Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Department of Mathematics and Physics

Master of Data Science

Analysis and Implementation of Different Open-Source Federated Learning Frameworks to Assess their Technical Implications

---

THESIS to obtain the **DEGREE** of
MASTER OF DATA SCIENCE

A thesis presented by:
Antonio de Jesús Juan Fernández

Thesis Advisors:
Dr. Gema Berenice Gudiño Mendoza

Tlaquepaque, Jalisco, May, 2025

Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Department of Mathematics and Physics

Master of Data Science Approval Form

Thesis Title: **Analysis and Implementation of Different Open-Source Federated Learning Frameworks to Assess their Technical Implications**

Author: **Antonio de Jesús Juan Fernández**

Thesis Approved to complete all degree requirements for the Master of Science Degree in Data Science.

Thesis Advisor, **Dr. Gema Berenice Gudiño Mendoza**

Thesis Reader, **Dr. Edgar Alejandro Guerrero Arroyo**

Thesis Reader, **Dr. Ivan Esteban Villalón Turrubiates**

Academic Advisor, **Dr. Rocío Carrasco Navarro**

Tlaquepaque, Jalisco, May, 2025

Analysis and Implementation of Different Open-Source Federated Learning Frameworks to Assess their Technical Implications

Antonio de Jesús Juan Fernández

Abstract

Google introduced Federated Learning, an approach to decentralized machine learning model training, in 2016. It is designed to allow the use of private data to train machine learning models without the need to possess the data or even "see" it.

The main premise of Federated Learning is a paradigm shift from the traditional centralized machine learning training workflow to a distributed setting. In this setting, users carry out the training locally without ever revealing their data and only share the results of their efforts anonymously as model parameter updates, either to a local server or a network of other users.

Over the years, several Federated Learning frameworks have emerged, each offering different sets of settings and serving either a broad or a particular purpose.

While several comparisons have been made to determine the framework with the most comprehensive set of features, no comparison is available to assess their utility and the implications of using them at an empirical level.

This case study uses the popular frameworks NVFlare, Flower, and Federated Scope to evaluate and showcase their main strengths and potential drawbacks, emphasizing the use of an external dataset and model.

The results showed that regardless of whether the frameworks displayed considerable strengths in certain areas, there is still room for improvement, and that, even if they simplify the implementation of Federated Learning, a factor of manual work still needs to be taken into account, regardless of the framework at hand.

Ultimately, the frameworks have relevant features and areas of opportunity that anyone looking to adopt Federated Learning will need to consider; however, the technical analysis should give a broad perspective on the implications of using the chosen frameworks.

Keywords: Federated Learning, Framework, FedAvg, Distributed Machine Learning.

Analysis and Implementation of Different Open-Source Federated Learning Frameworks to Assess their Technical Implications

Antonio de Jesús Juan Fernández

Resumen

Google introdujo Federated Learning, un acercamiento para el entrenamiento de modelos de machine learning, en 2016. Está diseñado para permitir el uso de datos de propiedad privada para entrenar modelos sin la necesidad de poseer los datos o incluso de verlos.

La premisa principal de Federated Learning es un cambio de paradigma del flujo de trabajo de un entrenamiento de machine learning centralizado a uno distribuido. En este entorno, los usuarios llevan a cabo el entrenamiento de forma local sin revelar su información en ningún momento y solo comparten sus resultados como actualizaciones de párametros del modelo, anónimamente.

A lo largo del tiempo, una gran cantidad de librerías de Federated Learning han emergido, cada una ofreciendo distintos sets de configuraciones, sirviendo un propósito general o uno específico.

A pesar de que se han realizado distintas comparaciones para determinar cuál es la librería con el set de herramientas más amplio, no hay ninguna comparación para evaluar la utilidad individual y las implicaciones de su uso a un nivel empírico.

Este caso de estudio usa las populares librerías NVFlare, Flower y Federated Scope para evaluar y mostrar sus principales fortalezas y potenciales desventajas, enfatizando el uso de un set de datos y un modelo ajeno a las librerías.

Los resultados mostraron que, a pesar de que cada librería presentaba fortalezas en ciertas áreas, aún había espacio de mejora y que, incluso si éstas simplifican la implementación de Federated Learning, existe un grado de factor manual que debe mantenerse en consideración, no importa que librería se utilice.

Al final, las librerías tienen herramientas relevantes y áreas de oportunidad que cualquiera buscando adoptar Federated Learning necesitará tomar en cuenta; no obstante, el análisis técnico debería de proveer una perspectiva amplia de las implicaciones de usar las librerías presentadas.

Keywords: Federated Learning, Framework, FedAvg, Machine Learning Distribuido.

Contents

List of Figures

[1] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019b. URL http://arxiv.org/abs/1905.11946

List of Tables

Listings

List of Algorithms

List of Acronyms

FL	Federated Learning
DP	Differential Privacy
AI	Artificial Intelligence
ML	Machine Learning
I.I.D	Independent Identically Distributed
non-I.I.D	non-Independent Identically Distributed
SGD	Stochastic Gradient Descent
NbAFL	Nosing Before Aggregation Federated Learning
PP	Percentile Privacy
SVT	Sparse Vector Technique
GLv2	Google Landmark Version 2
LDA	Latent Dirichlet Allocation
HE	Homomorphic Encryption

Dedicated to:

This thesis is dedicated to those who have supported me throughout my education.

Thank you for allowing me to take another step in this adventure.

Dedicado a:

Esta tesis está dedicada a aquellos que me han ayudado a lo largo de mi educación.

Gracias por permitirme dar un paso más en esta aventura.

1 Introduction

Contents

1.1 The need for a paradigm shift

1.1.1 Stricter Data Restrictions

As the world has become more data-aware and more restrictive privacy-preserving laws and regulations have been introduced. These measures help ensure sensible information is safely handled by third parties; thus, obtaining valuable data to train machine learning models effectively has become increasingly difficult.

In 2016, the General Data Protection Regulation (*GDPR*) was introduced in the European Union to establish a series of rules for data governance regarding personal data processing[1]; this was a massive stepping stone for privacy preservation. The GDPR establishes a framework of enforcement mechanisms, penalty determination, expanded security incident notification, and procedural requirements that document deviations from the law[2], with costly fines for non-compliant corporations or individuals.

Other relevant regulations passed in the last few years include:

[1] European Parliament. Regulation (EU) 2016/679 of the European Parliament and of the Council, 2016. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[2] Bart van der Sloot Chris Jay Hoofnagle and Frederik Zuiderveen Borgesius. The european union general data protection regulation: what it is and what it means*. *Information & Communications Technology Law*, 28(1):65–98, 2019. DOI: 10.1080/13600834.2019.1573501. URL https://doi.org/10.1080/13600834.2019.1573501

- The Artificial Intelligence Act of 2024 (AIA)[3] addresses several components related to AI system concerns, such as data governance and forbidden practices.
- The Financial Transparency Act of 2021, which, among other things, requires federal financial regulatory agencies to adopt specified data standards with respect to format, searchability, and transparency[4].
- The California Consumer Privacy Act of 2020 (CCPA)[5], which "grants California residents certain rights over their personal information, including the right to know what data is collected, the right to opt-out of data sales, and the right to request deletion of their information"[6].

[3]

[4] H.R.2989 - 117th Congress. Financial transparency act of 2021, 2021. URL https://www.congress.gov/bill/117th-congress/house-bill/2989

[5] California State Legislature. California consumer privacy act (ccpa), 2018. URL https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375

[6] Seun Solomon Bakare, Adekunle Oyeyemi Adeniyi, Chidiogo Uzoamaka Akpuokwe, and Nkechi Emmanuella Eneh. Data privacy laws and compliance: A comparative review of the eu gdpr and USA regulations. *Comput. sci. IT res. j.*, 5(3):528–543, 3 2024

Institutions have rightfully so become more cautious about utilizing and lending data for AI training purposes; therefore, a complete paradigm shift is required to comply with strict privacy regulations in cases where acquiring data to train ML algorithms is not possible or very difficult, either by law or by proprietary institutional policies.

1.1.2 *Decentralized machine learning*

Federated Learning (FL) is an approach introduced by Google in 2016[7] that aims to allow ML algorithms to be trained in a decentralized setting. The main premise of FL is to enable data scientists and ML engineers to perform model training without actually seeing the data that is being used, which is strictly distributed across a network of clients.

[7] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023. URL https://arxiv.org/abs/1602.05629

The paradigm that FL proposes enables companies to keep their sensible information safe while still being able to contribute to a machine learning project.

Since it was first introduced, FL has proven to be a valuable tool for tackling problems related to next-word prediction[8] and traffic detection to improve route selection in GPS systems[9], among others.

At the start, the main goal of FL was to train neural networks; however, as time has passed, new implementations have come out to dive into different machine learning algorithms, like logistic regression and clustering, or have specialized in a subset of the neural network spectrum, such as convolutional neural networks [10].

[8] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019. URL https://arxiv.org/abs/1811.03604

[9] Chenming Xu and Yunlong Mao. An improved traffic congestion monitoring system based on federated learning. *Information*, 11(7), 2020. ISSN 2078-2489. DOI: 10.3390/info11070365. URL https://www.mdpi.com/2078-2489/11/7/365

1.2 *Justification*

1.2.1 *A problem of many alternatives*

An FL framework comprises a series of modules integrating diverse, complex tasks to enable the user to train an ML model using the FL paradigm. Currently, most frameworks are available as open-source

[10] Chaoyang He, Alay Dilipbhai Shah, Zhenheng Tang, Di Fan1Adarshan Naiynar Sivashunmugam, Keerti Bhogaraju, Mita Shimpi, Li Shen, Xiaowen Chu, Mahdi Soltanolkotabi, and Salman Avestimehr. Fedcv: A federated learning framework for diverse computer vision tasks, 2021. URL https://arxiv.org/abs/2111.11066

Python libraries that use different technologies at a low level, such as Java or C++ components, to handle communication or computation processes.

A wide variety of FL frameworks have come out ever since, covering different areas using a wide variety of algorithms, and while there have been articles and discussions on which frameworks have the most comprehensive set of features, very little research has been done on using case studies to test their implementations comparatively using an empirical approach.

Not having a hands-on framework comparison poses a challenge for machine learning practitioners who seek to adopt this type of model training methodology and have no way of knowing the difference in implementation regarding the technical side of using them or how the set of tuning parameters differs among FL frameworks.

Sharing details about each framework's implementation will help understand their technical implications.

1.3 Problem Statement

Since FL was first introduced, several frameworks have been introduced; however, no comprehensive comparison has empirically addressed their implications, which makes it challenging for ML practitioners to choose a framework correctly.

Not having a clear perspective on the technical ramifications of using the frameworks might deter people from using them or even demotivate anyone interested in adopting FL altogether.

1.4 Research Question

Is it possible to assess and compare the individual strengths of a framework via an empirical implementation that uses external components, such as a model and data?

1.5 Hypothesis

It is possible to have a hands-on quantitative and qualitative framework comparison so long as the implementation thereof contains similar components and the implementation's scope remains at a standard level.

1.6 Main Objective

Implement different federated learning frameworks to assess and compare empirically their general purpose and individual strengths by

means of a case study.

1.7 Secondary Objectives

1.7.1 Qualitative analysis

- Evaluate qualitatively the ease of implementation across selected frameworks to contrast their feature availability against their technical implications. This degree of easiness will be analyzed from the point of view of an individual with machine learning (specifically deep learning) and federated learning background; in this context, the main goal is to review how much does the person need to get involved in the framework to carry out simulations.
- Assess the range of tuning options available for the evaluated algorithms to showcase the versatility of the individual frameworks' implementations, as the algorithms being the same doesn't directly imply the same degree of parameter tuning, there might (and most definitely will) be a difference in this regard.
- Review each implemented framework's documentation to analyze the degree of completeness; this will give insight into what anyone looking to implement FL will face, involving component guidance and functionality.

1.7.2 Quantitative analysis

- Compare computational overhead through runtime analysis to analyze the differences in processing time at a high level. The runtime analysis will provide a better perspective on how moving across different frameworks escalates the time it takes for the models to be trained.

1.8 Chapter Summary

This chapter describes the context that has made machine learning increasingly difficult due to privacy regulations and concerns. It motivated the case study by briefly explaining the problem of not having a comprehensive hands-on framework comparison that underlies technical implications regarding FL implementations. It also mentioned the study's primary and secondary goals to give a perspective of where the main emphasis of the work will be placed.

2 Literature Review

Contents

2.1 Chapter Overview

Analyzing and comparing FL frameworks is a concept that has already been explored. Many authors have dived into the ins and outs of the individual frameworks at a feature level to share their value propositions and contrast them against one another. However, there is a void in the current comparison landscape when it comes to FL framework comparisons.

This chapter discusses and gives a broad view of the current work done on FL framework analysis, including previous comparisons, initial framework value proposals, and even previous implementation approaches to framework usage implications.

Finally, the discussion will shift to the value this case study provides in providing a fresh perspective on what an actual implementation looks like, utilizing a hands-on approach to contrast the previous theoretical work.

2.2 Previous work

Federated Learning is a relatively novel concept that, in its short life, has shown great potential to improve upon current systems in regards to privacy. It provides a paradigm shift for the decentralized machine learning sector, and as one might imagine, several frameworks have emerged since it was first announced. Over the years, some comparisons have been published to clarify what the frameworks offer feature- and component-wise.

Karimireddy et al. [1] provide a comprehensive component comparison between several frameworks, such as NVFlare, PySyft, PaddleFL, Substra, Flower, FederatedScope, and FedML, among others, emphasizing fundamental features, such as data distribution support (for I.i.d. and non-I.i.d. datasets), client dropout resilience, and algorithm availability, among others, from a high-level standpoint, while also recommending frameworks based on the number of features available.

Kaur et al. [2] also describe components related to some FL frameworks, namely Tensorflow Federated, FATE, PySyft, PaddleFL, and FL&DP, albeit in more detail about how each framework's inner components' architecture works together. Additionally, there are several examples of recent advances and applications of FL across different fields, like medical and computer vision. Finally, they mention key challenges for FL regarding privacy, model performance, or production-level considerations like monitoring or deployment, which, as they point out, might not be mature enough.

Li et al. [3] describe the main participants (parties, manager, and the framework) in an FL system and give an overview of their role in the overall workflow. They also discuss other key concepts related to privacy, model aggregation, and communication. Regarding specific frameworks, they mention popular libraries like TensorFlow Federated, PySyft, and PaddleFL, among others, and compare them using a high-level component analogy.

[1] Sai Praneeth Karimireddy, Narasimha Raghavan Veeraragavan, Severin Elvatun, and Jan F. Nygård. Federated learning showdown: The comparative analysis of federated learning frameworks. In *2023 8th International Conference on Fog and Mobile Edge Computing, FMEC 2023*, pages 224–231. Institute of Electrical and Electronics Engineers Inc., 2023. ISBN 9798350316971. DOI: 10.1109/FMEC59375.2023.10305961

[2] Harmandeep Kaur, Veenu Rani, Munish Kumar, Monika Sachdeva, Ajay Mittal, and Krishan Kumar. Federated learning: a comprehensive review of recent advances and applications. *Multimedia Tools and Applications*, 83:54165–54188, 5 2024. ISSN 15737721. DOI: 10.1007/s11042-023-17737-0

[3] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4): 3347–3366, April 2023. ISSN 2326-3865. DOI: 10.1109/tkde.2021.3124599. URL http://dx.doi.org/10.1109/TKDE.2021.3124599

Release and Support summary					
Framework	Introduced	Year	O.Source	Commits	Last commit
TFlowFL	article	2019	yes	5,565	this week
OpenFL	paper	2021	yes	6,928	this week
Flower	paper	2020	yes	3,512	this week
PaddleFL	paper	2019	yes	735	1+ year
PySyft	paper	2019	yes	35,228	this week
FATE	paper	2019	yes	13,850	4+ months
FedML	paper	2020	yes	12,120	10+ months
FedScale	paper	2022	yes	707	1+ year
NVFLare	paper	2022	yes	2,115	this week
FedScope	paper	2022	yes	544	1+ year

Table 2.1: Popular frameworks' release and support information. The table is updated as of March 31, 2025.

Argemi [4] surveyed some popular frameworks to implement a federation using a multi-cloud architecture, mentioning and describing FL-related concepts along the way. Argemi's approach was to survey

[4] Albert Pita Argemi. Design, implementation, and analysis of a cloud federated learning architecture. Technical report, Universitat Politècnica de Catalunya, 2023

and select a framework meeting the requirements needed to build a federation using the cloud; his approach was not comparative as much as it was production-focused. However, he did indeed implement an FL system using NVFlare in his study, which he alluded to as being the most complete framework out of the ones surveyed in his work.

Finally, as frameworks have been introduced over time, to mention a few Nvidia Flare[5], Federated Scope[6], Flower[7], OpenFL[8], and FedML[9], an introductory paper is typically published discussing what each brings to the table compared to what was available.

The discussion often focuses on features implemented, including datasets incorporated for experimentation, performance comparisons, and low-level framework mechanisms. Table 2.1 shows a summary of the frameworks' release information and support since their introduction.

