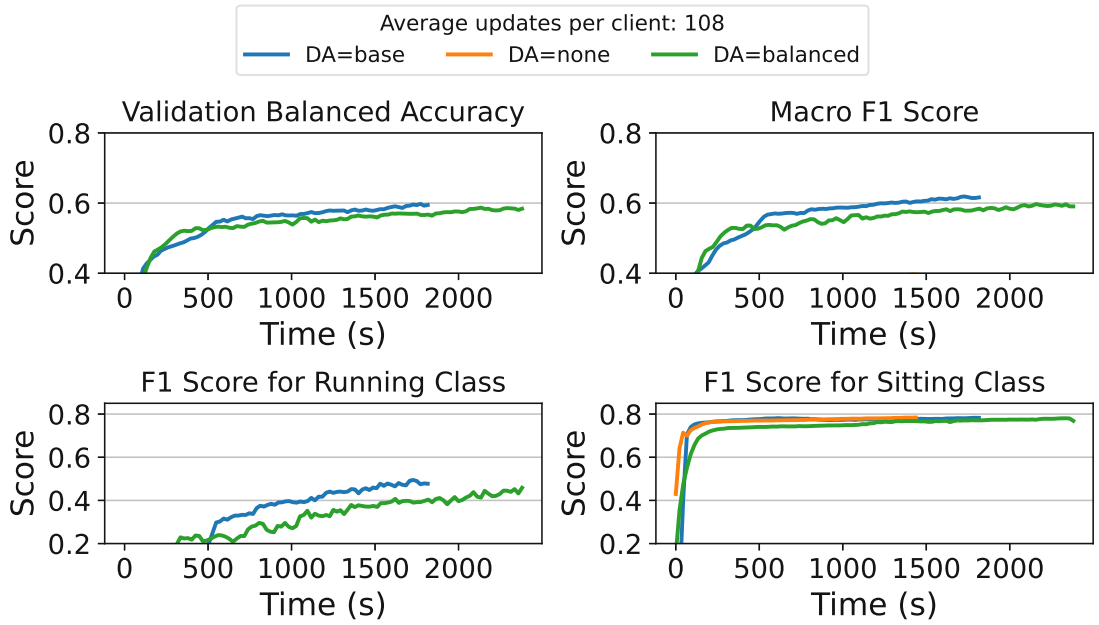


Figure 6.4: Zoomed-in convergence plots of the evaluation metrics for different data augmentation schemes (DA) in asynchronous FL. Different line length is the result of fixing a number of average client updates rather than train time.

the *oversampling bias*, making it perform better compared to no augmentation but worse than base augmentation.

Takeaway: *Adding Gaussian noise as data augmentation scheme partially addresses the label skew in CL, SFL, and AFL. The magnitude of the performed data augmentation is a tunable parameter, as excessive augmentation can lead to oversampling bias and no augmentation leaves the minority classes severely underrepresented and, consequently, ignored by the model.*

6.3.3 Global Data Scaling and Persistence of Feature Skew

We further evaluate the influence of two data scaling (standardization) methods (introduced in Section 4.2.2) on model convergence in FL with non-IID data. In *local standardization*, the features are scaled based on **client-local** feature means and standard deviations. In contrast, *global standardization* consistst of first sharing the local feature means and standard deviations with the server before training, and then scaling the data on all clients with the aggregated **global** means and standard deviations. While the latter carries privacy risks due to the communicated statistics, it improves the performance of the models across all four tracked metrics, as it is visible from the Figure 6.5 and Table 6.3. Individuals' personal habits and way of performing them influence their local feature distributions (e.g. every individual runs differently and introduces different sensor

Figure 6.5: Zoomed-in convergence plots of the evaluation metrics for different feature standardization schemes (STD) in asynchronous FL.

readings for this class). Secondly, additional feature skew among the clients stems from *device heterogeneity*. As a result, scaling the features *locally* tends to keep a considerable portion of the already present feature skew and creates skewed data views that are confusing for the model and, consequently, degrade its performance.

Global scaling addresses these causes for feature skew, however, we found that a significant amount of feature skew remained even after global scaling, especially for the *running* class samples. Figure 6.6 depicts the feature skew of *running* samples across 5 clients (accelerometer sensor features are considered). It is visible that client 18 and even client 3 have significantly different "views" of the *running* class according to their accelerometer feature distributions compared to clients 2 and 9. This issue is, therefore, still preventing the federated global model from progressing effectively and reaching a performance comparable to CL. We believe that this skew stems from the difference in local *label distributions*. As each client has a different ratio of the samples of each label, feature means and standard deviations will be skewed towards each client's label distribution *as well*. Moreover, due to the minority label having fewer samples per client, the effects of such skews are more prominent for such labels.

Takeaway: *Scaling the data globally, significantly improves the convergence of the models, but it introduces privacy risks. Even with global data scaling, the clients might still have different representations of the same class. This proves that feature skew is a persistent issue in non-IID IoT (HAR) datasets and is especially prominent in minority classes.*

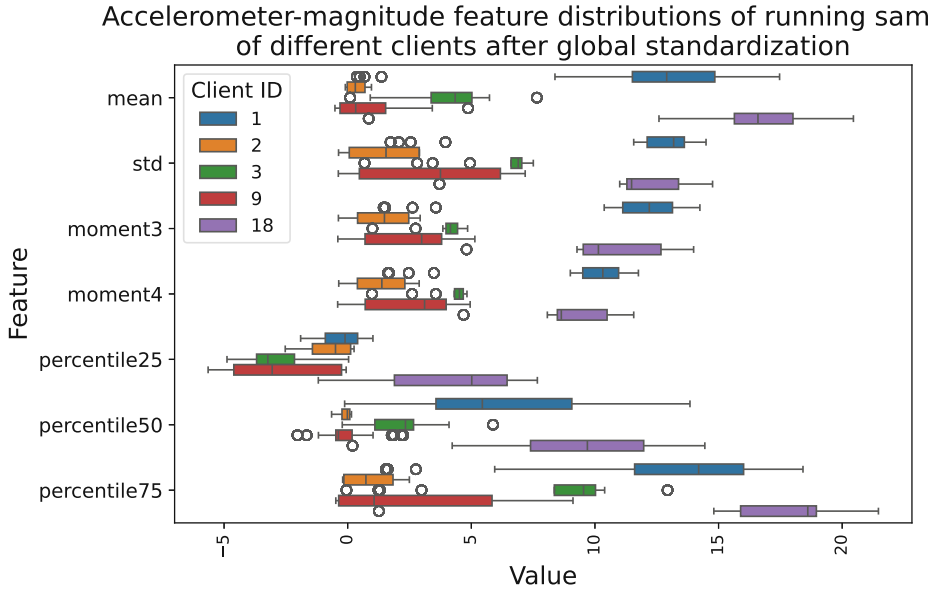


Figure 6.6: Boxplots depicting features distributions of the *running* class samples for different clients (varied by color) [Sensor: accelerometer-magnitude]

6.4 System & Model-related Decisions

In the following, we focus on the effect of different optimizers on model performance with non-IID data and the implications of server-processing delays on AFL with non-IID data.

