This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement N° 101070046

D3.1 Federated Learning infrastructure implementation

Deliverable No.	D3.1	**Due Date**	31/08/2023
Description	This report provides an analysis of the state of the art in Federated Learning and the privacy-preserving techniques that can be applied to this Machine Learning approach. It also presents the framework to be used in the Alchimia project.		
Type	OTHER	**Dissemination Level**	PU
Work Package No.	WP3	**Work Package Title**	Federated Learning and Continual Learning
Version	1.0	**Status**	Final

Authors

Name and surname	Partner name	e-mail
Jorge Mira Prats	ATOS	jorge.mira@atos.net
Julia Ruiz Cardon	ATOS	julia.ruiz@atos.net

History

Date	Version	Change
07/07/2023	0.0	Initial ToC
10/07/2023	0.1	Final ToC
11/08/2023	0.2	Final version for reviewing
29/08/2023	0.3	SSSA review
31/08/2023	1.0	Final version

Key data

Lead Editor	Julia Ruiz Cardon (ATOS)
Internal Reviewer(s)	Stefano Dettori (SSSA), Marco Vannucci (SSSA)

Abstract

This report consists of the Deliverable "D3.1: Federated Learning framework" of the European HORIZON-CL4-2021-DIGITAL-EMERGING-01 "Alchimia". The scope of this deliverable is to analyse the Federated Learning possibilities and present the framework used in the project.

D3.1 includes an extensive analysis of the state of the art focusing on the type of Federated Learning system, its challenges, aggregation algorithms and open-source solutions available. In addition, the privacy-preserving concept and the main techniques that can be applied in a Federated Learning system are provided together with a comparison of the open-source tools. Later, the Federated Learning framework selected for the support of the ML model training is presented. The framework has been developed with the purpose of covering the needs of the existing architecture for the Alchimia system up to the present time. Concludingly, an example of a Deep Learning model, which was trained to utilize the Federated Learning framework, is described, accompanied by a presentation of the outcomes achieved.

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both.

Table of contents

List of tables

List of figures

1 Introduction

In the pursuit of a sustainable environment, the convergence of Artificial Intelligence (AI) and Industry 4.0 has emerged to drive efficiency, resource optimization and environmental preservation. This transformative combination allows the exploration of innovative solutions to address pressing ecological challenges while propelling society towards a greener future and improving processes' efficiency. The key elements introduced by Industry 4.0, including seamless connectivity, enhanced interoperability, and intelligent automation, have paved the way for cloud computing, big data analytics, and increased processing power to facilitate the integration of AI on an unprecedented scale. These advancements have unlocked a new frontier, enabling AI adoption in practically all sectors, including metallurgical. However, the expansion of AI has grown the concerns surrounding data privacy. As AI technologies rely on vast amounts of data for training, the collection and storage of sensitive information raise security issues.

Conventional AI-driven solutions need to have the whole data stored in a central server so that the model can be trained, this supposes the centralization of all data, which can lead to privacy problems, overhead in data transfer, and assumptions of homogeneity in the data, which may not occur when data is captured in different places.

Federated Learning (FL), introduced by Google [1], solves these concerns. By adopting a decentralized training approach, FL allows AI model training to occur locally at the source where data is captured, eliminating the need for centralization of the entire dataset. This decentralized training paradigm ensures that raw data is not shared, thereby preserving data privacy, and protecting sensitive information. As a result, FL overcomes the privacy problem inherent in traditional AI systems while maximizing the utility of distributed data sources.

Furthermore, FL operates on a collaborative model, enabling multiple entities, such as edge devices or individual facilities, to collaboratively train a shared global model. It also reduces data transfer overhead since only local model weights are shared with the central server. Then, the central server aggregates all local weights creating a global model which is again shared with the trainers. This way, the homogeneity assumption is avoided since local models are trained on local data which can present slight distribution differences.