2.3 *What is missing in the comparison scene?*

All of the previous works share the characteristic of discussing different features available and making comparisons purely based on what the frameworks offer; however, aside from Argemi, no other study gets its hands dirty implementing the frameworks, let alone comparing how using them pans out or what their implications are.

This study will, by the end, provide clarity as to the primary considerations when selecting one framework over others, empirically testing them to showcase their utility and adaptability to external factors, such as proprietary data and/or models.

2.4 *Chapter Summary*

This chapter gave a broad perspective on the existing Federated Learning literature scene regarding framework comparisons and descriptions by discussing some related works and their approaches thereto. Additionally, this section underlay how this work fills the gap by empirically comparing the implications of using any of these frameworks to perform Federated Learning simulations.

[5] Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona Flores, and Andrew Feng. Nvidia flare: Federated learning from simulation to real-world. 10 2022. DOI: 10.48550/arXiv.2210.13291. URL http://arxiv.org/abs/2210.13291http://dx.doi.org/10.48550/arXiv.2210.13291

[6] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope: A flexible federated learning platform for heterogeneity. 4 2022. URL http://arxiv.org/abs/2204.05011

[7] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework. 7 2020. URL http://arxiv.org/abs/2007.14390

[8] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards, Micah J. Sheller, Sarthak Pati, Prakash Narayana Moorthy, Shih han Wang, Prashant Shah, and Spyridon Bakas. Openfl: An open-source framework for federated learning. 5 2021. DOI: 10.1088/1361-6560/ac97d9. URL http://arxiv.org/abs/2105.06413http://dx.doi.org/10.1088/1361-6560/ac97d9

[9] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. 7 2020. URL http://arxiv.org/abs/2007.13518

3 Theoretical Framework

Contents

3.1 Section Overview

Federated Learning is a technology that aims to solve the ever-increasingly tight data regulations by allowing machine learning practitioners and companies to carry out their operations without accessing third-party data.

The idea of Federated Learning sounds fantastic, but many moving parts are behind it. To understand what Federated Learning is all about, one needs to comprehend at least at a high level what those parts are and the role they play in the entire process.

This chapter will review some of the most critical components and address the following questions:

- What is Federated Learning?
- How does a typical Federated Learning workflow look?
- How are the models trained?
- How do both the clients and the server handle the parameters?
- At what point do the models get updated?
- How is the privacy of the participants preserved throughout the entire process?

These questions will be answered by the end of the chapter, and enough theory will be provided through a series of definitions and algorithm descriptions to follow the rest of the case study.

3.2 Federated Learning

In its simplest form, FL is conducted by a node in the network that acts as a server and orchestrates the training and aggregation of the model parameters, and a network of clients, the federation.

The training is carried out iteratively: the global model on the server is sent to the clients of the federation, usually a small subset of them, to train the model locally and return its update to the server.

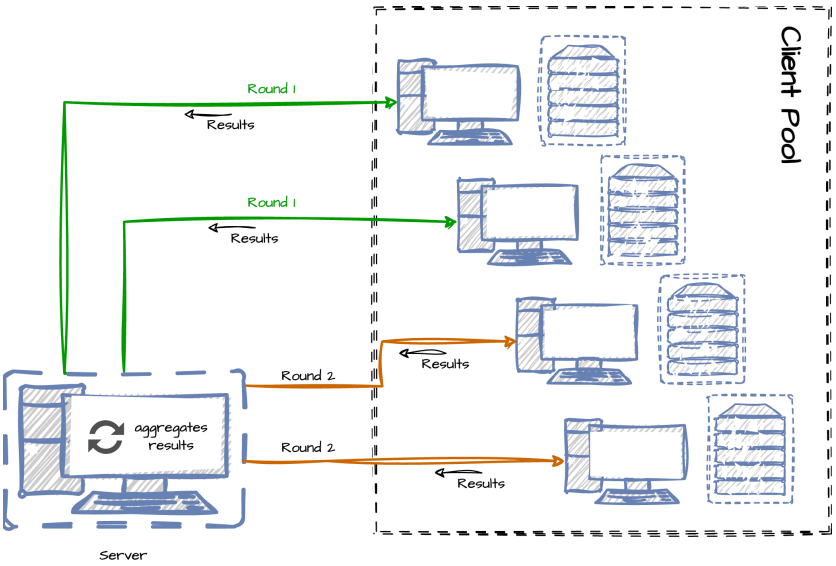

Figure 3.1: Fundamental idea of Federated Learning, a central server subsamples clients and aggregates the results of their training jobs.

At this point, via an aggregation algorithm, the server merges all updates to obtain a new global model, and a subsequent round is initiated to train the model using a different subset of clients[1,2]. Figure 3.1 illustrates at a high-level this workflow.

3.3 Federated Learning Algorithms

This section will describe some of the algorithms that constitute key areas of FL: aggregation, privacy, and communication. Additionally, although out of the scope of this case study, it is worth mentioning that the computation area is also present.

1. During the aggregation step, as its name suggests, the results received for a given communication round are aggregated to update the global model.
2. The privacy-preserving step ensures that the other participants, malicious actors, or the server can infer no relevant information about any participant in particular.

[1] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. A performance evaluation of federated learning algorithms. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, New York, NY, USA, December 2018. ACM

[2] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020. URL https://arxiv.org/abs/1812.06127

3. The communication step ensures that preserving the most information possible, each participant sends the minimal amount of data possible to the server.
4. The computation step instructs the clients in the federation to use only as many computational resources as necessary to complete their tasks and engineers ways to reduce the overall computing required for the federation to complete its goals.

Visually, the FL areas are visible in Figure 3.2. Note how the privacy-preserving aspect is present at all times and everywhere, as is arguably the most important factor.

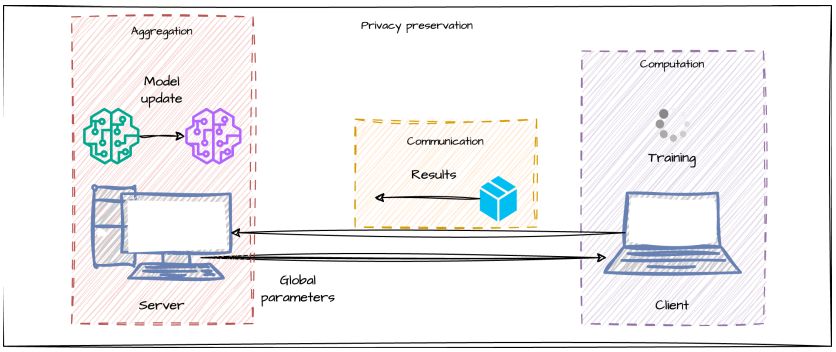

Figure 3.2: Different FL areas in a typical server-client information exchange, note privacy-preserving is present in the entire flow.

3.3.1 *Aggregation*

Federated Averaging

When FL was first introduced by McMahan et al.[3], Federated Averaging (FedAvg) was the fundamental algorithm that tied the idea of FL together. Given its simplicity and widespread adoption, it is the most important aggregation algorithm. While recent aggregation algorithms attempt to improve upon FedAvg's premise, they all branch out from it in one way or another, whether in terms of optimizing computations in networks with client heterogeneity or accelerating performance convergence.

The main idea of FedAvg is simple but powerful: Given a network of K clients, each with its own set of data $\mathcal{B}$, train a decentralized model by using weighted averages of the clients' weights as a function of each client's sample size $|n_k|$, after a series of t training communication rounds.

Taking a weighted average with FedAvg makes sense, as clients could potentially have data sets with widely different sample sizes, and modulating the "influence" that clients with very few samples have on the global model against those having large datasets is important for the overall convergence.

[3] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023. URL https://arxiv.org/abs/1602.05629

A server orchestrates the training by keeping a global model, whose parameters are sent to each client for local training at the beginning of each communication round; subsequently, the clients train the model on their local data and send the resulting weights back to the server for averaging.

Given that the client network S (or federation, as it is also called) may contain an immense amount of clients, a random subsampling process is carried out to select only a fraction C of the total number of clients K, the subsample S_t at a given round t will train the weights locally.

More formally, Algorithm 1 shows the workflow proposed by McMahan et al.[4].

[4] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023. URL https://arxiv.org/abs/1602.05629

Algorithm 1 FederatedAveraging

1: initialize w_0
2: **for** $t = 0, 1, \ldots, n-1$ **do**
3: $\quad m = \max(C \cdot K, 1)$
4: $\quad S_t = s_1, \ldots, s_m \in_R S$
5: $\quad$ **for** each client $k \in S_t$ **in parallel do**
6: $\quad\quad w_{t+1}^k = \text{ClientUpdate}(k, w_t)$
7: $\quad$ **end for**
8: $\quad m_t = \sum_{k \in S_t} n_k$
9: $\quad w_{t+1} = \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
10: **end for**

11: ClientUpdate(k, w_0):
12: $\mathcal{B}$ = split $\mathcal{P}_k$ into batches of size B
13: **for** each local epoch $e = 1, \ldots, E$ **do**
14: $\quad w = w_0$
15: $\quad$ **for** batch $b \in \mathcal{B}$ **do**
16: $\quad\quad w = w - \eta \nabla \ell(w; b)$
17: $\quad$ **end for**
18: **end for**
19: return w

The initial global parameters for the model, w_0, are either initialized randomly or are passed to the server on initialization; n_k is the number of training samples of client k; $\mathcal{P}_k$ is the local dataset for a given client; $\ell(w; b)$ represents the loss of the prediction.

An important consideration regarding FedAvg, and generally speaking, other aggregation algorithms, is that although one could instruct clients to perform several epochs during the local training step (Line 13 of Algorithm 1), assuming that they will be connected to a network to perform their training uninterruptedly is not recommended.

The mere nature of FL suggests that clients will generally default on their jobs if given computationally expensive tasks. It is good practice to take a *conservative* approach and use a small number of epochs for local training, usually one, especially in the cross-device (see Appendix 7.2) scenario.

Finally, two variations for FedAvg achieve the same result but differ in what the clients send back to the server:

1. The server instructs the clients to send their local weights after the local training step so that the server can average the raw parameters as they are.
2. The clients return the element-wise difference between the final local parameters and the original parameters sent from the server at the beginning of the local training step.

The second approach has sparked different opportunities for research regarding convergence improvement, particularly *Federated Optimization*, which will be discussed in more detail in the next section.

Consider the scenario of a subset of clients S_t participating in a given communication round t (for simplicity's sake suppose that the training data sample size is uniform across the clients $|\mathcal{P}_1| = |\mathcal{P}_2| = \cdots = |\mathcal{P}_k|$), each will contribute equally to the global model update at time t; the update function to aggregate the weights for round $t+1$ with FedAvg is shown in Equation 3.1, using the first variation, and in Equation 3.2 for the second one.

$$w_{t+1} = \frac{1}{|\mathcal{S}_t|} \sum_{i=1}^{|\mathcal{S}_t|} w_t^i \tag{3.1}$$

$$w_{t+1} = w_t - \frac{1}{|\mathcal{S}_t|} \sum_{i=1}^{|\mathcal{S}_t|} (w_t - w_t^i) \tag{3.2}$$

Both of the previously shown functions achieve the same result.

Federated Optimization

Federated Optimization[5] (FedOpt) was introduced to build upon FedAvg. The concept of FedOpt also performs an optimization step on the server side, not only locally for each client individually; this is done through a parameterized global model update step on the server, using an update function similar to what the users would perform on their end.

However, how would the server perform an optimization step on the global model if it cannot access the data or perform the training itself? The answer is simple: it does not require access to the data; it only requires the individual client parameter updates every communication round.

[5] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. 2 2020. URL http://arxiv.org/abs/2003.00295

Each client sends its updates to the server as the difference between the original weights received and the final weights after its training step $\Delta_t = \Delta_t^1, \ldots, \Delta_t^k$; when averaged across all clients in a communication round, this set of differences acts as a *gradient*. The server can then update its model, taking a step using the gradient and a learning rate η_s.

Looking back at line 9 of Algorithm 1, FedAvg updates the global model using the results from a given round t and averages them to obtain the parameters w_{t+1}. Since the model parameters w_t do not influence this update, it essentially has a learning rate of $\eta = 1$. FedOpt allows for an update of the global model considering the model at time t and using an optimization algorithm, such as SGD, Adam, Adagrad, or others, and applying a η different to 1.

Algorithm 2 shows FedOpt; For the most part, it remains the same as FedAvg, the only change comes in line 9, where the server updates the parameters using it's own learning rate and optimization algorithm ServerOpt.

Algorithm 2 FederatedOptimization

1: initialize w_0
2: **for** $t = 0, 1, ..., n-1$ **do**
3: $m = \max(C \cdot K, 1)$
4: $S_t = s_1, \ldots, s_m \in_R S$
5: **for** each client $k \in S_t$ **in parallel do**
6: $\Delta w_t^k = \text{ClientUpdate}(k, w_t, \eta_c)$
7: **end for**
8: $\Delta_t = \frac{1}{|S_t|} \sum_{k \in S_t} \Delta w_t^k$
9: $w_{t+1} = \text{ServerOpt}(w_t, \Delta_t, \eta_s)$
10: **end for**

For FedOpt, ServerOpt could take different shapes, depending on the approach taken. In its simplest form ServerOpt could be fully parameterized by just the learning rate η_s, and so line 9 of Algorithm 2 would become:

$$w_{t+1} = w_t - \eta_s \Delta_t \tag{3.3}$$

Other popular variations to the FedOpt algorithm that were also introduced by Reddi et al.[6] make use of different parameters to allow for a tuneable degree of adaptability for the duration of training, or even momentum, such as FedAdagrad, FedYogi, and FedAdam; however, the idea remains the same.

[6] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. 2 2020. URL http://arxiv.org/abs/2003.00295

3.3.2 Communication

Compression

Compression is an active area of research in informatics, given its utility in reducing communication, computation overhead, memory usage, and others; it is no wonder that many techniques used there are also commonly seen in other fields, such as FL.

For FL, some algorithms have been proposed to deal with the amount of data sent across the network between the clients and the server. Given the potentially large number of parameters to be exchanged and the fact that not all devices in the network might have the same communication capacities, reducing the size of the parameters being shared has become increasingly relevant.

Conceptually, the easiest way to perform compression is to use linear 8-bit quantization, which involves mapping a floating point value into an integer-based $[-127, 127]$ range. It is called linear because a difference between two values implies a fixed proportional quantized value increment; for example, the difference between three quantized values is proportional to the difference between them unquantized. An example of this is shown below.

floating	1.5	3.0	6.0
quantized	32 (31.75)	64 (63.5)	127

There are two main ways to achieve 8-bit quantization: high-precision asymmetric quantization and symmetric quantization[7] [8]. The latter method is more popular; some frameworks use this algorithm, which will be analyzed further.

Consider a vector $\mathbf{h} = [h_1, \ldots, h_n]^\top$, to linearly quantize $\mathbf{h}$ using symmetric quantization (also called absolute maximum quantization), one could use the absmax function

$$\text{absmax}(\mathbf{h}) = \mathbf{h}_q = \left[\frac{127 \cdot \mathbf{h}}{q}\right], \; q = \max_{1 \leq i \leq N} \mathbf{h}_i \tag{3.4}$$

here q is also called the quantization factor.

Dequantizing the values is also straightforward; only the quantization factor of the original vector would be required.

$$\text{dequantize}(\mathbf{h}_q, q) = \left[\frac{\mathbf{h}_q \cdot q}{127}\right] \approx \mathbf{h} \tag{3.5}$$

Naturally, there is some loss of precision, given that rounding is involved during the quantization process; nonetheless, this loss is sometimes worthwhile in the communication overhead-precision trade-off.

[7] Don Moon. [vLLM - Quantization] bitsandbytes: 8-bit Optimizers, LLM.int8(), qlora, and k-bit inference scaling laws. https://medium.com/byte-sized-ai/vllm-quantization-bitsandbytes-efaec31f00df, 2024. [Accessed 25-02-2025]

[8] Sachinsoni. Introduction to Model Quantization. https://medium.com/@sachinsoni600517/introduction-to-model-quantization-4effc7a17000, 2023. [Accessed 25-02-2025]

3.3.3 Differential Privacy

Differential Privacy Overview

In the context of FL, companies would be reluctant to lend their data due to privacy-related concerns; this might arguably be the most formidable obstacle to overcome for an FL project to be passed. Even in FL, where the data remains local and is not shared at any point, there is potential for inference attacks, where participants might be able to reconstruct certain aspects of the data from other clients or infer characteristics from the rest of the population [9], among other types of attacks.

Thus, Differential Privacy has recently attracted considerable interest because it allows ML practitioners to protect the parameters sent across the network and from other participants.

As its name suggests, Differential Privacy is a concept related to privacy preservation that was introduced and developed greatly by Cynthia Dwork [10]. The idea of Differential Privacy is simple: one should be able to draw the same conclusions from a population whether a sample is in it or not, with the difference being minimal.

In Cynthia Dwork's words, "Differential privacy addresses the paradox of learning nothing about an individual while learning useful information about a population."[11] The previous statement is powerful; it allows participants, whether individuals or institutions, in any given study or project to be assured that their data will not only be protected from third parties but also that any influence that they might have on the results will not be able to be traced back to them in any way.