6.4.1 Optimizer Selection

Here, we compare two popular model optimizers for Deep Learning: **ADAM** [67] and **SGD with momentum** (SGD-m). To examine the magnitude of the optimizer's impact on the model performance, we fix the remaining training parameters. We present the results of this comparison for AFL in Figure 6.7. A tabular view of the results for both SFL and AFL is provided in Table 6.3.

We observe that while ADAM converges faster, it has poorer final performance. Moreover, from Figure 6.7 it is evident that training using ADAM has a more unstable behavior compared to SGD-m, especially visible in the case of F1-score for the *running* class (bottom left). We explain the unsatisfactory results with the ADAM optimizer through extremely short local train times (2 epochs only) and the nature of training FL models. The former does not provide ADAM enough time to develop an effective optimizer state, while the latter involves restarting the ADAM state with each round, where the optimizer essentially forgets any progress made.

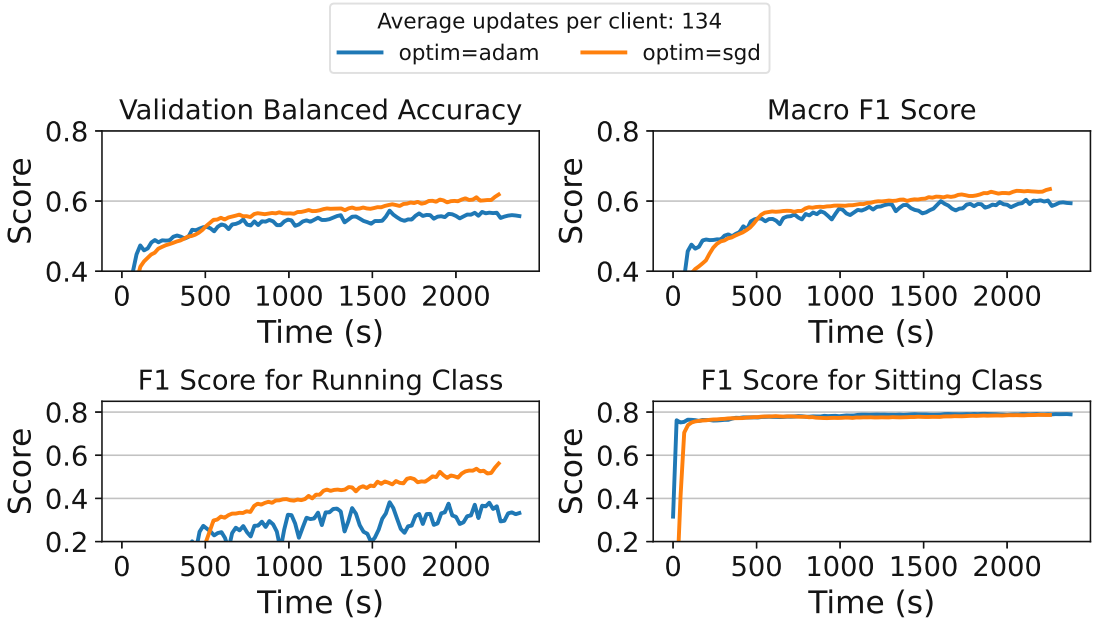


Figure 6.7: Zoomed-in convergence plots of the evaluation metrics for different optimizer (optim) in asynchronous FL.

At the same time, leveraging naïve approaches such as extending the local train time or locally maintaining the ADAM state are not recommended for the following reasons. First, significantly increasing the (local) train time would lead, in FL settings, to divergence of local gradients toward the clients' local distributions, causing issues during aggregation due to large differences among the gradients (See Section 3.2). On the other hand, maintaining the local ADAM state and keeping fewer local epochs carries the risk of ADAM overfitting to the local data. To make the momentum and learning rate adaptive and utilize ADAM's full potential, one could consider either applying ADAM on the server side, as presented in [68, 69], or sharing ADAM state among the clients, as presented in [70].

Takeaway: *Applying SGD-m as the optimizer in FL settings has proven to be more advantageous than applying ADAM. As ADAM contains more stateful parameters that are tracked over multiple epochs, the approach does not perform well in federated optimization which is typically stateless. To improve the performance of ADAM, one can either apply ADAM on the server or share the optimizer's state with the server during the entire training.*

6.4.2 Effects of Busy Servers

Custom FL workflows that adapt the vanilla FL process were discussed in Section 4.2.5. These typically include additional processing steps on the server (e.g. for clustering