However, model sharing can lead to some security inconveniences such as model inversion attacks that consists of extracting information from aggregated model which could reveal sensitive information about individual data. In addition, ensuring the confidentiality and integrity of model weights during transmission is essential to prevent "man-in-the-middle" attacks, model stealing or model poisoning. To address these security risks, techniques such as differential privacy, secure aggregation protocols and encryption can enhance the privacy and security of the FL system.

Considering the above, Alchimia provides a FL framework with enhanced privacy-preserving features for the FL system.

This document consists of a technical report that includes the state of the art regarding Federated Learning together with privacy-preserving techniques. It also describes the tools and methods that will be developed during the Alchimia project.

1.1 Intended Audience

This document focuses on the technical aspects of FL and performs an analysis of open-source tools available. Furthermore, a new approach to implementing a FL framework is described. Therefore, the intended audience for this document includes ML engineers, AI engineers and data scientists who are interested in deploying an FL system since they can benefit from the study performed and the new FL framework implementation approach.

Information about the FL paradigm can be consulted in this document, so it can be useful internally, for all project participants.

1.2 Document Overview

The rest deliverable is divided into the following sections:

- Section 2 carries out an analysis of the state of the art of the FL paradigm. It provides research on challenges present in this approach, and the different solutions in terms of system architecture and FL learning aggregations. Finally, a description of the main open-source FL frameworks is given.
- Section 3 provides a description of the main methods for Privacy-Preserving together with a comparison of the different FL frameworks presented in Section 2 considering the functionalities they present from a Privacy-Preserving perspective.
- Section 4 presents the FL framework proposal for this project. It explains the philosophy behind the framework and describes the core components of the implementation.
- Section 6 provides a prototype example applying FL in a scenario with 2 participants and one central server for training a Deep Learning (DL) model.
- Section 6 presents the conclusions reached in this document together with next steps.

2 Federated Learning

Federated Learning, an innovative approach to decentralized machine learning, has gained considerable attraction in recent years as a transformative solution to privacy and data security concerns in AI-driven applications. FL allows multiple entities, such as organizations with distributed facilities or individual devices, to collaboratively train a shared global model while keeping their raw data localized and secure. This approach effectively avoids the need to collect a vast amount of data from different devices located in distinct places and transfer the data to a central server for training. By decentralizing the training process, Federated Learning not only preserves data privacy but also minimizes data transfer overhead, making it an efficient and resource-saving alternative. Furthermore, the collaborative nature of FL empowers organizations to leverage the collective intelligence of diverse sources without compromising data security.

The FL training process typically begins with a global model being initialized on a central server or aggregator. In this setting, each participating entity, e.g., a facility or a device, also known as *the client*, performs local model training using its own data. Instead of sharing raw data, only model updates or gradients are transmitted to the central server. Finally, in each iteration, the server aggregates the individual client models to obtain a new server model and sends the current version of the machine learning model to the clients.

This collaborative, privacy-preserving approach empowers organizations to leverage the collective intelligence of diverse sources without compromising data security.

FL approaches present high versatility and can be adopted in diverse use cases and scenarios, including the manufacturing or metallurgical sector. In manufacturing, factories and industrial facilities can leverage the power of FL to optimize production processes, enhance quality control, and predict equipment failures by collaboratively training AI models using data from different production lines or machines. By training local models locally, FL enables manufacturers to maintain data privacy and security while still benefiting from the collective knowledge of the entire ecosystem.

Despite its numerous benefits, FL also presents challenges that must be addressed to ensure the successful implementation and adoption of this paradigm.

2.1 Challenges

This section describes key challenges faced in the implementation and deployment of FL systems.

Table 1 : Key challenges in Federated Learning

Challenge	Category	Description
Data Poisoning	Privacy/Security	Malicious participants may inject poisoned data or modify local model updates to compromise the global model integrity. [2]
Model Inversion	Privacy/Security	Model updates could potentially reveal sensitive information about individual data points, leading to privacy breaches.