Pure Differential Privacy Definition

A succinct definition for Differential Privacy is the following: given two adjacent sets[12] X and X', a function F satisfies ϵ-Differential Privacy, if for all sets $S \subseteq \mathcal{R}$, the ratio of the probabilities of the function's output belonging to S for both sets X and X', are smaller than a certain threshold modulated by ϵ[13].

$$\frac{Pr[F(X) \in S]}{Pr[F(X') \in S]} \leq e^{\epsilon} \tag{3.6}$$

Equation 3.6, shows the *pure* form of Differential Privacy, where only ϵ acts as the parameter to modulate the *privacy budget*. The most desirable setting for Differential Privacy is that ϵ be as small as possible; however, there is a trade-off: as $\epsilon \to 0$, more privacy is preserved, but the model performance gets increasingly worse, and the inverse is also true.

Tweaking around the privacy budget using ϵ is key to balancing between privacy preservation and model performance in FL. There is no rule of thumb for doing this, experimentation should always be performed to tune ϵ to an acceptable value.

[9] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020

[10] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006

[11] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4): 211–407, 2013. ISSN 1551-3068. DOI: 10.1561/0400000042. URL http://dx.doi.org/10.1561/0400000042

[12] Two datasets are considered adjacent datasets if $d(x, x_i) = 1$, their distance is exactly one, that is, they differ in, at most, a single observation.

[13] Joseph P Near and Chike Abuah. Programming differential privacy. *URL: https://uvm*, 2021

Approximate Differential Privacy Definition

A variation called *approximate* Differential Privacy uses an additional parameter δ, which also should be kept small, sometimes recommended to be $\delta = \frac{1}{n}$, where n is the total sample size, or even smaller than that.[14]

[14] ϵ, δ-Differential Privacy

$$Pr[F(X) \in S] \leq e^{\epsilon} Pr[F(X') \in S] + \delta \tag{3.7}$$

The main implication of Differential Privacy (DP from here onwards) is that the output of the differentially-private function, F, ought to be very close whether a sample is removed or plugged into a dataset, and that it should be virtually indistinguishable to know which sample it was. The key is designing a function F, such that it guarantees either ϵ-DP or (ϵ, δ)-DP.

There are several ways to achieve DP in FL; the rest of this section will discuss some of the most popular algorithms, which could broadly be divided into two categories: noising and precise mechanisms.

Noising

Noising is any mechanism (algorithms related to parameter modification or selection are usually called mechanisms) that involves taking the weights X from either the client or the server, after passing them through a transformation function $f(X)$, and adding noise, generally in the form of random samples from a Gaussian or Laplacian distribution. In FL, $f(X)$ would represent the aggregation step on the server side, or the local training workflow on the client side.

For the Gaussian mechanism:

$$F(X) = f(X) + \mathcal{N}(\mu, \sigma^2), \quad \sigma^2 = \frac{2s^2 \log(1.25/\delta)}{\epsilon^2} \tag{3.8}$$

For the Laplacian mechanism:

$$F(X) = f(X) + \mathcal{L}(s/\epsilon) \tag{3.9}$$

Sensitivity

In Equations 3.8 and 3.9, s represents the global sensitivity, the maximum magnitude of change in the result of f, between any two adjacent sets X, X'[15]; in other words, how much can the output of a function f change if an element is added or removed from the dataset. Consider the local sensitivity, between two adjacent datasets:

[15] Two datasets are said to be adjacent if they differ in at most a single data entry.

$$s^{X,X'} = \max \left|\left| \frac{1}{|X|} \sum_{j=1}^{|X|} \arg\min F(\mathrm{w}, X_j) \right.\right. \tag{3.10}$$

$$\left.\left. - \frac{1}{|X'|} \sum_{j=1}^{|X'|} \arg\min F(\mathrm{w}, X_j^{'}) \right|\right| \tag{3.11}$$

The global sensitivity, then could be defined as the maximum pairwise local sensitivity:

$$s = \max\{s^{X,X'} : d(X, X') = 1\} \tag{3.12}$$

As its name suggests, the definition for global sensitivity holds regardless of which two adjacent datasets are queried[16].

To illustrate the key premise of noising, consider the Gaussian mechanism and let $\boldsymbol{X}_k = \{X_j\}_{j=1}^{k}$, $X_j \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, as $k \to \infty$, $E[\bar{\boldsymbol{X}}_{\boldsymbol{k}}] \to \mu_k = 0$ by the Law of large numbers.

Supposing that every client in the federation is individually training $f(X_1), ..., f(X_k)$ as an unbiased estimate using their local data, the larger k is, the closer it is expected for $E[\bar{X}_k]$ to converge to μ_k; aggregating the results across a large enough series of clients would result in $F(X_k) = f(X_k)$. The resulting aggregation would update the model taking into account the noised parameters from every client, without needing the *real* parameters.

Another consideration regarding noising is that it can be performed locally by each client, globally by the server, or on both sides.

The first approach is the first one that comes to mind when thinking about noising. It is used when the clients do not fully trust the server and would prefer to noise their parameters before sending them back for aggregation.

The second would be appropriate when the server is concerned with other clients somewhat inferring information from other clients upon receiving the global model's parameters for local training. It seems counter-intuitive that the server concerned with model convergence would send out noised parameters of the global model; nonetheless, when considering that the parameters, though wrong at first, will be received by each client and locally trained, thus brought back to normality after each communication round. There is naturally a performance-privacy trade-off to be considered; however, in a big network with uncertainty, it is worth using global noising.

The third and final approach would involve both sides using noising, which would inherently hinder the performance and increase the computation efforts required to achieve convergence, but it would be the most secure.

Precise mechanisms

[16] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575

A different approach to DP is achieved using precise mechanisms. They are called precise because instead of returning non-authentic values to the server (noised values with a random value added to them are not authentic), they return existing values from the original parameter set. However, a selection algorithm chooses which parameters are returned to the server.

Diverse ways exist to achieve DP using precise methods; this study is concerned with Percentile Privacy (PP) and the Sparse Vector Technique (SVT), which perform model parameter optimization by selecting the most critical subset of the original weights and updating the model based on those weights only.

Defining what critical weights mean and how the degree of criticality will be measured is essential. Weights presenting relatively large gradients could be considered significant, but how large do the gradients have to be to become important? Usually, a scoring function based on a threshold is used to determine if the weight is substantial.

The threshold could be based on an arbitrary gradient value or a given gradient quantile q. Only gradients with an absolute value whose rank lies above q will be sent back to the server, with the remaining being fixed to a specific value, often 0. SVT and PP, which will be covered shortly, use the previous two threshold methods, respectively.

Noising before Model Aggregation FL

Also known as NbAFL[17], this algorithm uses noising, both on the server side after the aggregation step but before broadcasting the parameters to the clients $w_t + n_D$ and on the client side before returning the individual update back to the server $w_t^k + n_t^k$, where n represents the noise, which could either be gaussian or laplacian; NbAFL is displayed in Algorithm 3.

The server-side noising ensures no clients can infer information from other clients from the server-broadcasted parameters. This noise is then canceled during the local training process. Note that Line 7 of Algorithm 3 makes use of a regularization technique called Clipping, which prevents large parameter updates from being sent to the server by upper-bounding the parameters' ℓ_1 norm to a Clipping threshold C, such that $||w_t^k|| \leq C$.[18]

Differential Privacy with Adaptive Clipping

Differential Privacy with Adaptive Clipping[19] is also a noising technique, with the novelty that instead of using a constant value for the Clipping norm C, it uses a learning rate, η_C, to update it progressively in order for it to clip a fixed quantile $0 \leq \gamma \leq 1$, along the weights distribution.

Adaptive Clipping is achieved by iteratively not only receiving

[17] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575

[18] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575

[19] Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping, 2022. URL https://arxiv.org/abs/1905.03871

Algorithm 3 NoisingBeforeAggregationFL

1: define: T, μ, ϵ, δ
2: initialize w_0, C
3: **for** $t = 1, 2, \ldots, T$ **do**
4: **Local training:**
5: **for** each client $k \in S_t$ **in parallel do**
6: $w_t^k = \text{argmin}(F_i(w^k) + \frac{\mu}{2}||w^k - w_{t-1}||^2)$
7: $w_t^k = w_t^k / \max\left(1, \frac{||w_t^k||}{C}\right)$
8: $\tilde{w}_t^k = w_t^k + n_t^k$
9: **end for**
10: **Model Aggregation:**
11: $w_t = \sum_{k=1}^{N} p_k \tilde{w}_t^k$
12: $\tilde{w}_t^k = w_t + n_D$
13: **end for**

weight updates from the clients as element-wise differences Δ_i^t, just the ones mentioned a few sections ago, but also b_i^t, a proportion of the number of absolute weight differences that lie below C:

$$b_i^t = \mathbb{I}_{||\Delta_i^t||_2 \leq C} \tag{3.13}$$

The complete workflow is shown in Algorithm 4. Notably, to prevent any information about clients' updates that might be revealed by using b_i^t as is, some noising is added to it before averaging it, as is visible in line 13.

Sparse Vector Technique

In its most general form, Sparse Vector Technique is a derivation of Dwork's AboveThreshold algorithm[20]. Algorithm 5 shows AboveThreshold, where a mechanism takes care of iterating over a series of queries or functions, $f_1, \ldots, f_k$ for a dataset D, and a selected noised threshold $\hat{T}$, to approximately select the first instance of the queries that satisfy $f_i(D) + v_i \geq \hat{T}$, where v_i is random noise.

$\top, \bot$ represent whether the query is above the threshold or not, respectively.

The random noise added to each query makes this algorithm differentially private given that a_i is dependent on both $\hat{T}$ and $f_i(D) + v_i$, both random values. The output will often return a correct answer; however, AboveThreshold preserves differential privacy by sometimes returning values that didn't initially lie above $\hat{T}$: $f_i(D) \leq \hat{T} \leq f_i(D) + v_i$.

As Near et al.[21] point out, Sparse Vector Technique receives its name from the fact that it is expected that most of the input vector's values

[20] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4): 211–407, 2013. ISSN 1551-3068. DOI: 10.1561/0400000042. URL http://dx.doi.org/10.1561/0400000042

[21] Joseph P Near and Chike Abuah. Programming differential privacy. *URL: https://uvm*, 2021

Algorithm 4 DPAdaptiveClipping

define: $m, \gamma, \eta_c, \eta_s, \eta_C, z, \sigma_b, \beta$
initialize w^0, C^0
$z_\Delta \leftarrow (z^{-2} - (2\sigma_b)^{-2})^{-1/2}$
for $t = 1, 2, \ldots, T$ **do**
    $\mathcal{Q}^t \leftarrow q_1, ..., q_m \in_R \mathcal{Q}$
    for each client $k \in \mathcal{Q}^t$ **in parallel do**
        $(\Delta_i^t, b_i^t) \leftarrow \textsc{FedAvg}(k, \mathrm{w}^t, \eta_c, C^t)$
    end for
    $\sigma_\Delta \leftarrow z_\Delta C^t$
    $\tilde{\Delta}^t = \frac{1}{m}(\sum_{k \in \mathcal{Q}^t} \Delta_i^t + \mathcal{N}(0, I\sigma_\Delta^2))$
    $\bar{\Delta}^t = \beta \bar{\Delta}^{t-1} + \tilde{\Delta}^t$
    $w^{t+1} \leftarrow \mathrm{w}^t + \eta_s \bar{\Delta}^t$
    $\tilde{b}^t = \frac{1}{m}(\sum_{k \in \mathcal{Q}^t} b_i^t + \mathcal{N}(0, \sigma_b^2))$
    $C^{t+1} \leftarrow C^t \cdot \exp(-\eta_C(\tilde{b}^t - \gamma))$
end for

$\textsc{FedAvg}(k, \mathrm{w}^0, \eta, C)$
$\mathrm{w} = \mathrm{w}^0$
for batch $g \in \mathcal{G}$ **do**
    $\mathrm{w} \leftarrow -\eta \nabla \ell(\mathrm{w}; g)$
end for
$\Delta \leftarrow \mathrm{w} - \mathrm{w}^0$
$b \leftarrow \mathbb{I}_{||\Delta|| \leq C}$
$\Delta' \leftarrow \Delta \cdot \min\left(1, \frac{C}{||\Delta||}\right)$
return (Δ', b)

Algorithm 5 AboveThreshold

define: $D, \{f_k\}, T, \epsilon$
Let $\hat{T} = T + \mathrm{Lap}(\frac{2}{\epsilon})$
for $f_i \in f_k$ **do**
    Let $v_i = \mathrm{Lap}(\frac{4}{\epsilon})$
    if $f_i(D) + v_i \geq \hat{T}$ **then**
        Output $a_i = \top$
        Halt
    else
        Output $a_i = \bot$
    end if
end for

won't exceed $\hat{T}$. In contrast to AboveThreshold where only the first value exceeding $\hat{T}$ was returned, SparseVectorTechnique might

return more than one answer.

To concisely explain how everything is tied together, consider Algorithm 6; the base of AboveThreshold is there. However, there are two important features of SVT:

1. The algorithm iterates over the remaining elements whenever an element exceeds the threshold, it doesn't halt like AboveThreshold.
2. A halt condition is reached whenever the number of elements exceeding the threshold reaches a value c, which is a hyperparameter.

Algorithm 6 SparseVectorTechnique

define: $D, \{f_k\}, T, c, \epsilon, \delta$
If $\delta = 0$ **Let** $\sigma = \frac{2c}{\epsilon}$ **Else Let** $\sigma = \frac{\sqrt{32c \ln \frac{1}{\delta}}}{\epsilon}$
Let $\hat{T}_0 = T + \text{Lap}(\sigma)$
for $f_i \in f_k$ **do**
    Let $v_i = \text{Lap}(2\sigma)$
    if $f_i(D) + v_i \geq \hat{T}_{count}$ **then**
        Output $a_i = \top$
        Let count = count+1
        Let $\hat{T}_{count} = T + \text{Lap}(\sigma)$
    else
        Output $a_i = \bot$
    end if
    if count $\geq c$ **then Halt**
    end if
end for

In the context of FL, the SVT would go over each of the weights gradients and only select the subset S having a value above $\hat{T}$, with the remaining values being set to 0.

Defining c involves either selecting an arbitrary number or choosing a fraction of the parameters $0 \leq \phi \leq 1$.

Percentile Privacy

Percentile Privacy is a simple yet powerful approach to privacy: only the most significant weights are updated at each round. This is what has been previously called selective parameter update by Shokri [22].

An arbitrary percentile threshold $0 \geq \gamma \geq 1$ is set to configure the weight update process. Only those differences (or gradients) whose absolute value lies above the percentile threshold $|\Delta_{\mathbf{w}_i}| \geq \gamma$ will be updated; in other words, the client would only need to send the updates concerning these weights to the server.

Additionally, there is a clipping process on the weight differences

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

resulting after the local training process is concluded, each client calculates the element-wise weight difference $\Delta_{\mathbf{w}}^{k}$, and then these weight differences are clipped to regularize the updates.

$$\Delta_{\mathbf{w}}^{k\top\gamma} = \min_{ij}(\Delta_{\mathrm{w}_{ij}}^{k}, \gamma) \tag{3.14}$$

Here, the subscript $\top\gamma$ represents the clipped values using the γ threshold. Then $\Delta_{\mathbf{w}}^{k\top\gamma}$ is sent to the server. Using FedAvg 1 as an example, plugging in the Percentile Privacy component into the workflow would only require modifying the ClientUpdate section of FedAvg, as is visible in Algorithm 7:

Algorithm 7 PercentilePrivacy

1: initialize w_0
2: **for** $t = 0, 1, \ldots, n-1$ **do**
3: $\quad m = \max(C \cdot K, 1)$
4: $\quad S_t = s_1, \ldots, s_m \in_R S$
5: $\quad$ **for** each client $k \in S_t$ **in parallel do**
6: $\quad\quad w_{t+1}^{k} = \textsc{ClientUpdate}(k, \mathrm{w}_t)$
7: $\quad$ **end for**
8: $\quad m_t = \sum_{k \in S_t} n_k$
9: $\quad \mathrm{w}_{t+1} = \sum_{k \in S_t} \frac{n_k}{m_t} \mathrm{w}_{t+1}^{k}$
10: **end for**

11: ClientUpdate$(k, \mathrm{w}_0, \gamma, q)$:
12: $\mathcal{B}$ = split $\mathcal{P}_k$ into batches of size B
13: $\mathrm{w} = \mathrm{w}_0$
14: **for** each local epoch $e = 1, \ldots, E$ **do**
15: $\quad$ **for** batch $b \in \mathcal{B}$ **do**
16: $\quad\quad \mathrm{w} = \mathrm{w} - \eta \nabla \ell(w; b)$
17: $\quad$ **end for**
18: $\quad \Delta_{\mathrm{w}} = \mathrm{w}_0 - \mathrm{w}$
19: $\quad \Delta_{\mathrm{w}}^{\top\gamma} = \Delta_{\mathrm{w}_i}^{\top\gamma} \begin{cases} 0 & \text{if } |\Delta_{\mathrm{w}_i}| < P_q(\Delta_{\mathrm{w}}) \\ (\Delta_{\mathrm{w}_i})^{\top\gamma} & \text{otherwise} \end{cases}$
20: **end for**
21: return $\Delta_{\mathrm{w}}^{\top\gamma}$

Line 19 of Algorithm 7 makes use of the percentile function $P_q(\cdot)$ to ensure that only those gradients with a value above the significance percentile threshold q are updated.