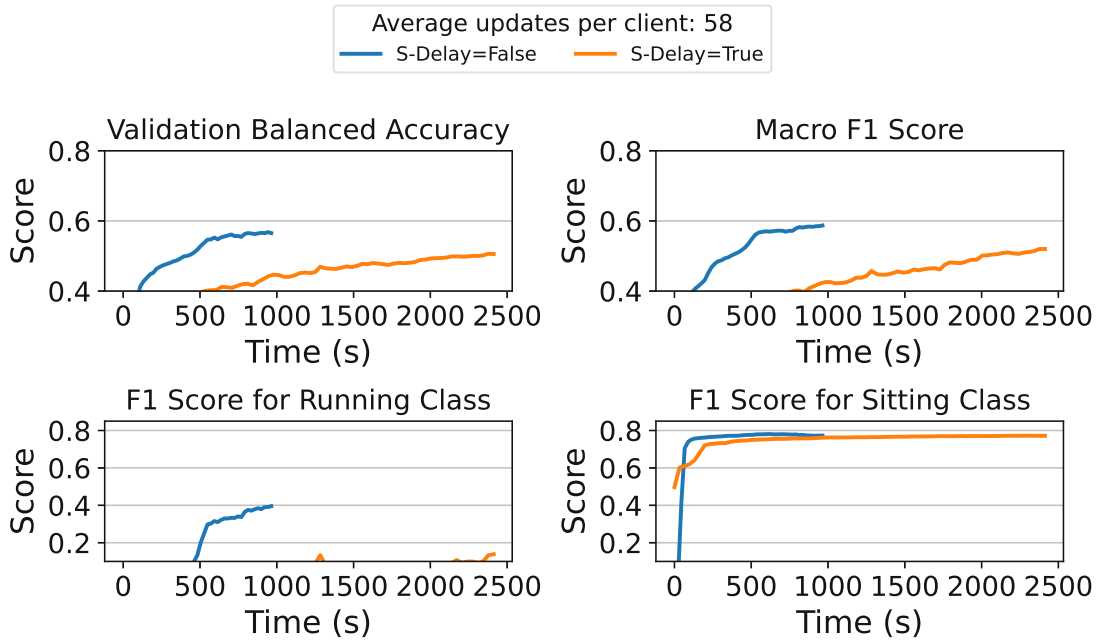


Figure 6.8: Zoomed-in convergence plots of the evaluation metrics for different server-delay (S-Delay) amounts (True if additional delay was introduced.) in asynchronous FL.

[71, 58]) that introduce delays. To examine the effect of these delays on AFL with non-IID data we simulate them by adding busy waiting on the server on two execution points:

1. before centralized evaluation - a busy wait of 10 seconds is added
2. before each client's update is merged into the global model - 1 second of busy wait is added

The results of this run compared to the baseline (without the delays) are illustrated in Figure 6.8 and in Table 6.3. The model trained without the delays naturally reaches the fixed number of updates sooner (hence the shorter line). However, adding delay to the server significantly degrades model performance *even with the same average number of updates per client*. This discrepancy is especially visible in the minority class, *running*.

This observation highlights the importance of timely model updates and underscores the need for adequate server resources and time-efficient adaptation techniques, even without direct model training expected on the server.

Takeaway: *AFL workflows involving intensive and time-consuming computations on the server may degrade model performance even with extended training time. This also raises concerns about the comparability of these approaches with other baselines that do not encompass server delays.*

	Param	data augmentation			standardization		optimizer		server delay	
	Value	none	base	bal.	global	local	adam	sgd	False	True
CL	m-F1	0.47	**0.71**	0.68	-	-	-	-	-	-
	BA	0.46	**0.71**	0.69	-	-	-	-	-	-
	F1-R	0.00	**0.64**	0.60	-	-	-	-	-	-
	F1-S	**0.84**	0.83	0.81	-	-	-	-	-	-
SFL	m-F1	0.51	**0.63**	0.61	**0.63**	0.61	0.59	**0.63**	-	-
	BA	0.47	**0.61**	0.59	**0.61**	0.61	0.56	**0.61**	-	-
	F1-R	0.00	**0.53**	0.49	**0.53**	0.44	0.31	**0.53**	-	-
	F1-S	**0.79**	0.79	0.77	**0.79**	0.78	**0.79**	0.79	-	-
AFL	m-F1	0.39	**0.62**	0.59	**0.63**	0.60	0.59	**0.63**	**0.58**	0.52
	BA	0.39	**0.60**	0.58	**0.60**	0.59	0.56	**0.61**	**0.57**	0.50
	F1-R	0.00	**0.48**	0.45	**0.53**	0.48	0.32	**0.53**	**0.39**	0.11
	F1-S	**0.78**	0.78	0.78	**0.79**	0.77	**0.79**	0.79	**0.77**	0.77

Table 6.3: Summary of the influence of *data augmentation*: none, base and balanced (bal.), *standardization (scaling)* : local and global, *optimizer*: SGD-m and ADAM and the presence of additional server delays on model performance across the four metrics: macro F1 score, balanced accuracy, F1 score on *running* class, F1 score on *sitting* class. Only the federated settings were considered for the effects of data *scaling* and *optimizer* and only AFL for *server delay*. Maximum values for each (*row*, *parameter*) pair are bolded. *For each parameter* (e.g. data augmentation, server delay) equal number of communication rounds was ensured for AFL, hence the variation in "best" results across the AFL configurations. Thus the AFL results reflect the scores after a fixed number of communication rounds and not fixed training time.

6.5 Summary

Table 6.3 provides an overview of the results obtained through empirical evaluation of various parameters including *data augmentation, standardization, optimizer* and *server delay* acoss the three training paradigms: CL, SFL and AFL. From the Table we observe that *base* and *balanced augmentation* provide significantly better results than *no* augmentation. We also observe that *base* augmentation performs better than the *balanced* augmentation. *Global* scaling significantly improves the performance of federated models and this is primarily indicated by better performance on the minority class *running*. As stated in Section 6.3.3 even with *global* scaling, feature skew still persists. We further notice that *SGD-m* optimizer performs significantly better than *ADAM* in both federated settings and that increased server load degrades model performance significantly. Lastly, we find that using a more representative and fairer test set comes with privacy risks.

CHAPTER 7

Framework Implementation

In this chapter, we provide the background and implementation details of the AFL Flower extension with the goal of improving the clarity and reproducibility of the results. Before delving into the implementation specifics, the Flower framework and its main components are summarized first, along with the depiction of a typical SFL process implemented in Flower. We then describe our workflow for enhanced metrics monitoring. Then, we turn to our implementation of asynchronous FL in Flower and discuss the implementation details of the three modified components. Lastly, for the sake of reproducibility, we state the runtime information and external packages used along with their versions.

7.1 Flower Framework

A federated learning framework introduced by Beute et al. [14] has been a very popular choice for a baseline federated learning implementation, especially in academia. It offers user-friendly abstractions that enable quick prototyping. Under the hood, Flower uses the Ray[1] framework [72] for client-server communication. Flower can be deployed in both multi-device and device-simulated settings. In our implementation, we rely on the latter. Flower implements this through its Virtual Client Engine which contains a pool of Ray actors, each representing one virtual client.