3.3.4 Secure Aggregation

Secure Aggregation (SecAgg) attempted to solve the problem of secret sharing: how can a secret be divided into a number k of shares, such that it can be reconstructed with all k parts, but no information can be inferred with a set of at most $k-1$ shares? [23]

SecAgg proposed to have secret sharing through a masking process. In short, each client would create a masking vector based on a combination of a private key and public keys from other clients using a pseudo-random generator[24] and share it with all other clients; these vectors would then mask their models and send them back to the server.

The main takeaway from SecAgg is that it allows the server to update its model without knowing the individual contributions from each client, as the masked vectors would cancel out each other during the aggregation process.

However, SecAgg has a problem: What happens when a client drops out and its masking vector is not received for aggregation? Secure Aggregation Plus (SecAgg+) attempts to solve the problem of information decryption if a client drops out.

At a high lever, as Joseph[25] points out, instead of sharing its masking vector with the whole client network, each client shares it with only a subset of k of them. Subsequently, each client calculates a polynomial of degree $k-1$ and plots a series of points in that polynomial, with the y-intercept representing the mask. Lastly, the client shares random points in that polynomial with its peers.

The idea behind it is that if, at any point, the server requires the masking vector from a client who has dropped out, it can look around for other clients that could have the points of the polynomial and attempt to reconstruct the y-intercept mask vector from these points. Figure 3.3 shows graphically what the masking vector process reconstruction looks like.

[23] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017

[24] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017

[25] Brooke Joseph. Secure Aggregation with Flower. https://medium.com/@brookeajoseph17/secure-aggregation-with-flower-24574d84da69, 2023. [Accessed 18-03-2025]

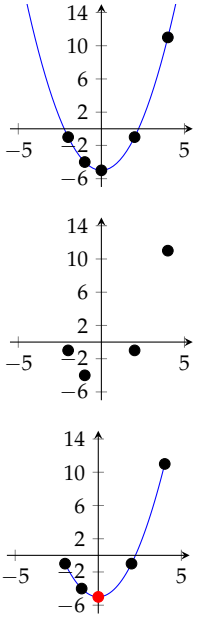

Figure 3.3: Top: Polynomial calculated by a client and randomly placed points to be shared with other clietns. Middle: Points shared with other clients. Bottom: Reconstructed polynomial and mask vector (red) from gathered points.

3.4 Chapter Summary

This chapter comprehensively described many of the most relevant concepts related to Federated Learning, both at a conceptual and technical level. The emphasis shifted from defining what Federated Learning was and how it worked by describing the core workflow areas that shape a regular Federated Learning system to conveying what goes on underneath the hood while the different workflow stages take place, such as aggregation, communication, and privacy preservation. This section provided enough context, terminology, and intuition about how Federated Learning works to understand the rest of the study.

4 Research Methods

Contents

4.1 Chapter Overview

This section outlines the process the data used for this case study underwent to prepare for the simulations, the model used, the method selected to survey the available frameworks and choose the appropriate ones, the parameters used to make the results as comparable as possible, the environment in which the implementations took place, and finally the limitations to this approach.

4.2 Research Design

4.2.1 Data

The data used for this study was Google's Landmark V2 dataset (GLv2) [1], which provides a wide variety of images with features that make them look like a typical dataset in the real world. Notably, this dataset has been previously used for a variety of use cases, such as model performance head-to-head analysis[2] or to benchmark newer model proposals against state-of-the-art algorithms[3].

GLv2 is a dataset used to train classification models. It contains more than 5 million images depicting over 200,000 popular monuments

[1] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google landmarks dataset v2 – a large-scale benchmark for instance-level recognition and retrieval, 2020. URL https://arxiv.org/abs/2004.01804

[2] Kangrui Wang and Xiaobing Yu. Mobilenet and efficientnet demonstration on google landmark recognition dataset. *International Core Journal of Engineering*, 7: 2021. DOI: 10.6919/ICJE.2021037(3).0043

[3] Bingyi Cao, Andre Araujo, and Jack Sim. Unifying deep local and global features for image search. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pages 726–743. Springer, 2020

or landmarks, the main goal of this dataset is to predict the classes of the landmarks.

Notably, the images come in non-uniform resolutions and image aspect ratios, as the images were originally fetched from Wikimedia Commons[4], an open image repository.

Since its introduction, the dataset has gained popularity given how challenging and noisy its images are; often, photos of the same class are taken from different angles, positions, or locations relative to each other, which makes it really difficult to predict correctly; additionally, there are large class representation imbalances, as landmarks such as the Eiffel tower are very well represented with thousands of samples, while others might not even get to ten samples.

To exemplify the in-class noise, Figure 4.1 shows a sample of resized images from the dataset, all belonging to the same landmark *Hauptfriedhof Karlsruhe*; a PyTorch resizing filter was used during the storing step to save up time during the training phase.

The dataset was downloaded using direct download links on the project's Github page. However, as the entirety of the dataset is around half a terabyte in size, a subsample was needed; otherwise, a tremendous amount of computing power would have been required to finish a single epoch.

[4] Wikimedia, "Wikimedia Commons" Wikimedia Commons, https://commons.wikimedia.org/wiki/Main_Page (accessed Mar. 12, 2025).

Figure 4.1: Images belonging to the same category: Hauptfriedhof Karlsruhe, resized using PyTorch.

Two hundred classes were selected at random to subsample the original GLv2 dataset. To avoid under- and over-representation, the classes needed to have between 50 and 1000 samples each. The sample distribution for the dataset is shown in Figure 4.2, it contained just over 22,500 labeled images.

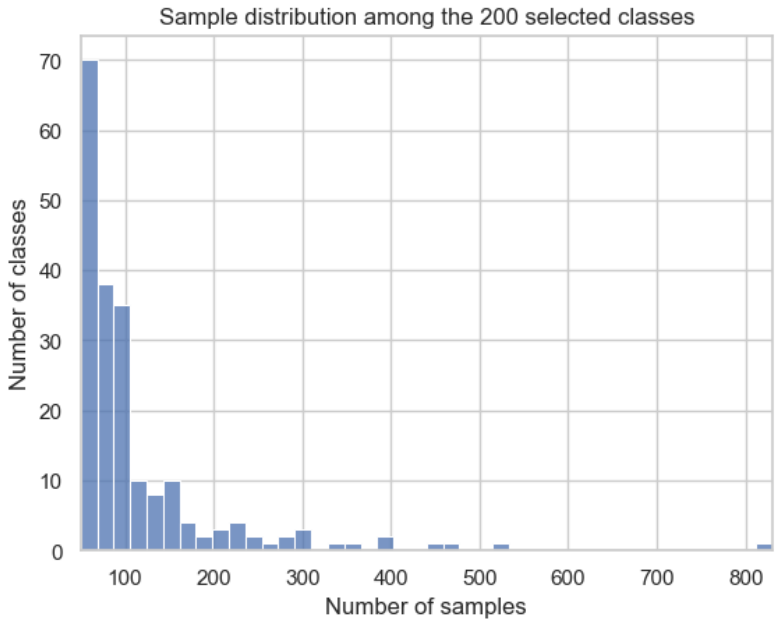

Figure 4.2: Sample distribution across the 200 randomly selected landmark classes. Most classes have less than 200 samples.

The process of acquiring the data, processing it, and sub-sampling it was developed from the ground up. HuggingFace's dataset format was used to store the data due to its quick loading times given that it uses parquet scanning to load and process data, ease of access, and how widespread it is becoming for natural language processing models; using this dataset-storing format would also make sense for a bigger subsample of data or even other datasets, not necessarily text-based ones, but also image-based or other types.

4.2.2 *Model*

There were two primary considerations when it came down to choosing a model, aside from it being designed to handle image-processing tasks:

1. Compared to most cutting-edge image-processing networks, it must have a small size, parameter-wise, which translates to it being lightweight;
2. It must be relevant performance-wise, meaning it must have shown promising results in the past compared to other top-performing models.

Though not obligatory, it was a big plus if it was available for use with PyTorch or had an open-source implementation.

In the end, EfficientNetBo[5] was selected because it checked both boxes, was available with PyTorch, and had a series of pre-trained weight configurations for transfer learning and fine-tuning; specifically, those pre-trained on the Imagenet1k dataset[6], a popular dataset containing one thousand classes as output, and whose input involves images of common objects, arguably the most popular dataset to benchmark new cutting-edge classification models.

Mingxing et al.[7] introduced the EfficientNetBo as part of the EfficientNet family of convolutional neural networks, a series of models that aimed to improve upon state-of-the-art bigger and deeper image classification models by using a reduced number of network parameters.

The EfficientNetBo is the smallest of the models in the aforementioned family, and as such, it is the most lightweight of its kind. Figure 4.3 shows the architecture of the EfficientNetBo network.

The main characteristic that made EfficientNetBo popular is its performance using a relatively modest number of parameters (around 5 million). The network has previously been used to classify images for different tasks, such as brain tumor classification[8], deepfake detection[9],

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 4.3: EfficientNetBo architecture. The output layer contains 1000 using the Pytorch pre-loaded ImageNet1k weights. Table from Mingxing et al.[a]]Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019b. URL http://arxiv.org/abs/1905.11946

[a] [

and has even been used to compare the performance of lightweight models on a variation of the GLv2 dataset![10]

The main reason behind choosing the EfficientNetBo model is simple: It is a lightweight production-calibre model, which makes it easier and more realistic to simulate with. Due to its proven performance, engineers and data scientists could potentially consider this model for their needs, and given that it is a relatively small model, simulations will be carried out quickly.

The method used here was to use the pre-trained weights from the

[5] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019a

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009

[7] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019b. URL http://arxiv.org/abs/1905.11946

[8] Veeranki Goutham, Abdul Sameerunnisa, Sallagundla Babu, and Tugu Bhanu Prakash. Brain tumor classification using efficientnet-bo model. In *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pages 2503–2509, 2022. DOI: 10.1109/ICACITE53722.2022.9823526

[9] Davide Alessandro Coccomini, Nicola Messina, Claudio Gennaro, and Fabrizio Falchi. Combining efficientnet and vision transformers for video deepfake detection. In *International conference on image analysis and processing*, pages 219–229. Springer, 2022

[10] Kangrui Wang and Xiaobing Yu. Mobilenet and efficientnet demonstration on google landmark recognition dataset. *International Core Journal of Engineering*, 7 (3):313–319, 2021

ImageNet1k dataset and fine-tune the last layer using the GLv2 dataset. This approach's main upside was reducing training time, as only one layer required gradient updates; additionally, since fine-tuning freezing the last layer is a common task performed by neural network engineers, this is a task that seems reasonable.

After modifying the last layer to allow for fine-tuning while freezing the rest of the network, the number of trainable weights totaled just below 260 thousand; which is a great fit for this study, as having a lightweight network moving around between the server and the clients vastly reduces runtime and communication overhead.

4.2.3 *Algorithms*

Generally speaking, the expectation was that there would be some overlap regarding algorithm availability among the frameworks selected. However, it was not realistic to assume that the overlap would be big enough to cover all areas, such as aggregation, privacy preservation, or computation burden enhancement; at the end of the day, every framework is allowed to have its strengths and not judged by which algorithms it is missing.

This case study's approach explored the algorithms available for each framework and considered three fundamental workflow areas:

- Aggregation
- Privacy
- Communication

Every framework selected must have had at least three algorithms contained in these areas, and they had to be included if they coincided with those available in other frameworks to allow comparability whenever possible.

4.2.4 *Simulation Uniformity*

The parameters selected to run the simulations were set as similarly as possible to guarantee the highest degree of uniformity. A few settings were considered when choosing the configurations:

1. The simulations will use a 30-client federation, each with a different dataset partition size.
2. At runtime every client will run the same training script, the main difference comes in the data that is used by each client. Naturally, the script used will vary depending on the framework, but the parameters used therein will be uniformly set across simulations.
3. The dataset will be partitioned at runtime unless otherwise specified.

4. The LDA partitioning will ensure that the splits are not uniform; each client will have non-I.i.d. partitions, allowing for more realistic scenarios.
5. The number of communication rounds will be ten, with each sampled client running the training for a total of two local epochs, per communication round, given that this is the most significant driving factor for runtime, it will be kept low just to give an idea of runtime.
6. If possible, the optimizer parameters used will be the same for all matching algorithms. Unless otherwise mentioned, the local training will use an Adam optimizer with a learning rate of 0.01; the communication rounds' sampling rate will be 0.8 (24 clients); training will stop after the tenth communication round and results will be evaluated given the results obtained thereafter.

4.2.5 *Framework Selection*

Selecting the frameworks to be implemented might be the single most important decision for this case study. Even though other related works mentioned in previous sections have suggested similarity at different levels across some frameworks, there is uncertainty as to whether those resolutions still hold. Examining related work was not enough to assess framework selection, especially since many of the frameworks to be analyzed have had a considerable number of changes (or commits) made since these investigations came to light.

Several ML libraries are available nowadays; however, as pointed out in the previous sections, most of the work related to FL has emphasized deep learning as the central ML core upon which to build.

For this case study, there were two stages in choosing the most suitable frameworks:

1. First, surveying popular and widespread open-source FL frameworks, emphasizing reading through the documentation for existing features and looking at the source code directly for algorithm availability, if necessary. Notwithstanding the amount of effort required for this step, it would later be worth it just for the time saved if the right choice was made at this stage, to one's best ability.
2. Second, compare the algorithms and features found during the survey for the contemplated frameworks, contrast overlapping components, and assess potential trade-offs.

As acknowledged in this section, no two frameworks were expected to mirror each other's algorithm availability. Thus, it was necessary to pinpoint what the frameworks brought to the table, if not directly comparable to the others, in order to select the ones that best met the criteria.

After looking initially at the FL frameworks available it was apparent that almost all of them supported PyTorch; in some cases, it was the only one being supported, and in most cases, this was overwhelmingly better supported than others, like Tensorflow or Scikit-learn. Thus, an important criterion that the framework needed to meet was to support PyTorch as a backend ML framework.

4.2.6 *Environment*

Regarding hardware, the framework results were achieved using a MacBook Pro M3 Pro with 11 cores and 36 GB of total memory. The Python versions used were not uniform across all frameworks due to a numpy version dependency; however, at the time of this writing, they might differ from those used during the implementation step. In any case, any special Python version requirements will be indicated.

4.2.7 *Research Limitations*

Despite looking at both framework documentation and source code during the framework survey step, this was not an exhaustive search, and it was expected that some elements would go under the radar; in the end, these frameworks contain dozens of thousands of lines of code each. However, a keyword-search mechanism was used to avoid reading every line and only jumping to function definitions, documentation, and mentions of said keywords in comments throughout the files.

Additionally, this case study will examine the implications and cover the strengths and weaknesses of utilizing the frameworks purely at a simulation level, as a production-level evaluation of the frameworks was not possible for time and budget constraints.

4.3 *Chapter Summary*

This chapter provided insight into the methodology used to achieve the goals mentioned in a previous chapter. It discussed the different resources utilized to carry out the simulations, such as the data, the model, the environment, and the process to survey and choose the frameworks to be analyzed. Additionally, this chapter assessed the investigation's scope regarding the framework survey process, the simulation configuration, and the hardware environment itself.

5 Results

Contents

5.1 Chapter Overview

This chapter will describe the most significant findings regarding feature availability, experiment design, and overall implementation considerations specific to each chosen framework. The results will be tackled in the following order: first, the environment setup required to run simulations on the framework; second, the steps needed to integrate the data into the framework's training mechanism; third, the setup needed to plug an external model into the framework; fourth, algorithm implementation review, including any relevant details related to parameter tuning; fifth, simulation-adjustable parameters; sixth, a review of how the framework stores the global model, the logs, and how they display information mid-simulation; finally, a catch-all subsection where any interesting factor not belonging to any of the previous subsections, such as documentation accessibility and completeness, potential source code adjustments needed, and hardware requirements or limitations, if applicable.

For a review of the source code used, you can visit the repository containing the files at: `https://github.com/antonio-jf/FL`.

5.2 Choosing the frameworks

After surveying the documentation and source code of some of the most popular frameworks, the following components were found for the aggregation, privacy-preservation, and communication areas of interest.

Several privacy algorithms are available across all frameworks, but there is very little overlap among them; it is almost impossible to compare them head-to-head. The approach here is not to rule out any framework based on the algorithms used but instead on whether they lie in any of the broad privacy-preserving categories: Differential Privacy, Homomorphic Encryption (see Appendix 7.2), and Secret Sharing.

Data splitting is not a deal breaker if the frameworks provide no partitioning methods; however, for this case study, it is relevant to know if they are available, as the data set used for the simulations will require partitioning to imitate a federation.