Besides the underlying client simulation, Flower also implements a customizable synchronous FL server that carries out all responsibilities of an FL server (starting rounds, monitoring metrics, aggregating the results and general management of the FL process) as well as a multitude of model aggregation strategies, among others FedAvg[4], FedProx[73] and FedOpt[74]. In our work, we primarily use Federated Average (FedAvg) strategy, proposed by McMahan et al.[4]. Lastly, Flower offers a *History* object, containing the

[1] https://www.ray.io/

collected metrics per round, enabling a solid overview of the FL progression and quality. There exist three types of metrics in this context:

1. *centralized* metrics - these are collected during central evaluation (on the server with a dedicated test set). This process happens once per round, after model aggregation.
2. *distributed* (*federated*) metrics - these are collected in separate evaluation rounds similar to training rounds. Instead of training, the clients that are selected *evaluate* the global model on their validation sets and the results of this distributed evaluation are then aggregated into one scalar value per metric per round.
3. *distributed fit* metrics - these metrics are collected directly before/after local training and returned among training (fit) results. They too are aggregated across clients and saved as one scalar value per metric per round.

A sequence diagram depicting one training round of SFL in Flower is provided in Figure 7.1. The *Server* contains the reference to the *Strategy* object, which is first used for sampling of the clients for the upcoming round and making the fit inputs (*FitIns*) for each of the selected clients. *FitIns* object is a tuple of the form $(Model\ Parameters, Config\ Dictionary)$ containing the current global model parameters and a configuration dictionary which can be used to pass additional information to the client. To sample the clients, the *ClientManager* is used. It waits until all clients are available and then selects the subset of clients with a predefined subset size. In the simulation, communication with the clients is carried out via *ClientProxy* objects, more specifically *RayClientProxy* objects. After sampling and definition of fit inputs, these are returned to the server. The *ThreadPoolExecutor* then starts a thread for each $(Client\ Proxy, Fit\ Inputs)$ pair. Each thread calls the `fit()` method on the client proxy with the respective fit inputs. Clients then run local training and each of them returns a fit result (*FitRes*) object that contains a tuple of the form $(Model\ Parameters, Number\ of\ samples, Dictionary\ of\ locally\ tracked\ metrics)$. The server's executor waits until the fit method of each client proxy yields a result, before aggregating the fit results. Fit results are aggregated in the *Strategy* object following the corresponding strategy. Finally, the global model is replaced with this aggregation and evaluated (centrally and/or distributed). The process of distributed evaluation is identical to model training: clients are sampled, the executor calls `evaluate()` function on client proxies, and the evaluation results from all clients are aggregated in the *Strategy* object.

7.2 Augmented Monitoring

The original code of the FL server and clients was further adapted to track several metrics of interest that are not tracked by Flower. These changes encompass tracking round duration, local training time on the clients, and client-measured metrics before and after training. Consequently, these metrics were added to the Flower's History object. All

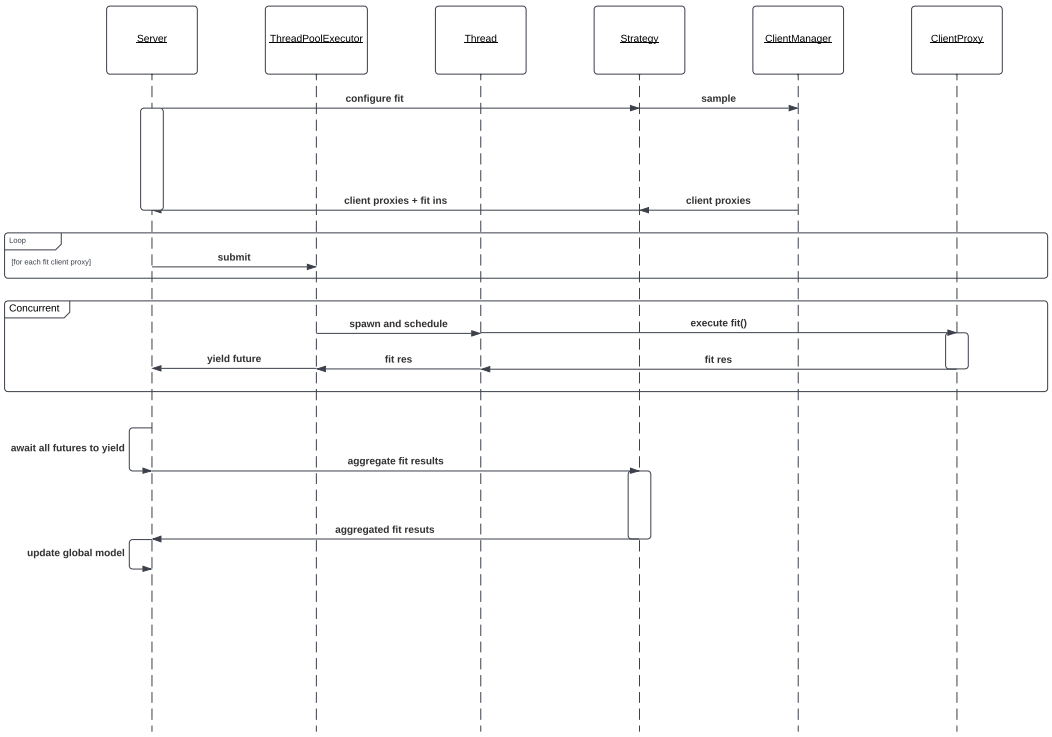


Figure 7.1: Sequence diagram depicting one training round in Flower. More precisely the fitting process. The distributed evaluation process is identical. It should be noted that multiple *Thread* objects are spawned in the concurrent block, one for each client participating in the training round. For the sake of clarity, interactions of only one *Thread* object are represented.

metrics not related to time are collected both in a centralized and distributed-fit manner. To facilitate metrics tracking, we implemented a wrapper around the History object called *Tracker*, which contains information about the configuration that produced the results, i.e. number of clients, data scaling, number of rounds, learning rate, testing set mode, etc. Finally, upon the completion of the FL process, the *Tracker* object is persisted as a `pickle`[2] file that can be easily accessed and used for data analysis and evaluation purposes.

7.3 Asynchronous Federated Learning with Flower

In total, three Flower classes were adapted to enable AFL: *Server, Strategy* and *History*. In the following, we first describe the adaptations of each class and then provide an

[2] `https://docs.python.org/3/library/pickle.html`

example of the usage of these new classes and how the usage differs from the vanilla Flower SFL implementation.

7.3.1 Asynchronous Server

To implement asynchronous behavior, we decided to adapt the use of *ThreadPoolExecutor* in the *Server* class as suggested by the members of the Flower repository in one of the GitHub issues discussing AFL with flower [3]. Namely, instead of synchronizing at each training round, a callback is added for each function submitted to the executor. This callback processes the model update and, if the maximum train time is not reached, submits the same client (*ClientProxy* to the executor once again (with the same callback function). A sequence diagram of the asynchronous workflow is depicted in Figure 7.2. The first step is the same as in SFL: the server fetches the client proxy objects and fit inputs (with the same structure as in SFL) from the *Strategy*. It is important to note that the result contains the client proxies of *all* available clients and not a subset as in SFL. Then, `fit` function of each client proxy is submitted to the executor and our modified *"done-callback"* function is attached to this submission. The executor spawns a thread that executes the `fit` function on the client proxy and awaits the fit result. Upon arrival of the fit result, the model update is merged into the global model and if there is still training time left it is re-submitted to the executor with the new global model parameters as fit inputs. The same function is, again, set as the callback.

Concurrently with these processes (training, merging and re-submitting the fit tasks to the executor) the server periodically (every 20 seconds) evaluates the current global model on a centralized test set.

7.3.2 Asynchronous Strategy

As the previously used *Federated average* strategy was aggregating the models in a round-based manner, it had to be replaced with a Strategy compatible with AFL. The new strategy for merging the incoming update into the global model is implemented according to Equation 4.2 described in the methodology. The update rule described through the equation assumes that model-update *gradients* are communicated from the client by default and not the model *parameters* as in SFL. For the sake of completeness, the *AsynchronousStrategy* optionally allows the model parameters to be aggregated instead of gradients, as described in Equation 4.3.

7.3.3 Asynchronous History

There are two major shortcomings of the built-in *History* object when considering its use in AFL:

1. metrics are recorded on *per-round* level which is not applicable to AFL.

[3] `https://github.com/adap/flower/issues/469#issuecomment-1171340206`

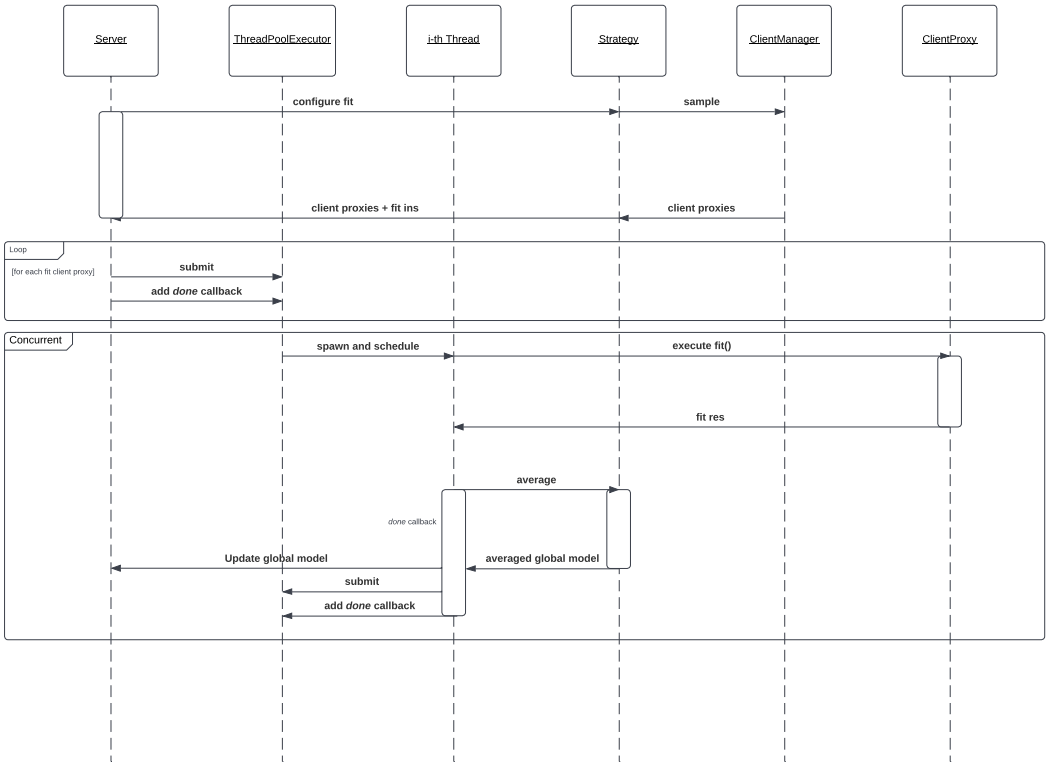


Figure 7.2: Sequence diagram of the implemented asynchronous workflow.

2. distributed fit and evaluation metrics are aggregated across clients and saved *as one scalar per round per metric* which is not applicable to AFL. Through such aggregation, useful insights about *individual* clients' local training are lost.

To address these issues, we first introduce *timestamps* instead of the round counter to track *centralized*, *distributed* and *distributed fit* metrics. Second, we restructure the *distributed* and *distributed-fit* metrics dictionaries to enable the persistence of each client's scores **without aggregation**. These changes are illustrated through a side-by-side example comparison presented in Figure 7.3. On the left (Figure 7.3a) is the structure of the two *History* fields made for SFL, while on the right (Figure 7.3b), we see the adaptations made to enable AFL: *timestamp introduction* and *monitoring metrics per client, without aggregation*.

We also had concerns about the *absence of explicit concurrent-access regulation mechanisms* preventing concurrent writes that might happen in a multi-threaded environment such as AFL. Upon further investigation, we found that all objects (*including dictionaries*) are protected from concurrent modification by Python's *global interpreter lock*[4].

[4] https://docs.python.org/3/glossary.html#term-global-interpreter-lock

```
metrics_centralized: {
    "accuracy": [ (round1, value1
        ) , (round2, value2) ... ]
        ,
    "f1": [ ... ],
    ...
}

metrics_distributed_fit: {
    "accuracy": [ (round1, value1)
        , (round2, value2) ... ],
    "f1": [ ... ],
    ...
}
```

(a) Example structure of the *metrics_centralized* and *metrics_distributed_fit* dictionaries within the built-in *History* object

```
metrics_centralized: {
    "accuracy": [ (timestamp1,
        value1) , ... ]
    "f1": [ ... ],

}

metrics_distributed_fit_async: {
    "accuracy": {
        cid1: [
            (timestamp1, value1),
            (timestamp2, value2),
            (timestamp3, value3),
            ...
            ],
        ...
    },
    "f1" : { ... }
    ...
}
```

(b) Example structure of the *metrics_centralized* and *metrics_distributed_fit* dictionaries within the new *AsynchronousHistory* object. *metrics_centralized* contains central evaluation results that are happening periodically (every 20 seconds) while *metrics_distributed_fit* contains metrics monitored by each client (marked with client id (*cid*)) before or after local training.