The frameworks commonly provide the Latent Dirichlet Allocation algorithm to partition the datasets using a non-I.i.d. setting (see Appendix 7.2), but other popular partitioning mechanisms, like uniform distribution allocation, are also often available.

Aggregation-wise, FedAvg and FedOpt are standard algorithms that do not require any unique setup to run, like modifying the clients to adhere to resource constraints or running the federation on simulated homogeneous settings, as the algorithm's nature involves a faster

Privacy-preserving Algorithms			
Framework	Diff Privacy	Homomorphic Enc.	Secret Sharing
Tensorflow FL	O		
OpenFL			
Flower	O		O
PaddleFL	O	O	O
PySyft	O	O	O
FATE		O	O
FedML	O	O	O
FedScale	O		
NVFLare	O	O	
FedScope	O		O

Table 5.1: Different privacy preserving algorithms included in the frameworks considered. The table is updated as of March 31, 2025.

Realistic data split		
Framework	Dirichlet	Other
Tensorflow FL	O	
OpenFL	O	
Flower	O	O
PaddleFL		
PySyft		
FATE		
FedML	O	O
FedScale		
NVFLare		
FedScope	O	O

Table 5.2: Different data split approaches by framework. The table is updated as of March 31, 2025.

convergence objective, a purely parameterized approach.

Compression was expected to be available in just a few cases. It was surprising to find out that only a single framework (Federated Scope) made the algorithm available. Nonetheless, its inclusion makes sense for the communication side of the simulations; even if few frameworks include it, communication is a key element of the FL workflow.

In the end, there are four frameworks that provide the most comprehensive set of features for a standard FL implementation, namely NVFlare, Flower, FedML, and Federated Scope.

One could make the case for any of those previously mentioned frameworks; however, here are the frameworks ultimately chosen and the main reason behind them:

Aggregation algorithms and communication overhead			
Framework	FedAvg	FedOpt	Compression
Tensorflow FL	O		
OpenFL	O		
Flower	O	O	
PaddleFL	O		
PySyft			
FATE	O		
FedML	O	O	
FedScale	O		
NVFLare	O	O	
FedScope	O	O	O

Table 5.3: Aggregation- and communication-related algorithms surveyed for several popular frameworks. The table is updated as of March 31, 2025.

1. Going into this case study, Flower was a safe choice regarding how well the community supports the framework and the amount of spotlight it gets as an easy-to-use library. After the survey, it contains most of the components sought for this study.
2. Federated Scope was a choice that became apparent only after the survey, as it seemed like the least supported framework out of all the frameworks surveyed, according to the commit count and last commit date; nonetheless, it possesses some of the most comprehensive algorithms and feature sets.
3. NVFlare was also a sure shot, as it is one of the most well-known frameworks and arguably the most mature in support, low-level configuration, and user experience. The survey showed that it supports most of the areas under consideration.

5.3 NVFlare

5.3.1 Environment

There are a couple of ways to run experiments using NVFlare, and as one can imagine, this leads to two different environment setup requirements. On the one hand, the user can make use of NVFlare via a notebook to orchestrate the simulation; on the other hand, one could also orchestrate the simulation using a series of JSON files, this method allows the user to customize each of the components used throughout the simulation by the framework, which in turn gives the user flexibility to control the simulation job at a lower level. For this study, the simulations ran using the latter setup, which required the

user to choose the components to be used by NVFlare individually.

While this might seem like a difficult task at first, the framework's documentation [1] contains several sample implementations that the user can quickly adapt and override if desired. Moreover, most components could remain unchanged from those used in the documentation unless the user plans to overhaul the framework's core functionality.

The files required to run the simulation were a file containing the components to be used by the clients in the federation to execute their training jobs, a file defining the components used by the server to carry out the client-server communication and model-parameter handling, and a file containing simple meta information for the job at hand.

Notably, NVFlare uses what is called a model learner to handle local training, this model learner is a script used by every client in the federation; defined in this model learner is a regular PyTorch training workflow and several functions used to validate the model, save it, load in each client's data, among others.

Adapting a model learner can be somewhat challenging due to the number of operations one needs to consider when performing the training; nevertheless, NVFlare has sample model learners [2] that can be used as a starting point.

Lastly, a helper file was also created to initialize an EfficientNetBo model with pre-trained weights.

Here's what the project tree looks like for a simple FedAvg simulation.

[1] "NVIDIA FLARE" NVIDIA, 2024. https://nvflare.readthedocs.io/en/main/ (accessed Apr. 07, 2025)

[2] "Model Learner" NVIDIA, 2024. https://nvflare.readthedocs.io/en/main/programming_guide/execution_api_type/model_learner.html (accessed Apr. 07, 2025).

```
project
├── nvflare-fedavg
│   ├── jobs
│   │   └── fedavg
│   │       ├── fedavg
│   │       │   └── config
│   │       │       ├── config_fed_client.json
│   │       │       └── config_fed_server.json
│   │       └── meta.json
│   └── run_simulator.sh
└── pt
    ├── learners
    │   ├── __init__.py
    │   └── model_learner.py
    └── networks
        └── get_model.py
```

5.3.2 Data Integration

The most crucial aspect of data integration is plugging it into the training workflow, that is, the model learner. As mentioned in previous sections, NVFlare handles the training by using the same model learner for all clients and utilizing runtime variables to manage which client is performing the jobs at any given time; plugging the data into this model learner achieves the data integration step of the implementation.

However, there is an inherent problem: whenever a client needs to load its *own* data to perform the local training, it would need to access all data, perform some partitioning mechanism, and then extract only the data that belongs to a given client.

While there are certainly ways of doing this partitioning at runtime, NVFlare doesn't provide any right out of the box; the user has to either program its functions to perform the data partitioning and/or partition the data beforehand, store it somewhere in the host and then, at runtime, access the directory containing the data for a given client given some identity variable.

The main trade-off in choosing one of the approaches mentioned above is reusability against time spent. Creating functions to handle partitioning could be very time-consuming, but in exchange, they are reusable for other datasets. The inverse is true for manually partitioning and storing the data; it is much faster regarding implementation and simulation runtime than the previous approach but is not reusable.

The manual partitioning approach was taken for these simulations, meaning the data was split beforehand and stored separately in a different directory. The data-loading mechanism present in the model learner had to be tweaked to load the data from the directory, and the `site_name` attribute was used as the identity variable to load in the corresponding partition. Since the simulation uses 30 clients as the entire pool, the splitting had to account for each one individually.

5.3.3 Model Integration

The model integration is straightforward for NVFlare. Whenever the clients are up for a training round, a model is initialized, and the parameters are then inherited from the global model; the user only needs to handle the initialization. In this case, the EfficientNetB0 was obtained from the Torchvision module using the pre-trained Imagenet1K weights.

The entire process was handled via a class that initializes both the model and the weights and creates a new forward method. NVFlare then used this class via the model learner.

5.3.4 *Algorithm Implementation*

The algorithms implemented in this framework fall into two different categories: aggregation and privacy preservation. Several algorithms are available for both areas, with privacy-preserving mechanisms implemented as filters for data leaving the server and arriving. For this study, the algorithms made use of were the following:

1. Aggregation: FedAvg and FedOpt.
2. Privacy preservation: Percentile Privacy (PP) [3] and Sparse Vector Technique (SVT) [4].

On the privacy side, PP and SVT were chosen because they are relatively well-known and offer differential privacy guarantees. No pertinent algorithm to communication was found for NVFlare, thus, it was opted to analyze an additional privacy preserving algorithm instead.

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

[4] Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy, 2016. URL https://arxiv.org/abs/1603.01699

Federated Averaging

Moving on to how these algorithms are used in NVFlare, starting with aggregation, FedAvg is the most basic algorithm. It is defined on the server side of the simulation as a workflow. Given the simple nature of FedAvg itself, the customizable parameters that NVFlare provides are the number of clients, the number of training rounds to run, both via the `nvflare` command, and the amount of local aggregation rounds or epochs, configurable via the client JSON definition.

Federated Optimization

FedOpt is also defined on the server side, although this time, it is not defined as a workflow but rather as a shareable generator component, as is visible in Listing 5.1. The framework allows the user to change some parameters related to the optimizer on the server during aggregation; given that this optimizer comes from the PyTorch module, one could modify this definition to suit any other algorithm.

```
{
    "id": "model",
    "path": "pt.networks.get_model.Efnet",
},
{
    "id": "shareable_generator",
    "path": "nvflare.app_opt.pt.fedopt.PTFedOptModelShareableGenerator",
    "args": {
        "device": "cpu",
        "source_model": "model",
        "optimizer_args": {
            "path": "torch.optim.SGD",
            "args": {
                "lr": 0.1
            },
            "config_type": "dict"
```

```
        }
    }
}
```

Listing 5.1: FedOpt job definition in NVFlare

Percentile Privacy

To use PP, one can make use of any of the aggregation algorithms mentioned previously, as this is a privacy-preserving block which can be used on top of aggregation. What's different when using PP is the inclusion of the client-to-server privacy-preserving process, which is achieved using a `task_result_filter` component on the client parameter file. PP requires two parameters: percentile, which modulates the percentage of weights to send back to the server as they are, and gamma, the value at which to truncate the remaining weights.

Sparse Vector Technique

Similarly, to use SVT, a similar setup using a `task_result_filter` is required, though several hyperparameters must be set due to the nature of this algorithm, these are shown in Listing 5.2. NVFlare allows users to tune these using the component definition.

Getting good results from this algorithm is quite challenging, as the number of parameters it uses (5) is large; however, if the user is interested in its privacy guarantees, this is probably the best framework to achieve that.

```
"task_result_filters": [
    {
        "tasks": ["train"],
        "filters":[
            {
            "path": "nvflare.app_common.filters.svt_privacy.SVTPrivacy",
            "args": {
                "fraction": 0.1,
                "epsilon": 0.1,
                "noise_var": 0.1,
                "gamma": 1e-5,
                "tau": 1e-6
            }
            }
        ]
    }
]
```

Listing 5.2: SVT algorithm component definition.

5.3.5 *Simulation-adjustable parameters*

The simulation starts after executing a command with arguments for the job at hand: the name of the job itself, the location where the resulting models and all the logs will be stored, the number of threads

to parallelize workers, and the number of federated learning clients to create; below is an example of the command and the parameters needed. Optionally, the command shown can be incorporated into a bash script to invoke simulations more easily.

```
nvflare simulator "jobs/<job>" --workspace <out_workspace> \
--threads <threads> --n_clients <n_clients>
```

Additionally, a few parameters from the configuration files for both the client and the server are configurable. These include the number of local epochs each client trains its local model over before sending the results back to the server per communication round or the minimum number of clients required for the simulation.

Notably, there is no parameter in NVFlare to regulate the client sample rate during training, which means that every client takes part in every communication round.

5.3.6 *Information-persisting mechanisms*

This section will describe how NVFlare displays information mid-simulation, stores its logging, and stores the resulting model.

From the get-go, it is very noticeable that this framework shows plenty of information. Some of it relates to the model performance at a given point, but most is the framework starting and stopping processes to accommodate threads for client training rounds (see Appendix .1). As a result, the terminal becomes cluttered with information unrelated to the actual model performance, which might become too difficult to follow if monitoring evaluation metrics.

One more straightforward way to follow performance metrics is by looking at the logs, which NVFlare does a fantastic job organizing. After defining the `out_workspace` in the simulation settings, the framework stores the logs separately for the server and each client; this means that every client has its directory, and the logs stored therein are specific to them.

Moreover, inside each directory, there is a `local_model.pt` and a `best_local_model.pt` file to keep track of the models for each client, while on the server directory, a `FL_global_model.pt` is created and used throughout the training.

All in all, this behavior makes it easy to access logs and final models.

5.3.7 *Additional Remarks*

This section briefly discusses four main topics: documentation, hardware considerations, source code, and machine learning framework flexibility.

Regarding documentation, this is a well-documented framework. However, some of the algorithms present here, such as SVT, should be researched elsewhere to fully grasp what each is trying to accomplish, as the framework provides the tools to make them work and a brief description of the functions used but not an in-depth explanation.

In terms of hardware, the simulations ran on a MacBook Pro M3 Pro, which was enough to run the experiments using the algorithms mentioned previously, even using the GPU, as the training ran using PyTorch, and it supported the engine with essentially no source code changes. However, there was a limitation to using a Mac instead of a regular PC with a graphics card: Homomorphic Encryption (HE) could not run, as the Tenseal [5] dependency used by NVFlare to run HE was not available for the Silicon versions of the Mac, at the time of this writing. Any desire to use HE would require the use of different hardware. Production-wise, this is rarely a limitation since no one uses MacBooks to train high-end models, but in this case, for simulation purposes, it is, so it is worthwhile to mention it.

[5] "TenSEAL" OpenMined, Jul. 24, 2023. https://github.com/OpenMined/TenSEAL

Concerning source code changes, the framework is solid. No changes were necessary during the experiments.

Finally, when considering this framework, one must also look at the machine learning frameworks it supports. In this regard, NVFlare supports PyTorch and Tensorflow.

5.4 *FederatedScope*

5.4.1 *Environment*

FedScope is a framework that requires from-source installation. Luckily, the process is relatively easy to follow even if one is not familiar with this sort of procedure; the only requirement is that some basic dependencies, namely PyTorch and torchvision, be installed prior. The installation is documented[6] and should not cause inconvenience if using a fresh virtual environment.

[6] "Installation" FederatedScope, Sep. 06, 2023. https://federatedscope.io/docs/installation/ (accessed Mar. 31, 2025).

Regarding the project tree used for the simulations, there are two different setups available:

1. A `main.py` file handles simulation initialization and job definition, and a parameter file contains the parameters to be loaded and used for the simulation.
2. A single Python notebook is used to define both the jobs and parameters.

The first approach uses a simple project tree with the script runner and the configuration file. Note that the FederatedScope directory holds the framework's files, which are naturally required, though this directory can be stored somewhere else if desired.

```
project
├── FederatedScope
│   └── ...
└── fedavg-job
    ├── job_config.yaml
    └── main.py
```

Running the simulation is then achieved with a single command calling the runner and referencing the parameter file. For instance, from the project directory:

```
python fedavg-job/main.py --cfg fedavg-job/job_config.yaml
```

The second approach, using a Python notebook, is also succinct; this time, the entire setup would involve just a Python notebook file inside the `fedavg-job` directory. This study used the second setup for reasons discussed later in the Additional Remarks subsection.

5.4.2 *Data Integration*

FedScope calls for the user to integrate its data into the framework via a custom source code file to register it and use it via the simulation configuration; the plug-in nature of this process might initially sound complicated, however, the documentation on this process is forthright to follow, and no source code changes are required to let the framework take the data in.

Additionally, if the data being used has a standard format that PyTorch has no problem using, creating the module to plug your data might be all you will ever need to run the simulations on FedScope; which may not be the case if the user is attempting to use a fancier data structure.

The otherwise process mentioned in the previous paragraph was, unfortunately not the only this that was needed for the implementations mentioned in this section. After creating the data-loading file to run simulations and ensure everything was working correctly, several hours had to be invested in getting the trainer to take the data in and access it correctly at runtime. An example of the aforementioned adaptation is shown below in Listing 5.3.

```
1 def _hook_on_batch_forward(self, ctx):
2     # Code modification
3     # x, label = [_.to(ctx.device) for _ in ctx.data_batch]
4     x, label = ctx.data_batch["image"].to(ctx.device), ctx.data_batch["↩
       label"].to(ctx.device)
5     pred = ctx.model(x)
6     if len(label.size()) == 0:
7         label = label.unsqueeze(0)
8     ...
```

Listing 5.3: Source code change to adapted to huggingfaceś data structure.

While this was expected due to how the data was being fed to FedScope (using Huggingface's dataset data structure), the fact that the trainer is not exposed to the user made things trickier to adapt, as source code needed to be modified to suit the data structure; anyhow, this is entirely avoidable if a more standard data structure were fed to the framework instead, and users holding data in uncommon structures might find it easier to translate it to a different format altogether than tweaking around with the framework's code.

Another thing about data integration is that FedScope provides a data partitioning mechanism via LDA, which surprisingly worked well with Huggingface's format. So, there are no significant changes aside from a couple of adaptations of the `LDASplitter` class to access the sample labels, which were necessary for the partitioning component to work. A one liner that had to be adapted is shown below in Listing **??**.

```
# label = np.array([y for x, y in tmp_dataset])
label = np.array([sample["label"] for sample in tmp_dataset])
```

Listing 5.4: One liner adjustment to LDASplitter.

After integrating the dataset, it can then be referenced on the parameter files.

5.4.3 *Model Integration*

Similar to how the data has to be registered directly into the source, the model also needs to be plugged into FedScope. However, integrating the model is rather easy to accomplish if one goes through the data integration step, as the process is similar. The user only needs to adjust two functions: one to load the model in `load_my_net` and the other to call it at runtime `call_my_net`, which makes use of the former.

The same class that was used for NVFlare was utilized here to fit the EfficientNetBo model. Overall, this is a straightforward process, and the documentation makes it seamless.