Figure 7.3: Side-by-side comparison of JSON-formatted dictionary structures of the built-in *History* object and the proposed *AsynchronousHistory* object.

7.3.4 Usage

The usage of the modified classes is analogous to the usage of the original built-in classes. Just as *Strategy* is passed as a constructor argument upon *Server* object instantiation, *AsynchronousStrategy* is passed as a constructor argument to the *AsynchronousServer* object. Upon calling `fit()` on the *Server* object, a *History* object is returned in SFL. Correspondingly, when `fit()` is called on *AsynchronousServer*, an *AsynchronousHistory* object is returned in AFL. Figure 7.4 contains a side-by-side comparison of these workflows in Python.

```
strategy = FedAvg(...)

sfl_server =  Server(
              ...
              strategy=strategy,
              ...
              )

history : History = \
    sfl_server.fit(...)
```

(a) Flower's built-in *Strategy*, *Server* and *History* usage for implementing an SFL workflow

```
async_strategy = \
    AsynchronousStrategy(...)

afl_server =  AsynchronousServer(
              ...
              async_strategy= \
                  async_strategy,
              ...
              )

async_history \
    : AsynchronousHistory = \
        afl_server.fit(...)
```

(b) Our new *AsynchronousStrategy*, *AsynchronousServer* and *AsynchronousHistory* usage for implementing an AFL workflow

Figure 7.4: Depiction of usage differences between SFL and AFL. On the left is the typical sequence of instances in Flower for an SFL workflow, while the right listing contains the sequence for an AFL workflow.

Library	Version
flower	1.5.0
torch	1.12.1
torchmetrics	1.3.2
numpy	1.26.4
pandas	2.2.1
scikit-learn	1.5.0
hydra-core	1.3.2

Table 7.1: Used Python packages and their versions.

7.4 External Packages & Versioning

For reproducibility, here we present the Python runtime information and versions of the used Python packages. The framework was implemented with **Python 3.10.14** runtime and the main dependencies and their versions are summarized in Table 7.1.

CHAPTER 8

Conclusion

Monumental developments in IoT processing power and sensing technologies have opened up new opportunities for training machine learning (ML) models, particularly in the area of Human Activity Recognition (HAR). HAR, which leverages the vast amount of data collected by smartphones and smartwatches, holds significant potential in fields such as medicine, security, and smart cities. While the prospects of HAR data are promising, it carries stringent privacy constraints that limit the applicability of common ML pipelines such as centralized learning (CL) on very powerful servers. Federated Learning (FL) offers a viable solution since it distributes model training and keeps the data on clients' premises. Further, FL leverages the processing power of data collection devices, which are now capable of supporting both ML model training and inference. However, HAR poses another issue: the presence of data with a high degree of heterogeneity (non-IIDness). This makes the transition process from CL to FL non-trivial, introducing numerous challenges and pitfalls. Moreover, while this transition process is very common in academia and industry, it remains largely unstandardized, introducing significant overheads due to its repeated implementation.

In this thesis, we presented a novel and prescriptively structured methodology for transitioning from CL to both synchronous (SFL) and asynchronous FL (AFL) in non-IID settings. Within this methodology, a CL baseline is established and used as the foundation of subsequent FL models. We provide a thorough discussion of all design decisions introduced by federating the training process and non-IID data.

To illustrate and evaluate the practical feasibility of our presented methodology in real-world settings we applied it on a realistic HAR use case. Through this case study we examined the Extrasensory dataset and the data preprocessing steps undertaken and offered a discussion of the model architecture and tuning decisions.

Subsequently, we empirically evaluated the influence of various FL design decisions on model training and evaluation. Among others, these design decisions included model

hyperparameters, such as batch size and learning rate, optimizer, data augmentation, data scaling, as well as the strategy for generating the test set. These insights were distilled into key takeaways to facilitate quick access and enable the community to leverage these findings more easily.

Lastly, to produce the AFL results presented in this work, we developed an open-source AFL extension of the Flower framework which fosters research on AFL and is publicly available. Together with the framework, an extensive discussion about implementation details was provided to promote reproducibility and ease the navigation of the repository.

8.1 Takeaways

In our proposed methodology, we introduced several data- and model-related design decisions. We then conducted an empirical evaluation of these data and model-related decisions within the suggested non-IID HAR use case. Based on our evaluation results we identified the following main takeaways:

1. Generating a stable and **fair test set**, considering data of all clients, carries privacy risks but is crucial to ensuring reliable performance evaluation.

2. **Data augmentation** is a tunable parameter that can significantly improve performance in non-IID FL. It is tunable because no data augmentation makes the model ignore the minority classes, while excessive augmentation leads to oversampling bias.

3. **Global data scaling** carries privacy risks while offering a more consistent view of the data,

4. Even after global standardization the **feature skew** often persists and this is especially prominent in minority classes.

5. Using a state-based **optimizer** such as ADAM degrades the performance in non-IID FL, giving advantage to more naive optimizers such as SGD-m.

6. Additional **delays on the server** degrade performance even with accordingly extended trainining time.

8.2 Limitations

In this section, we discuss the limitations and potential shortcomings of this work, acknowledging the challenges we faced and how they might affect the broader application of our findings.

8.2.1 Simulation vs Training on Physical Devices

Instead of performing the model training on actual devices (smartphones, smartwatches), these were simulated in our work. Simulating the devices, naturally, limits the applicability of our findings to real-world scenarios, however, throughout this thesis we kept the models relatively small and limited ourselves to a smaller number of packages used by the clients with the goal of reducing the resource footprint.

8.2.2 Periodicity in asynchronous FL updates

After examining the order in which AFL updates were merged into the global model on the server, we observed a significant amount of periodicity, i.e. the updates of different clients were merged in similar and repeating order over time. This raises questions about the asynchronicity properties of AFL and points to a bottleneck where clients have to wait until their updates are merged. Moreover, once the pattern of update merges gets established, it is only facilitated by the fact that clients immediately start training again, which gives an advantage to the clients that merged their update sooner. We suspected that the *ThreadPoolExecutor*, responsible for merging the models, might be the bottleneck here, as only four threads were used for merging the updates coming from 60 clients, however increasing this number did not bring improvement in this regard. Introducing random client-side delays in training partially alleviated the issue. Therefore, we speculate that the issue might lie in the underlying mechanisms implemented by the Flower framework.

8.3 Future Work

We conclude this work by exploring potential avenues for future research that build upon the contributions presented in this thesis.

8.3.1 Federated Learning Adaptations

In Chapter 4, we mention targeted adaptations of the typical FL workflow. These adaptations hold great potential for pushing the boundaries of research in this field and many authors already contributed by suggesting various adaptations. We aim to investigate the causes of performance degradation in non-IID settings further and find critical points that lead to effective FL adaptations. Thus our future work will be continued in several directions. As we already established the performance degradation present in non-IID federated settings, we aim to further investigate the model update gradients and develop an adaptive asynchronous solution based on the new findings. This solution will have the goal of improving the performance of underrepresented classes and, by proxy, the general model performance.

8.3.2 Online Asynchronous Federated Learning

We also intend to extend our current AFL solution and methodology to *online (data streaming) settings* which are paramount for the real-time FL systems. In this scenario, the clients would gradually gain access to new samples over time. This introduces several new design considerations. One key decision is determining the newness and amount of data required for starting the local training on asynchronous clients.

Placing this approach in a broader temporal context leads to the concept of *continuous FL*, where there is no predefined maximum training time. Here, however, further challenges such as *client drift* appear, where a client's data distribution changes over time due to changes in user behavior or environmental factors. This makes the older models obsolete for such clients and demands *adaptive training mechanisms*. In the presence of non-IID data, these challenges are substantially intensified, making the development of continuous online AFL schemes for realistic and heterogeneous datasets particularly complex.

Successfully addressing these challenges would represent a significant advancement in IoT FL and facilitate advanced Federated Learning systems that operate more effectively and offer stable performance in dynamic, real-world environments.

Overview of Generative AI Tools Used

ChatGPT-4o[1] was used sporadically throughout the thesis to correct grammar and improve the clarity of the already existing text. The tool did not generate entire sentences but rather suggested better/correct wording of the parts of the already existing sentences.

GitHub Copilot[2] (version 1.223.0) was used during the implementation primarily for a more advanced autocomplete function and not for design suggestions.

[1] https://chatgpt.com/
[2] https://github.com/features/copilot

List of Figures

List of Tables

Bibliography

[1] M. G. Morshed, T. Sultana, A. Alam, and Y.-K. Lee, "Human action recognition: A taxonomy-based survey, updates, and opportunities," *Sensors*, vol. 23, no. 4, 2023.

[2] P. Kumar, S. Chauhan, and L. K. Awasthi, "Human activity recognition (har) using deep learning: Review, methodologies, progress and future research directions," *Archives of Computational Methods in Engineering*, vol. 31, no. 1, pp. 179–219, 2024.

[3] A. Saha, S. Rajak, J. Saha, and C. Chowdhury, "A survey of machine learning and meta-heuristics approaches for sensor-based human activity recognition systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 15, no. 1, pp. 29–56, 2024.

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

[5] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, "Federated learning for the internet of things: Applications, challenges, and opportunities," *IEEE Internet of Things Magazine*, vol. 5, no. 1, pp. 24–29, 2022.

[6] R. S. Antunes, C. André da Costa, A. Küderle, I. A. Yari, and B. Eskofier, "Federated learning for healthcare: Systematic review and architecture proposal," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1–23, 2022.

[7] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, 2018.

[8] G. Saleem, U. I. Bajwa, and R. H. Raza, "Toward human activity recognition: a survey," *Neural Computing and Applications*, vol. 35, no. 5, pp. 4145–4182, 2023.

[9] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018.

[10] J. Pei, W. Liu, J. Li, L. Wang, and C. Liu, "A review of federated learning methods in heterogeneous scenarios," *IEEE Transactions on Consumer Electronics*, pp. 1–1, 2024.

[11] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[12] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 2023.

[13] Y. Wang, Y. Cao, J. Wu, R. Chen, and J. Chen, "Tackling the data heterogeneity in asynchronous federated learning with cached update calibration," in *The Twelfth International Conference on Learning Representations*, 2024.

[14] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[15] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 15–24, IEEE, 2020.

[16] X. Lu, Y. Liao, P. Lio, and P. Hui, "Privacy-preserving asynchronous federated learning mechanism for edge network computing," *Ieee Access*, vol. 8, pp. 48970–48981, 2020.

[17] C. Jobanputra, J. Bavishi, and N. Doshi, "Human activity recognition: A survey," *Procedia Computer Science*, vol. 155, pp. 698–703, 2019.

[18] N. A. Choudhury and B. Soni, "In-depth analysis of design & development for sensor-based human activity recognition system," *Multimedia Tools and Applications*, pp. 1–40, 2023.

[19] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1192–1209, 2012.

[20] V. Bianchi, M. Bassoli, G. Lombardo, P. Fornacciari, M. Mordonini, and I. De Munari, "Iot wearable sensor and deep learning: An integrated approach for personalized human activity recognition in a smart home environment," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8553–8562, 2019.

[21] A. Chelli and M. Pätzold, "A machine learning approach for fall detection and daily living activity recognition," *IEEE Access*, vol. 7, pp. 38670–38687, 2019.

[22] R. K. Tripathi, A. S. Jalal, and S. C. Agrawal, "Suspicious human activity recognition: a review," *Artificial Intelligence Review*, vol. 50, pp. 283–339, 2018.

[23] Z. Sun, S. Tang, H. Huang, Z. Zhu, H. Guo, Y.-e. Sun, and L. Huang, "Sos: Real-time and accurate physical assault detection using smartphone," *Peer-to-Peer Networking and Applications*, vol. 10, pp. 395–410, 2017.

[24] N. Siddiqui and R. H. Chan, "A wearable hand gesture recognition device based on acoustic measurements at wrist," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 4443–4446, IEEE, 2017.

[25] L. Xie, C. Wang, A. X. Liu, J. Sun, and S. Lu, "Multi-touch in the air: Concurrent micromovement recognition using rf signals," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 231–244, 2017.

[26] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 790–808, 2012.

[27] Y. Vaizman, K. Ellis, and G. Lanckriet, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE pervasive computing*, vol. 16, no. 4, pp. 62–74, 2017.

[28] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[29] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, pp. 41–75, 1997.