5.4.4 *Algorithm Implementation*

For FedScope, there was also an emphasis placed on privacy, however, as the framework also provided support for compression, it was implemented as well. Thus, the algorithms tested fall under the categories below.

1. Aggregation: FedAvg and FedOpt.

2. Privacy preservation: Noising before Aggregation (NbAFL) [7].
3. Communication burden reduction: Compression.

[7]

Notably, the framework also provides a Secret Sharing (SS) algorithm as a privacy-preserving alternative to NbAFL; however, an error occurred when running the simulations involving a backend-component script, and it was deemed too time-consuming to fix; ultimately, the simulation using it was scrapped.

It is worth mentioning that FedScope supports attack simulation on the network during training, which comes in a few different flavors. Still, there is a limitation: after looking at the code and examples, it became apparent that no easy way to use these algorithms on own data was implemented. They were almost set out to work specifically for some popular datasets that come pre-packaged with the framework (precisely for FEMNIST and CIFAR-10), but they were not modular enough to be used with external ones.

Finally, before discussing the details of each algorithm, FedScope expects a particular configuration format for each simulation. FedScope provides a class `global_cfg` to extract a template with all available parameters for simulation configuration, including those related to algorithms to be used at runtime.

Even though, cloning every single parameter for a single simulation might seem unnecessary and tedious to sort out, FedScope only takes into consideration those workflows that are active for a simulation, such as FedOpt, compression, quantization, among others.

As a short example of what this configuration template looks like, in Listing 5.5 one can see all customizable parameters for the FedOpt workflow. Notably, the last parameter allows the user to turn on or off the workflow. The user doesn't have to worry about every parameter in each workflow section. The snippet shown below while it does have the parameters `__cfg_check_funcs__` and `__help_info__`, these were not even touched across all simulations; the user can safely ignore these and focus on the actual optimizer-related values.

```
fedopt:
  __cfg_check_funcs__: []
  __help_info__: {}
  annealing: False
  annealing_gamma: 0.5
  annealing_step_size: 2000
  is_ready_for_run: False
  optimizer:
    __cfg_check_funcs__: []
    __help_info__: {}
    is_ready_for_run: False
    lr: 0.01
    type: SGD
  use: False
```

Listing 5.5: FedOpt parameter snippet after clonning the global

configuration.

Federated Averaging

FedAvg is the default algorithm that FedScope uses for aggregation unless overwritten by activating a different one. The main parameters that one would consider changing are the number of clients, the client-sampling rate, the number of local training steps, and the number of communication rounds; modifying these parameters is as simple as changing the values in the global configuration.

Federated Optimization

Running a simulation using FedOpt does involve turning the workflow on the global parameter object; however, utilizing FedOpt naturally also requires setting up a few parameters that were not necessary with plain FedAvg: the server-side optimization algorithm and its learning rate (lr), which are defined separately to the client-side ones. Listing 5.6 shows the previously mentioned difference.

```
# Client-side optimizer definition
cfg.train.optimizer.type = 'SGD'
cfg.train.optimizer.lr = 0.01

# Server-side optimizer definition
cfg.fedopt.optimizer.lr = 0.1
cfg.fedopt.optimizer.type = "SGD"
```

Listing 5.6: Defining client-side and server-side optimization parameters

Nosing before Aggregation Federated Learning

Now, moving on to NbAFL, the framework allows the user to configure four variables:

- noising parameters: `constant, mu, epsilon`.
- regularization: `w_clip`

As one can imagine, since this is a privacy-preserving algorithm not related in any way to aggregation, one must also define an aggregation workflow (FedAvg if not defined). For this simulation, FedAvg was used, and the same parameters used for the regular implementation were used; the only addition was the NbAFL mechanism.

Compression

Finally, compression can be achieved using one of two compression sizes, 8-bit and 16-bit. A couple of simulations ran for each of these configurations, and while it did compress the weights during training (with a couple of source code changes due to some hard coding hidden inside the compression functions), the difference between them

was negligible, the logs showed nothing relevant between the two in comparison to a regular FedAvg job, communication-wise.

5.4.5 *Simulation-adjustable parameters*

Several simulation parameters have been mentioned throughout this section. FedScope has a vast repertoire of configuration settings that allow the user to experiment with all kinds of mechanisms. The global configuration object that it uses contains over 500 lines of parameters.

Some interesting workflows that one could also use are client over-selection, the hyper parameter alpha to partition the data using LDA, early stoppage, regularization via gradient clipping, etc. These are not constraint on any one workflow in particular, so one could mix and match any of them for experimentation purposes.

5.4.6 *Information-persisting mechanisms*

FedScope stores the resulting logs in an `exp` directory unless overwritten via a parameter; `exp` holds logs exclusively regarding model performance among the clients, communication-related statistics and converge times, and the `stdout` shown throughout the simulation (see Appendix .2 and .3).

FedScope shows arguably the most comprehensive set of statistics and model performance data out of the three chosen frameworks, both during training and after training.

Using FedScope, one realization that one must deal with is the global model not being saved; this feature is unavailable, and the user is responsible for altering the source code to save the global model, which is disappointing, considering that this is a fundamental feature.

5.4.7 *Additional remarks*

This framework is challenging to implement, as the user is in charge of making any source code adjustments to suit its requirements. Although the initial data and model setups are not complicated, it is the user's responsibility to make them work inside the training workflow itself; for specific algorithms and features, this might require a lot of effort or none at all, depending on several factors, such as the data format, the model, or the algorithm itself that the user wants to use for experiments.

Several source code changes were required to get all simulations up and running. Therefore, it is imperative to suggest that this framework requires an advanced Python programming level and that the user be comfortable looking at and changing source code.

Finally, a critical remark about this framework, which was hinted at earlier in this section, is that although there are two setups to run the

simulations, one using a Python notebook and the other using a script runner, the latter was impossible to run on the hardware used for all other frameworks.

The impossibility of running the simulations came from an extreme use of memory. The framework overshot the available memory by about 100%, which made the Mac use over 35GB of swap memory and ultimately crash after some minutes, using a tiny subset of the original data (around 5%). Let alone the fact that no support for Mac's silicone GPU was included in the framework, and there is no easy way to override this limitation by making source code changes.

The decision to use a Python notebook was made, and ultimately, it was the only way of running the simulations on the hardware setup used.

The Federated Scope installation required a numpy downgrade to work, which was only possible to satisfy by using a Python 3.9 version; no significant changes to any other component.

5.5 Flower

5.5.1 Environment

One can install Flower using the pip Python package manager. After that, one can create a project tree with the following structure:

```
project
├── fedavg
│   ├── __init__.py
│   ├── client_app.py
│   ├── server_app.py
│   └── task.py
└── pyproject.toml
```

There are in total 4 files that are relevant for setting up a simulation:

- `client_app.py`: defines a FlowerClient Python class and instantiates them at runtime using a wrapper function that calls a baseline EfficientNetB0 model and loads in a data partition separately for each client.
- `server_app.py`: defines the aggregation strategy, evaluation metrics, and initializes the global model.
- `task.py`: defines training and test workflows, the neural newtork class the user is planning on using, and a couple of helper functions to set and get parameters from local models.
- `pyproject.toml`: a parameter file defining simulation-related values, such as the number of clients or the amount of communication rounds.

Then the command `flwr run .` from the project root can be used to start the simulation.

5.5.2 Data Integration

Loading the data is straightforward. The only requirement for using it in a Flower simulation is that it should be partitioned; luckily, the framework provides several choices (iid, Dirichlet, exponential, etc.) to achieve the partitioning. Then, the user must create a function to pass the data to each client, given its partition ID, which is obtained from a runtime variable during the simulation; several examples of this mechanism are available throughout Flower's documentation[8].

[8] "Get started with Flower", Flower. https://flower.ai/docs/framework/tutorial-series-get-started-with-flower-pytorch.html

5.5.3 Model Integration

There is no novel requirement to use an external model with Flower; one can plug any PyTorch model into the simulation without any changes.

5.5.4 Algorithm Implementation

For this framework, the algorithms implemented are listed below.

- Aggregation: FedAvg, FedOpt.
- Privacy preservation: DP with Adaptive Clipping[9], Segure Aggregation++.

[9] Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping, 2022. URL https://arxiv.org/abs/1905.03871

Federated Averaging

To use FedAvg, one must define it on the `server_app.py` file. As part of a `server_fn` function. An example is shown in Listing 5.7. Some parameters that are available for use are: `fraction_fit` to sample a fraction of the total number of clients, `fraction_evaluate` to be used to evaluate the global model every round, `initial_parameters` to initialize the global model with a given set of weights, `evaluate_fn` to pass on a custom evaluation function for Flower to use every validation round, etc. A more comprehensive list of adjustable parameters is available on the Flower documentation page for FedAvg[10].

[10] "FedAvg", Flower, 2025. https://flower.ai/docs/framework/ref-api/flwr.server.strategy.FedAvg.html (accessed Mar. 25, 2025).

```
def server_fn(context: Context):
    # Read from config
    num_rounds = context.run_config["num-server-rounds"]
    fraction_fit = context.run_config["fraction-fit"]

    # Initialize model parameters
    ndarrays = get_weights(Efnet()) # Instantiate an EffnetB0
    parameters = ndarrays_to_parameters(ndarrays)

    # Define strategy
    strategy = SaveModelStrategy(
        fraction_fit=fraction_fit,
```

```
        fraction_evaluate=1.0,
        initial_parameters=parameters
    )
    config = ServerConfig(num_rounds=num_rounds)
    return ServerAppComponents(strategy=strategy, config=config)
```

Listing 5.7: FedAvg definition on the server_app.py file.

Similarly, for FedOpt, the same function can be reused, but instead pluggin in the FedOpt strategy chunk:

```
# Define strategy
strategy = SaveModelStrategy(
    fraction_fit=fraction_fit,
    fraction_evaluate=1.0,
    min_fit_clients=2,
    initial_parameters=parameters,
    eta=0.01,
    eta_l=0.01,
    beta_1=0.9,
    beta_2=0.9
)
```

Federated Optimization

In addition, the same parameters that were available for FedAvg are also available here for FedOpt, with the main difference being the server-side optimizer parameters; FedOpt uses the same ones defined in the original FedOpt article[11].

[11] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2021. URL https://arxiv.org/abs/2003.00295

DP with Adaptive Clipping

Moving on to DP, Flower implements four types of noising with regularization possibilities: client-side with fixed clipping, client-side with adaptive clipping, server-side with fixed clipping, and server-side with adaptive clipping, all mutually exclusive. For this simulation, client-side with adaptive clipping was used, as aside from the documentation available on Flower, there is also a popular paper showing its privacy guarantees; the parameters that are available for tuning are the same as those found in Algorithm 1 of Galen et al.[12].

Depending on which side clipping will be applied, the user might need to modify the client definition to allow them to noise their parameters. The modification to the client is a one liner though[13], so no major adjustments are required.

[12] Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping, 2022. URL https://arxiv.org/abs/1905.03871

[13] "Use Differential Privacy" Flower.ai, 2024. https://flower.ai/docs/framework/how-to-use-differential-privacy.html

Secure Aggregation Plus Plus (SecAgg++)

Finally, to use Secure Aggregation, one must use it in tandem with any aggregation algorithm, the workflow definition is found on the server side, Listing 5.8 shows the definition. Among the most important

parameters specific to the workflow are the number of shares to be used by the workflow, as well as the shares reconstruction threshold.

```
context = LegacyContext(
    context = context,
    config=ServerConfig(num_rounds=num_rounds),
    strategy=strategy,
)

fit_workflow = SecAggPlusWorkflow(
    num_shares=context.run_config["num-shares"],
    reconstruction_threshold=context.run_config["reconstruction-threshold"],
    max_weight=context.run_config["max-weight"],
)

workflow = DefaultWorkflow(
    fit_workflow=fit_workflow
)
```

Listing 5.8: Secure Aggregation Plus workflow definition on the server script.

Though one could hardcode the values for the Secure Aggregation algorithm, they could be passed on to the workflow using the `pyproject.toml` simulation parameter file.

5.5.5 *Simulation-adjustable parameters*

As mentioned previously, the simulation-related parameters are mostly adjustable on the `pyproject.toml` file. Although which parameters are adjustable per simulation varies from workflow to workflow, the main values that one would be changing on a regular basis are the number of training rounds in total, the fraction of clients to be sampled per round, local aggregation rounds, and the number of clients in the simulation. Listing 5.9 shows the format for filling these values in the simulation parameter file.

```
[tool.flwr.app.config]
num-server-rounds = 10
fraction-fit = 0.8
local-epochs = 2

[tool.flwr.federations]
default = "local-simulation"

[tool.flwr.federations.local-simulation]
options.num-supernodes = 30
```

Listing 5.9: Simulation parameters on the pyproject.toml file.

The list of parameters is not forced upon the users; however, if desired, one could, for example, also include values for the network's learning rate or the data-partitioning batch size to be used by Flower; the user would pull these values from the context at runtime.

5.5.6 *Information-persisting mechanisms*

Right out of the box, Flower offers a way to save both the global model and the logs shown throughout training, though one could do without using these persisting mechanisms. To use the saving mechanism, one must first create a wrapper function to handle the parameter-to-array conversion and then the array-saving process using numpy[14]. This is achieved via class inheritance using any aggregation algorithm; any parameters related to aggregation are passed to this new class as usual on the instance definition.

[14] "Save and Load Model Checkpoints", Flower, 2025. https://flower.ai/docs/framework/how-to-save-and-load-model-checkpoints.html (accessed Mar. 25, 2025).

Logging is also defined on the server side as a one-liner:

```
logger.configure(identifier="<expName>",filename="<filename>")
```

One more thing about logging: while Flower calculates performance using the metric functions described in the previous paragraph, the results are only shown at the end of the simulation. This means that if one is using this framework for proof-of-concept purposes, analyzing the global model's performance metrics will require running entire simulations to completion.

5.5.7 *Additional Remarks*

Flower is one of the most well-documented frameworks out there. Many resources are available to learn how to implement the framework's provided algorithms and to use baseline implementations of other algorithms not yet formally adopted by Flower but previously researched and tested by the community.

Additionally, several official events throughout the year showcase new features added to the framework and case studies, which show just how well-supported the framework is.

Currently, Flower is arguably the FL framework that supports the most ML libraries, including, among others, PyTorch, Tensorflow, HuggingFace, and scikitlearn.

Familiarity with the ML library at hand goes a long way toward succeeding with Flower, as there are a few functions that the user has to adapt to its use case; while it may sound complex at the beginning, many quickstart examples are available and well documented[15].

[15] "Flower Framework Documentation", Flower, 2025. https://flower.ai/docs/framework/ (accessed Mar. 25, 2025).

Finally, no source code changes were made for this case study, as almost all the logic used to handle the data, the model, and the model's training and evaluation was made from scratch. This somewhat manual process makes Flower one of the frameworks that the user has to be involved in the most to implement; nevertheless, this is definitely not a bad trait.

Similar to FedScope, Flower also needed a numpy downgrade to work, the same workaround of using Python 3.9 sufficed.

5.6 Direct Result Comparison

This section summarizes the most relevant findings regarding implication and feature diversity among Flower, FedScope, and NVFlare, for the different implementation workflow areas analyzed:

1. Environment
2. Model/Data Integrations
3. Logging and Information Persistence
4. Runtime Results

5.6.1 Environment

Feature	Flower	FedScope	NVFlare
Available through `pip`	Yes	No	Yes
Python notebook compatible	Yes	Yes	Yes
Parameter file	Yes	Yes	Yes
OS-agnostic	Yes	Yes	Yes
Universal hardware support	Yes	No*	Mostly yes**

Table 5.4: Simulation environment considerations and feature availability.

*: No Apple M-series silicon GPU support.
**: There is however, the limitation of using HE on Apple silicon processors, which is not supported.

5.6.2 Model/Data Integrations

Feature	Flower	FedScope	NVFlare
Torchvision model support	Yes	Yes	Yes
Source code changes required	No	Yes*	No
NN architecture changes	No	No	No
Diverse NN architecture support	Yes	Yes	Yes

Table 5.5: Model integration features and considerations.

*: Compression required a few source code changes that were previously hard-coded in order to work.

Feature	Flower	FedScope	NVFlare
Data partitioner	Yes	Yes	No
Pre-partitioning required	No	No	Yes
Source code changes required	No*	No**	No*

Table 5.6: Data integration constraints and features.

*: The training/evaluation workflows are designed by the user, not by the framework.

**: As long as the data is in a format that a PyTorch data-loader would support.

5.6.3 *Logging and Information Persistence*

Feature	Flower	FedScope	NVFlare
Client-wise logging	No	Yes	Yes
Custom metric support	Yes	No	Yes
Final performance summary	Yes	Yes	No
Model checkpoints	Yes*	No	Yes**
Performance logging during training	No***	Yes	Yes

Table 5.7: Regarding what kind of information the frameworks save during and after the simulation.

*: The global model is saved after every communication round is finished.

**: The best and the latest local/global models are saved for each client and the server.

***: Only at the very end of the simulations.

5.6.4 *Runtime results*

Runtimes obtained after running the simulations on the CPU. As stated in previous sections, FedScope does not support the Mac's GPU. To keep every runtime consistent and comparable with each other, every simulation ran on the CPU instead, albeit for a reduced number of communication rounds (10 instead of the original 50).

Algorithm	Flower	FedScope	NVFlare
FedAvg	2:12:10	3:44:34	5:25:25
FedOpt	2:24:06	3:53:58	2:43:38
DP	2:01:43[16]	3:47:10[17]	4:54:12[18]
Other Privacy-Preserving	2:21:18[19]		4:40:25[20]
Compression		3:45:58	

Table 5.8: Runtime comparison for implemented algorithms.

[16] Client-side adaptive clipping
[17] Noise before Aggregation FL
[18] Percentile Privacy
[19] Secure Aggregation
[20] Sparse Vector Technique

5.7 Chapter Summary

This chapter showed the main results of the experiments conducted to implement a Federated Learning workflow using Nvidia Flare, Federated Scope, and Flower. First, the framework selection was made according to the available algorithms found in each framework's documentation and source code findings; then, different simulations took place for each framework, considering the aggregation, communication, and privacy workflow areas.

The results showed that although there is some overlap in algorithm availability, the degree of simulation tuneability often differed across frameworks, and the degree of manual work varied framework-to-framework.

In the end, some of the most relevant factors that could push anyone towards one framework or another were discussed to give a realistic expectation of the implications of using one of the analyzed frameworks over the others.

6 Discussion

Contents

6.1 Section Overview

This chapter's order will first augment the overall results achieved in the previous chapter; it will first discuss key findings to better understand the different implications of using the frameworks, unforeseen difficulties, and advantages or disadvantages compared to their counterparts; second, what the results mean (and don't mean) for anyone looking to get into the Federated Learning scene and is looking to adopt a framework; third, why the analysis performed in this study is relevant for the overall Federated Learning landscape; fourth, a discussion of the limitations regarding scope and hardware constraints; finally, a review of a few personal recommendations related to framework and environment selection.

6.2 Summary

After finalizing the implementations, it is clear that all the frameworks have some clear strengths and weaknesses, as there is no clear "winner" in utility; each has something going for itself.

Flower's implementation is the easiest to follow. However, compared to NVFlare and FedScope, it lacks comprehensive performance logging out-of-the-box, which might be a deal breaker for someone looking to experiment with FL. In contrast, FedScope logs the most performance

metrics out of the three frameworks, hands down. Yet, the user might be reluctant to use it if its data is packaged in a non-standard data structure and/or unwilling to forego coercing it to the framework's original expectation.

Simulating workflows and jobs emphasized the differences in terms of parameter configurations. Even if the algorithms were the same, there were some inherent settings unique to every framework, if ever so slightly.

To exemplify the previous idea, a feature in both FedScope and NVFlare allows the user to instruct the clients in the simulation to send back to the aggregation server only the element-wise difference between the local weights at the end of any given round of FedAvg and those received at the beginning of the training round; this feature is not available in Flower.

One factor often overlooked when evaluating a library is how "polished" the actual source code is. Luckily, the frameworks used in this case study did not require many source code changes to adapt to the data and/or model; however, this doesn't mean that implementing FL will be a walk in the park. The engineer in charge of using a framework must be comfortable with programming general-purpose ML workflows.

A significant consideration that became clear as the implementation work began was that some of these frameworks (including some not directly used in this case study) supported novel and cutting-edge algorithms, such as inference attack simulations by FedScope; however, this support was not straightforward to adopt and/or was in an experimental phase, which ultimately made them challenging to use for proof-of-concept experiments.

Naturally, some of the implementation results shown here will differ to those potentially achieved using other datasets and/or models. Some of the source code changes made for FedScope could have been avoided if the data had been in a more standard format, or if FedScope had supported natively HuggingFace's data structure; there were some other source code changes, however, that were necessary, as there were some hard-coded lines that would have made it impossible for the workflows to work, such as <compression>.

6.3 *Interpretations*

After looking at what each framework offers, it is clear that no framework rules above the other. Different factors could make a user choose a framework over the others, like algorithm configurations or availability, whether the user feels uneasy modifying or adapting source code or hardware limitations. Individual preference notwithstanding,

there are reasons to select any of them. No one framework fits every profile and/or requirement; whoever is in charge of using these libraries should analyze what the objectives are, how well these are met using one over the others, and what challenges or implications are expected to be undertaken throughout experimentation.

Naturally, one of the main goals of the FL frameworks described in this study (and others not tackled as well) is that any person reasonably familiar with ML, especially DL, should be able to carry out FL experimentations without the need to spend vast amounts of time worrying about what is going on underneath the hood. While they do achieve this for the most part, special cases call for extraordinary measures; it is unrealistic to expect that the frameworks are universally case-agnostic. If a unique data structure needs to be handled, there might be a need to either adjust the source code or the underlying data structure; if a unique experiment configuration not initially available requires testing, like client-dropout resilience, entire workflows might then be subjected to analysis and adaptation in order to use them with some frameworks.

The person in charge of the FL implementation should expect at least some manual labor to get everything working, either on the user side (like those concerning special model architectures or tweaking around the data format) or, in the worst-case scenario, on the underlying framework-side code. Some Python programming proficiency goes a long way toward achieving good results with FL.

6.4 *Implications*

The results achieved throughout this study are relevant because they show firsthand what the frameworks offer and what the user can expect from them. As mentioned a few chapters ago, some articles were written about FL frameworks surveying features and comparing them; however, this study went beyond and put them to the test, implementing different algorithms while also plugging in new data and a new model, providing a better perspective on the implications of adopting them for experimental purposes. The insights and examples showcased along the way will hopefully guide ML practitioners interested in but undecided about adopting or committing to an FL framework.

6.5 *Limitations*

This case study's approach emphasized simulation to empirically test the frameworks, limiting the value users can get from this information to the experimental phase of FL adoption. Nonetheless, certain aspects

remain the same when moving to a production-based environment, such as each algorithm's parameters or the training workflow scripts. Of course, the user is responsible for researching the implications of moving from the experimental phase to the production counterpart; the implications of this process were outside the scope of this case study. Another limitation was the hardware environment used to test the frameworks, particularly the dependency issues using NVFlare and the overall framework's lack of support when using FedScope. It is hard to blame the developers of these frameworks, as supporting Apple's silicon processors is a plus for any ML-related library. However, not supporting them is rarely an issue since nobody uses M-Series processors to train large ML models. Using a different hardware setup would have allowed for better framework implementation testing across the board, which would have involved, among other things, using HE as a privacy-preserving algorithm alternative and uniform on-GPU training.

6.6 *Recommendations*

The vast majority of frameworks use PyTorch as their main backend engine. Since most research uses the library nowadays, it feels natural to recommend that anyone interested in FL stick to PyTorch for DL experiments, even though some frameworks now support other popular libraries like TensorFlow.

Regarding hardware, the main recommendation is to experiment using a CUDA-supported GPU processor; given that this is backed almost everywhere by ML and FL frameworks, the user should not have any significant concerns about dependency and feature support.

Finally, the user must also consider which algorithms are available in each framework and the privacy, convergence, or model performance goals to make an informed decision. At the time of this writing, some of the leading research areas for FL are privacy and convergence; the algorithm list shown here might differ as time passes, and it is a sane idea to look out for any area of interest.

6.7 *Chapter Summary*

The main results underlined in the previous chapter were discussed to better grasp the framework implementations' outcomes and implications. There are certain limitations to the approach taken in this case study that naturally play a role in the scope of the results; this role's influence is addressed to clarify further what these results should (and should not) mean for the reader. Lastly, given the implementation findings, a few essential recommendations were given as a baseline for

anyone looking to test these frameworks.

7 Conclusions

Contents

7.1 Final Remarks

This thesis involved implementing popular Federated Learning frameworks to test their general utility in a comparable manner, understand their strengths empirically, and determine how an implementation with external components would shape their use.

A well-known dataset, Google Landmarks v2, was downloaded and randomly subsampled to test the adaptability of the frameworks and the considerations that one would need to take into account when using them; the same goes for the employed model, EfficientNet B0, as understanding the implications of plugging in both model and data is critical in any ML project. It was important to get a picture of the ramifications of a standard Federated Learning implementation.

Assessing the ease of implementation from the standpoint of a machine learning engineer with a theoretical understanding of Federated Learning permitted the technical analysis of each framework's components at a simple-to-follow algorithm configuration level.

Different popular Federated Learning-related algorithms were implemented using both documentation available on each framework's website and source code. A special emphasis was placed on having algorithms related to the same operation area, such as aggregation, privacy, or communication, to have the most directly comparable results. Furthermore, reviewing the documentation of the frameworks for the algorithms and components used at each step of the way did help put the simulations together, which speaks to the integrity and well-documented state of the frameworks' information resources.

In the end, the spectrum of implementation gave a broad perspective

on using algorithms that are as common as the aggregation algorithm FedAvg or those that are not so widely adopted, like the Sparse Vector Technique privacy algorithm, which is available only in NVFlare and pretty much nowhere else.

The results showed that although the algorithms and parameters were similar across all frameworks, specific configurations and features made certain aspects exclusive to some frameworks and not others, even some fundamental elements like logging or model persistence, which one might consider vital for choosing a framework over another, are not always present.

The findings provided evidence that the hypothesis proposed in this case study was reasonable and accurate based on the efforts presented throughout this document; the approach emphasized the similarities and contrasts at diverse component levels by describing matching features and key workflow areas.

Furthermore, even though the frameworks make implementing Federated Learning as easy as possible, one should not take carrying out the task lightly. Undoubtedly, a manual factor is involved, whether on the data, the model, or any framework-related workflow, regardless of which framework is used.

Nonetheless, reviewing the documentation of the frameworks for the algorithms and components used at each step of the way did help put the simulations together, which speaks to the integrity and well-documented state of the frameworks' information resources.

Finally, this work provided guidance on Federated Learning framework selection, as mentioned at the beginning, by empirically comparing the implementation components, however distinct they were, and giving insight into their ins and outs.

7.2 *Further Studies*

There is a wide range of opportunity areas to further investigate the implications of using FL frameworks. Of course, an increase in the number of frameworks analyzed would significantly increase the scope of this case study's goal; going deeper into specifics regarding resource usage or other performance-specific facets would, among other things, tackle the comparison at a lower level. During the development of this case study, some ideas came to mind for further studies:

- Explore the implications of moving these FL systems to a production-based environment.
- Breaking down framework time-, computation-, and resource usage efficiency.
- Implications of developing newer algorithms and plugging them

into the frameworks.

Bibliography

Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping, 2022. URL https://arxiv.org/abs/1905.03871.

Albert Pita Argemi. Design, implementation, and analysis of a cloud federated learning architecture. Technical report, Universitat Politècnica de Catalunya, 2023.

Seun Solomon Bakare, Adekunle Oyeyemi Adeniyi, Chidiogo Uzoamaka Akpuokwe, and Nkechi Emmanuella Eneh. Data privacy laws and compliance: A comparative review of the eu gdpr and USA regulations. *Comput. sci. IT res. j.*, 5(3):528–543, 3 2024.

Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework. 7 2020. URL http://arxiv.org/abs/2007.14390.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

California State Legislature. California consumer privacy act (ccpa), 2018. URL https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375.

Bingyi Cao, Andre Araujo, and Jack Sim. Unifying deep local and global features for image search. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pages 726–743. Springer, 2020.

Bart van der Sloot Chris Jay Hoofnagle and Frederik Zuiderveen Borgesius. The european union general data protection regulation: what it is and what it means*. *Information & Communications Technology*

Law, 28(1):65–98, 2019. DOI: 10.1080/13600834.2019.1573501. URL https://doi.org/10.1080/13600834.2019.1573501.

Davide Alessandro Coccomini, Nicola Messina, Claudio Gennaro, and Fabrizio Falchi. Combining efficientnet and vision transformers for video deepfake detection. In *International conference on image analysis and processing*, pages 219–229. Springer, 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2013. ISSN 1551-3068. DOI: 10.1561/0400000042. URL http://dx.doi.org/10.1561/0400000042.

Veeranki Goutham, Abdul Sameerunnisa, Sallagundla Babu, and Tugu Bhanu Prakash. Brain tumor classification using efficientnet-bo model. In *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pages 2503–2509, 2022. DOI: 10.1109/ICACITE53722.2022.9823526.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019. URL https://arxiv.org/abs/1811.03604.

Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. 7 2020. URL http://arxiv.org/abs/2007.13518.

Chaoyang He, Alay Dilipbhai Shah, Zhenheng Tang, Di Fan1Adarshan Naiynar Sivashunmugam, Keerti Bhogaraju, Mita Shimpi, Li Shen, Xiaowen Chu, Mahdi Soltanolkotabi, and Salman Avestimehr. Fedcv: A federated learning framework for diverse computer vision tasks, 2021. URL https://arxiv.org/abs/2111.11066.

H.R.2989 - 117th Congress. Financial transparency act of 2021, 2021. URL https://www.congress.gov/bill/117th-congress/house-bill/2989.

Chao Huang, Jianwei Huang, and Xin Liu. Cross-silo federated learning: Challenges and opportunities, 2022. URL https://arxiv.org/abs/2206.12949.

Yaochu Jin, Hangyu Zhu, Jinjin Xu, and Yang Chen. *Federated Learning: Fundamentals and Advances*. Springer Nature Singapore, 2023. ISBN 9789811970832. DOI: 10.1007/978-981-19-7083-2. URL http://dx.doi.org/10.1007/978-981-19-7083-2.

Brooke Joseph. Secure Aggregation with Flower. https://medium.com/@brookeajoseph17/secure-aggregation-with-flower-24574d84da69, 2023. [Accessed 18-03-2025].

Sai Praneeth Karimireddy, Narasimha Raghavan Veeraragavan, Severin Elvatun, and Jan F. Nygård. Federated learning showdown: The comparative analysis of federated learning frameworks. In *2023 8th International Conference on Fog and Mobile Edge Computing, FMEC 2023*, pages 224–231. Institute of Electrical and Electronics Engineers Inc., 2023. ISBN 9798350316971. DOI: 10.1109/FMEC59375.2023.10305961.

Harmandeep Kaur, Veenu Rani, Munish Kumar, Monika Sachdeva, Ajay Mittal, and Krishan Kumar. Federated learning: a comprehensive review of recent advances and applications. *Multimedia Tools and Applications*, 83(18):54165–54188, November 2023. ISSN 1573-7721. DOI: 10.1007/s11042-023-17737-0. URL http://dx.doi.org/10.1007/s11042-023-17737-0.

Harmandeep Kaur, Veenu Rani, Munish Kumar, Monika Sachdeva, Ajay Mittal, and Krishan Kumar. Federated learning: a comprehensive review of recent advances and applications. *Multimedia Tools and Applications*, 83:54165–54188, 5 2024. ISSN 15737721. DOI: 10.1007/s11042-023-17737-0.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, April 2023. ISSN 2326-3865. DOI: 10.1109/tkde.2021.3124599. URL http://dx.doi.org/10.1109/TKDE.2021.3124599.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020. URL https://arxiv.org/abs/1812.06127.

Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy, 2016. URL https://arxiv.org/abs/1603.01699.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023. URL https://arxiv.org/abs/1602.05629.

Don Moon. [vLLM - Quantization] bitsandbytes: 8-bit Optimizers, LLM.int8(), qlora, and k-bit inference scaling laws. https://medium.com/byte-sized-ai/vllm-quantization-bitsandbytes-efaec31f00df, 2024. [Accessed 25-02-2025].

Joseph P Near and Chike Abuah. Programming differential privacy. *URL: https://uvm*, 2021.

Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. A performance evaluation of federated learning algorithms. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, New York, NY, USA, December 2018. ACM.

European Parliament. Regulation (EU) 2016/679 of the European Parliament and of the Council, 2016. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

European Parliament. Regulation (eu) 2024/1689 of the European Parliament and of the Council, 2024. URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R1689.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. 2 2020. URL http://arxiv.org/abs/2003.00295.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2021. URL https://arxiv.org/abs/2003.00295.

G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards,

Micah J. Sheller, Sarthak Pati, Prakash Narayana Moorthy, Shih han Wang, Prashant Shah, and Spyridon Bakas. Openfl: An open-source framework for federated learning. 5 2021. DOI: 10.1088/1361-6560/ac97d9. URL http://arxiv.org/abs/2105.06413http://dx.doi.org/10.1088/1361-6560/ac97d9.

Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona Flores, and Andrew Feng. Nvidia flare: Federated learning from simulation to real-world. 10 2022. DOI: 10.48550/arXiv.2210.13291. URL http://arxiv.org/abs/2210.13291http://dx.doi.org/10.48550/arXiv.2210.13291.

Sachinsoni. Introduction to Model Quantization. https://medium.com/@sachinsoni600517/introduction-to-model-quantization-4effc7a17000, 2023. [Accessed 25-02-2025].

Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015a.

Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015b.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019a.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019b. URL http://arxiv.org/abs/1905.11946.

Kangrui Wang and Xiaobing Yu. Mobilenet and efficientnet demonstration on google landmark recognition dataset. *International Core Journal of Engineering*, 7:2021. DOI: 10.6919/ICJE.2021037(3).0043.