[30] O. Dekel, P. M. Long, and Y. Singer, "Online multitask learning," in *International Conference on Computational Learning Theory*, pp. 453–467, Springer, 2006.

[31] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," *Advances in neural information processing systems*, vol. 30, 2017.

[32] A. Ahmed, A. Das, and A. J. Smola, "Scalable hierarchical multitask learning algorithms for conversion optimization in display advertising," in *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 153–162, 2014.

[33] D. Mateos-Núñez, J. Cortés, and J. Cortes, "Distributed optimization for multi-task learning via nuclear-norm approximation," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 64–69, 2015.

[34] S. Liu, S. J. Pan, and Q. Ho, "Distributed multi-task relationship learning," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 937–946, 2017.

[35] P. Baran, "On distributed communications networks," *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.

[36] S. Hu, X. Chen, W. Ni, E. Hossain, and X. Wang, "Distributed machine learning for wireless communication networks: Techniques, architectures, and applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1458–1493, 2021.

[37] S. He, X. Lyu, W. Ni, H. Tian, R. P. Liu, and E. Hossain, "Virtual service placement for edge computing under finite memory and bandwidth," *IEEE Transactions on Communications*, vol. 68, no. 12, pp. 7702–7718, 2020.

[38] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[39] P. A. Bernstein and E. Newcomer, *Principles of transaction processing*. Morgan Kaufmann, 2009.

[40] I. Raicu, I. Foster, A. Szalay, and G. Turcu, "Astroportal: A science gateway for large-scale astronomy data analysis," in *Teragrid conference*, pp. 12–15, 2006.

[41] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Federated learning: Collaborative machine learning without centralized training data." Google AI Blog, Apr 2017. Accessed: 06.02.2024.

[42] F. Cruciani, A. Vafeiadis, C. Nugent, I. Cleland, P. McCullagh, K. Votis, D. Giakoumis, D. Tzovaras, L. Chen, and R. Hamzaoui, "Feature learning for human activity recognition using convolutional neural networks: A case study for inertial measurement unit and audio data," *CCF Transactions on Pervasive Computing and Interaction*, vol. 2, no. 1, pp. 18–32, 2020.

[43] G. M. Weiss, J. L. Timko, C. M. Gallagher, K. Yoneda, and A. J. Schreiber, "Smartwatch-based activity recognition: A machine learning approach," in *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pp. 426–429, IEEE, 2016.

[44] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 2023.

[45] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen, "Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation," *arXiv preprint arXiv:2205.13797*, 2022.

[46] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*, pp. 3581–3607, PMLR, 2022.

[47] J. So, R. E. Ali, B. Güler, and A. S. Avestimehr, "Secure aggregation for buffered asynchronous federated learning," *arXiv preprint arXiv:2110.02177*, 2021.

[48] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz, *et al.*, "A public domain dataset for human activity recognition using smartphones.," in *Esann*, vol. 3, p. 3, 2013.

[49] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[50] Y. Vaizman, N. Weibel, and G. Lanckriet, "Context recognition in-the-wild: Unified model for multi-modal sensors and multi-label classification," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–22, 2018.

[51] M. Fazli, K. Kowsari, E. Gharavi, L. Barnes, and A. Doryab, "Hhar-net: H ierarchical h uman a ctivity r ecognition using neural net works," in *Intelligent Human Computer Interaction: 12th International Conference, IHCI 2020, Daegu, South Korea, November 24–26, 2020, Proceedings, Part I 12*, pp. 48–58, Springer, 2021.

[52] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human activity recognition using federated learning," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 1103–1111, IEEE, 2018.

[53] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen, "Smart devices are different: Assessing and mitigating-mobile sensing heterogeneities for activity recognition," in *Proceedings of the 13th ACM conference on embedded networked sensor systems*, pp. 127–140, 2015.

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

[55] Y. Chen, Z. Chai, Y. Cheng, and H. Rangwala, "Asynchronous federated learning for sensor data with concept drift," in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 4822–4831, IEEE, 2021.

[56] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-iid data: A survey," *Neurocomputing*, vol. 465, pp. 371–390, 2021.

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

[58] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," in *2020 international joint conference on neural networks (IJCNN)*, pp. 1–9, IEEE, 2020.

[59] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[60] N. H. Nguyen, P. L. Nguyen, T. D. Nguyen, T. T. Nguyen, D. L. Nguyen, T. H. Nguyen, H. H. Pham, and T. N. Truong, "Feddrl: Deep reinforcement learning-based adaptive aggregation for non-iid data in federated learning," in *Proceedings of the 51st International Conference on Parallel Processing*, pp. 1–11, 2022.

[61] G. Drainakis, P. Pantazopoulos, K. V. Katsaros, V. Sourlas, A. Amditis, and D. I. Kaklamani, "From centralized to federated learning: Exploring performance and end-to-end resource consumption," *Computer Networks*, vol. 225, p. 109657, 2023.

[62] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proceedings of the second workshop on distributed infrastructures for deep learning*, pp. 1–8, 2018.

[63] Y. Wang, "Co-op: Cooperative machine learning from mobile devices," 2017.

[64] Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang, "Federated learning with non-iid data: A survey," *IEEE Internet of Things Journal*, 2024.

[65] Q. Shen, H. Feng, R. Song, S. Teso, F. Giunchiglia, H. Xu, *et al.*, "Federated multi-task attention for cross-individual human activity recognition," in *IJCAI*, pp. 3423–3429, IJCAI, 2022.

[66] Y. Chen, Y. Ning, Z. Chai, and H. Rangwala, "Federated multi-task learning with hierarchical attention for sensor data analytics," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.

[67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[68] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 6341–6345, IEEE, 2019.

[69] L. Ju, T. Zhang, S. Toor, and A. Hellander, "Accelerating fair federated learning: Adaptive federated adam," *arXiv preprint arXiv:2301.09357*, 2023.

[70] S. P. Karimireddy, M. Jaggi, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Mime: Mimicking centralized stochastic algorithms in federated learning," *arXiv preprint arXiv:2008.03606*, 2020.

[71] J. Ge, G. Xu, J. Lu, C. Xu, Q. Z. Sheng, and X. Zheng, "Fedaga: A federated learning framework for enhanced inter-client relationship learning," *Knowledge-Based Systems*, vol. 286, p. 111399, 2024.

[72] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, *et al.*, "Ray: A distributed framework for emerging {AI} applications," in *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pp. 561–577, 2018.

[73] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[74] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.