Kangrui Wang and Xiaobing Yu. Mobilenet and efficientnet demonstration on google landmark recognition dataset. *International Core Journal of Engineering*, 7(3):313–319, 2021.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15: 3454–3469, 2020. DOI: 10.1109/TIFS.2020.2988575.

Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google landmarks dataset v2 – a large-scale benchmark for instance-level recognition and retrieval, 2020. URL https://arxiv.org/abs/2004.01804.

Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope: A flexible federated learning platform for heterogeneity. 4 2022. URL http://arxiv.org/abs/2204.05011.

Chenming Xu and Yunlong Mao. An improved traffic congestion monitoring system based on federated learning. *Information*, 11(7), 2020. ISSN 2078-2489. DOI: 10.3390/info11070365. URL https://www.mdpi.com/2078-2489/11/7/365.

Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. Pysyft: A library for easy federated learning. *Federated learning systems: Towards next-generation AI*, pages 111–139, 2021.

Appendices

Glossary

This appendix shows some concepts that were alluded to in the case study but were not revisited. Thus, they appear here to give a brief conceptual description.

Cross-device Federated Learning

As mentioned by Huang et al.[1], "in cross-device FL, clients are small distributed entities (e.g., smartphones, wearables, and edge devices), and each client is likely to have a relatively small amount of local data".

Cross-silo Federated Learning

To quote Huang et al.[2], "in cross-silo FL, (...) clients are typically companies or organizations (e.g., hospitals and banks). The number of participants is small (...), and each client is expected to participate in the entire training process.

Homomorphic Encryption

According to Jin et al.[3]:

> Homomorphic Encryption is a type of asymmetric encryption algorithms to allow two or more ciphertexts of the data encrypted with the same public key to do algebra computation directly.

Essentially, it allows for data to be encrypted in such a way that any algebraic operation performed on the encrypted data has the same result that it would've had, had it been applied to plaintext data.[4] [5]

Latent Dirichlet Allocation (LDA)

A data-partitioning algorithm that takes a vector of parameters $\boldsymbol{\alpha}$, to draw probability values from the Dirichlet distribution.

The most important properties of LDA being that, as each of the elements of $\boldsymbol{\alpha}$ become bigger, the samples drawn from the distribution are closer between them, and in contrast, as the values of $\boldsymbol{\alpha}$ get closer to 0, the probabilities are more unstable and differ the most between them.

The difference in values drawn from a Dirichlet distribution is shown below, where diverse values for α are considered.

[1] Chao Huang, Jianwei Huang, and Xin Liu. Cross-silo federated learning: Challenges and opportunities, 2022. URL https://arxiv.org/abs/2206.12949

[2] Chao Huang, Jianwei Huang, and Xin Liu. Cross-silo federated learning: Challenges and opportunities, 2022. URL https://arxiv.org/abs/2206.12949

[3] Yaochu Jin, Hangyu Zhu, Jinjin Xu, and Yang Chen. *Federated Learning: Fundamentals and Advances*. Springer Nature Singapore, 2023. ISBN 9789811970832. DOI: 10.1007/978-981-19-7083-2. URL http://dx.doi.org/10.1007/978-981-19-7083-2

[4] Yaochu Jin, Hangyu Zhu, Jinjin Xu, and Yang Chen. *Federated Learning: Fundamentals and Advances*. Springer Nature Singapore, 2023. ISBN 9789811970832. DOI: 10.1007/978-981-19-7083-2. URL http://dx.doi.org/10.1007/978-981-19-7083-2

[5] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. Pysyft: A library for easy federated learning. *Federated learning systems: Towards next-generation AI*, pages 111–139, 2021

$\alpha = 0.1$	0.9999265	0.0000733	0.0000002
$\alpha = 10$	32.5244233	36.963326	30.512251
$\alpha = 1000$	33.3951258	32.800069	33.804805

In federated learning, the probabilities would serve the purpose of representing what the label proportions should be for any given client, during the dataset-partitioning step. If one assumes a dataset with a class C that comprises 1000 samples, then by the first row of the previous table, the first client would be assigned all samples; however, as α gets increasingly large, the label distributions become more uniform.

Non-I.i.d. distributions

In the context of federated learning, non-I.i.d. refers to the data distributions differing from client to client, which can usually be categorized into attribute skew and label skew[6,7]:

1. attribute skew: any two clients do not share the same feature space, $S_1, S_2 \subseteq X, S_1 \neq S_2$; in other words, each client's datasets might contain different predictors.
2. label skew: the label distributions might differ significantly among the clients, for instance, a client might contain certain classes that are not present anywhere else.

[6] Yaochu Jin, Hangyu Zhu, Jinjin Xu, and Yang Chen. *Federated Learning: Fundamentals and Advances*. Springer Nature Singapore, 2023. ISBN 9789811970832. DOI: 10.1007/978-981-19-7083-2. URL http://dx.doi.org/10.1007/978-981-19-7083-2

[7] Harmandeep Kaur, Veenu Rani, Munish Kumar, Monika Sachdeva, Ajay Mittal, and Krishan Kumar. Federated learning: a comprehensive review of recent advances and applications. *Multimedia Tools and Applications*, 83(18): 54165–54188, November 2023. ISSN 1573-7721. DOI: 10.1007/s11042-023-17737-0. URL http://dx.doi.org/10.1007/s11042-023-17737-0

Framework-related Logging

This appendix includes some logging examples for NVFlare, Federated Scope, and Flower, respectively, showing information provided by the frameworks during and after training.

.1 *NVFlare: during training*

There is no easy way to show the amount of low level information that NVFlare includes in its logging for an FL task but some of the most relevant elements will be displayed below.

```
INFO - Connection [CN00002 127.0.0.1:58495 => 127.0.0.1:52973] is created: PID: 90896
INFO - PyTorch version 2.4.1 available.
INFO - registered aux handler for topic __end_run__
INFO - registered aux handler for topic __do_task__
INFO - Register blob CB for channel='aux_communication', topic='*'
INFO - registered aux handler for topic RM.RELIABLE_REQUEST
INFO - registered aux handler for topic RM.RELIABLE_REPLY
INFO - enabled reliable message: max_request_workers=20 query_interval=2.0
INFO - [identity=site-1, run=simulate_job]: Got learner: CustomModelLearner
INFO - registered aux handler for topic fed.event
INFO - [identity=site-1, run=simulate_job]: client runner started
INFO - Initialize ClientRunner for client: site-1
INFO - Received from simulator_server server. getTask: train size: 17.3MB (17324980 Bytes)
    time: 0.049540 seconds
INFO - pull_task completed. Task name:train Status:True
INFO - [identity=site-1, run=simulate_job, peer=simulator_server,
    peer_run=simulate_job]: got task assignment: name=train,
    id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]:
    invoking task executor ModelLearnerExecutor
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Client trainer got
    task: train
...
```

```
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Client identity: site-1
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: training acc
    (None): 0.3252
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: validation acc
    (None): 0.4386
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Evaluation finished.
    Returning result
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Current/Total Round: 4/10
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Client identity: site-1
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Local steps per epoch: 16
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Local epoch site-1:
    1/2 (lr=0.01)
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Local epoch site-1:
    2/2 (lr=0.01)
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: val_acc_local_model: 0.2281
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]:
    val_loss_local_model: 6.9953
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: Local epochs
    finished. Returning FLModel
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: finished processing
    task
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: try #1: sending task
    result to server
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: checking task ...
INFO - broadcast: channel='aux_communication', topic='__task_check__',
    targets=['server.simulate_job'], timeout=5.0
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: start to send task
    result to server
```

```
INFO - Starting to push execute result.
INFO -  SubmitUpdate size: 17.3MB (17324948 Bytes). time: 0.147667 seconds
INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job,
    task_name=train, task_id=44cb1d05-1591-4a29-a4ea-d6cf9dcb6329]: task result sent to
    server
INFO - Finished one task run for client: site-1 interval: 2 task_processed: True
INFO - Clean up ClientRunner for : site-1
INFO - Connection [CN00002 Not Connected] is closed PID: 90896
```

.2 *Federated Scope: during training*

Below is a piece of the resulting logging shown by Federated Scope throughout the training step. The timestamp present in the logs has been removed for the sake of space.

```
INFO: ----------- Starting a new training round (Round #7) -------------
INFO: {'Role': 'Client #8', 'Round': 7, 'Results_raw': {'train_loss': 2694.822188,
'train_loss_regular': 0.0, 'train_acc': 0.250389, 'train_total': 643,
'train_avg_loss': 4.191014}}
INFO: {'Role': 'Client #14', 'Round': 7, 'Results_raw': {'train_loss': 2861.305232,
'train_loss_regular': 0.0, 'train_acc': 0.300709, 'train_total': 705,
'train_avg_loss': 4.058589}}
INFO: {'Role': 'Client #16', 'Round': 7, 'Results_raw': {'train_loss': 2647.232286,
'train_loss_regular': 0.0, 'train_acc': 0.160596, 'train_total': 604,
'train_avg_loss': 4.382835}}
INFO: {'Role': 'Client #21', 'Round': 7, 'Results_raw': {'train_loss': 2751.699912,
'train_loss_regular': 0.0, 'train_acc': 0.296243, 'train_total': 692,
'train_avg_loss': 3.976445}}
INFO: {'Role': 'Client #15', 'Round': 7, 'Results_raw': {'train_loss': 2868.603226,
'train_loss_regular': 0.0, 'train_acc': 0.179059, 'train_total': 659,
'train_avg_loss': 4.352964}}
INFO: {'Role': 'Client #7', 'Round': 7, 'Results_raw': {'train_loss': 2588.328406,
'train_loss_regular': 0.0, 'train_acc': 0.083485, 'train_total': 551,
'train_avg_loss': 4.697511}}
INFO: {'Role': 'Client #9', 'Round': 7, 'Results_raw': {'train_loss': 2599.20856,
'train_loss_regular': 0.0, 'train_acc': 0.192893, 'train_total': 591,
'train_avg_loss': 4.397984}}
INFO: {'Role': 'Client #1', 'Round': 7, 'Results_raw': {'train_loss': 1757.632658,
'train_loss_regular': 0.0, 'train_acc': 0.234192, 'train_total': 427,
'train_avg_loss': 4.116236}}
INFO: {'Role': 'Client #4', 'Round': 7, 'Results_raw': {'train_loss': 3221.109741,
'train_loss_regular': 0.0, 'train_acc': 0.269531, 'train_total': 768,
'train_avg_loss': 4.194153}}
```

.3 *Federated Scope: training summary*

The following log shows the comprehensive training summary provided by Federated Scope right after the training process has finished.


```
{'client_best_individual': {'test_loss_regular': 0.0,
  'test_loss': 304.4866693019867,
  'test_acc': 0.40939597315436244,
  'test_total': 78.0,
  'test_avg_loss': 3.479367256164551},
 'client_summarized_weighted_avg': {'test_loss_regular': 0.0,
  'test_loss': 468.0534716824199,
  'test_acc': 0.29093050647820967,
  'test_total': 113.2,
  'test_avg_loss': 3.84152244236781},
 'client_summarized_avg': {'test_loss_regular': 0.0,
  'test_loss': 434.8603404760361,
  'test_acc': 0.290787734950864,
  'test_total': 113.2,
  'test_avg_loss': 3.8372677405675253},
 'client_summarized_fairness': {'test_loss_regular_entropy': 3.401197381662155,
  'test_loss_regular_cos1': nan,
  'test_loss_regular_top10%': 0.0,
  'test_loss_regular_bottom10%': 0.0,
  'test_loss_regular_max': 0.0,
  'test_loss_regular_min': 0.0,
  'test_loss_regular_top_decile': 0.0,
  'test_loss_regular_bottom_decile': 0.0,
  'test_loss_regular_std': 0.0,
  'test_loss_entropy': 3.367273876237825,
  'test_loss_cos1': 0.9607077584935093,
  'test_loss_top10%': 714.9895622730255,
  'test_loss_bottom10%': 323.17033886909485,
  'test_loss_max': 1009.4778943061829,
  'test_loss_min': 304.4866693019867,
  'test_loss_top_decile': 524.8931250572205,
  'test_loss_bottom_decile': 347.6368417739868,
  'test_loss_std': 125.63722621540347,
  'test_total': 113.2,
  'test_acc_std': 0.05863868056055292,
  'test_acc_bottom_decile': 0.21359223300970873,
  'test_acc_top_decile': 0.38596491228070173,
  'test_acc_min': 0.17647058823529413,
  'test_acc_max': 0.40939597315436244,
  'test_acc_bottom10%': 0.1847357124668049,
```

```
 'test_acc_top10%': 0.3971334964651534,
 'test_acc_cos1': 0.9802675243877295,
 'test_acc_entropy': 3.380448423391053,
 'test_avg_loss_std': 0.17682862403943256,
 'test_avg_loss_bottom_decile': 3.633758306503296,
 'test_avg_loss_top_decile': 4.096480369567871,
 'test_avg_loss_min': 3.479367256164551,
 'test_avg_loss_max': 4.144439697265625,
 'test_avg_loss_bottom10%': 3.5058191617329917,
 'test_avg_loss_top10%': 4.120412667592366,
 'test_avg_loss_cos1': 0.9989399168670542,
 'test_avg_loss_entropy': 3.400131957019693}}
```

·4 *Flower: during training*

Here's a snippet of the logging shown by Flower during training. The timestamp present in the logs has been removed for the sake of space.

```
server.py:113 | [ROUND 5]
server.py:226 | configure_fit: strategy sampled 24 clients (out of 30)
server.py:240 | aggregate_fit: received 24 results and 0 failures
server.py:178 | configure_evaluate: strategy sampled 30 clients (out of 30)
server.py:192 | aggregate_evaluate: received 30 results and 0 failures
server.py:112 |
server.py:113 | [ROUND 6]
server.py:226 | configure_fit: strategy sampled 24 clients (out of 30)
server.py:240 | aggregate_fit: received 24 results and 0 failures
server.py:178 | configure_evaluate: strategy sampled 30 clients (out of 30)
server.py:192 | aggregate_evaluate: received 30 results and 0 failures
server.py:112 |
erver.py:113 | [ROUND 7]
server.py:226 | configure_fit: strategy sampled 24 clients (out of 30)
server.py:240 | aggregate_fit: received 24 results and 0 failures
server.py:178 | configure_evaluate: strategy sampled 30 clients (out of 30)
server.py:192 | aggregate_evaluate: received 30 results and 0 failures
```

·5 *Flower: training summary*

Note the following result summary shown by flower. The loss history shown at the top and the accuracy displayed at the end come with flower, the rest was included during the implementation. Timestamps were removed to preserve space.

```
server.py:497 | [SUMMARY]
server.py:498 | Run finished 10 round(s) in 7927.92s
```

```
server.py:500 |  History (loss, distributed):
server.py:500 |   round 1: 3.9957508273308635
server.py:500 |   round 2: 3.7077665836391236
server.py:500 |   round 3: 3.587809463973882
server.py:500 |   round 4: 3.392217770903355
server.py:500 |   round 5: 3.2988361252210727
server.py:500 |   round 6: 3.2434051224810494
server.py:500 |   round 7: 3.1531953296160107
server.py:500 |   round 8: 3.1619216581220764
server.py:500 |   round 9: 3.1211294103918426
server.py:500 |   round 10: 3.115858381467096
server.py:500 |  History (metrics, distributed, fit):
server.py:500 |  {'avg_train_time': [(1, 141.45023037989935),
server.py:500 |                      (2, 135.019606590271),
server.py:500 |                      (3, 145.94761505723),
server.py:500 |                      (4, 138.38720326622328),
server.py:500 |                      (5, 143.75289497772852),
server.py:500 |                      (6, 140.310248285532),
server.py:500 |                      (7, 140.89176101485887),
server.py:500 |                      (8, 135.53835080067316),
server.py:500 |                      (9, 131.24846536914507),
server.py:500 |                      (10, 145.32233116030693)],
server.py:500 |   'train_loss': [(1, 5.289074437387782),
server.py:500 |                  (2, 4.34188064696114),
server.py:500 |                  (3, 3.8342456738823936),
server.py:500 |                  (4, 3.554741788036226),
server.py:500 |                  (5, 3.3563418201074353),
server.py:500 |                  (6, 3.1757741227706466),
server.py:500 |                  (7, 3.0587893820260943),
server.py:500 |                  (8, 2.881507779080469),
server.py:500 |                  (9, 2.9725834435179994),
server.py:500 |                  (10, 2.80687957709052)]}
server.py:500 |  History (metrics, distributed, evaluate):
server.py:500 |  {'accuracy': [(1, 0.23380447585394581),
server.py:500 |                (2, 0.2629564193168433),
server.py:500 |                (3, 0.2782685512367491),
server.py:500 |                (4, 0.3136042402826855),
server.py:500 |                (5, 0.33451118963486454),
server.py:500 |                (6, 0.3421672555948174),
server.py:500 |                (7, 0.36071849234393405),
server.py:500 |                (8, 0.37338044758539457),
server.py:500 |                (9, 0.37338044758539457),
server.py:500 |                (10, 0.3839811542991755)]}
```

Index