Challenge	Category	Description
Membership Inference	Privacy/Security	Analysing model updates exchanged during FL can enable adversaries to infer whether specific data points were part of participants' training datasets, compromising data privacy. [3]
Sybil Attacks	Privacy/Security	Malicious participants may create multiple identities to gain disproportionate influence over the model aggregation process, distorting the global model. [4]
Data Heterogeneity	Model Convergence	Coordinating the training process across entities with varying data distributions and computational capabilities may hinder model convergence. [5]
Training Process Coordination	Communication	Some participants may have intermittent connectivity or varying update schedules, leading to asynchronous training.
Communication Overhead	Communication	Frequent model updates and aggregations can lead to increased communication overhead, impacting the efficiency of FL in resource-constrained environments. [6]

2.2 Federated Learning Systems

FL systems can be categorized based on the structure and organization of the participants and the data distribution. The following subsections describe the main FL systems.

2.2.1 Centralized

A centralized FL system is the most widely used architecture. All the participants send their local model updates directly to a central server, also known as the aggregator. The central server performs the model aggregation and updates the global model, which is then redistributed back to all participants for the next round of training.

This schema allows adopting simplified communication since all communication between server and clients goes through a single point, making communication management and coordination straightforward. Furthermore, as the server has access to all participant updates, it can compute the aggregated model efficiently.

However, the simplicity of this architecture presents some drawbacks. Whether the central server experiences an issue or goes down the entire FL process may be affected. Communication also has its limitations. All participants must communicate with the server which can create a communication bottleneck. Finally, as the number of participants increases, the FL system may face scalability problems.

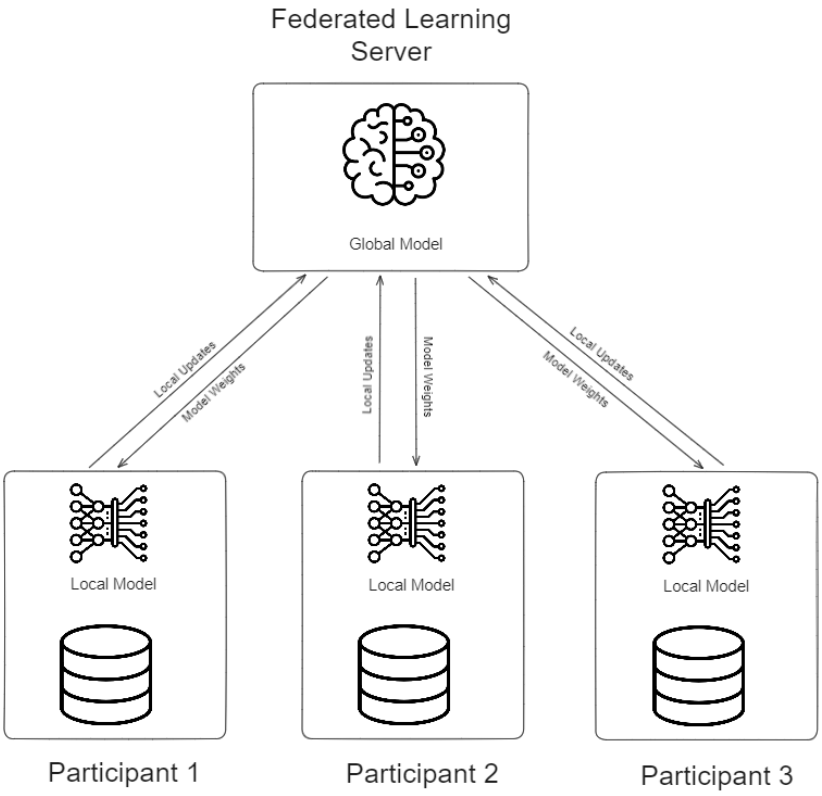

Figure 1. Centralized FL architecture

2.2.2 Decentralized

In a decentralized FL architecture, participants communicate and exchange model updates directly with each other without the need for a central server. Clients form peer-to-peer networks, and model aggregation is achieved through consensus algorithms or federated averaging techniques.

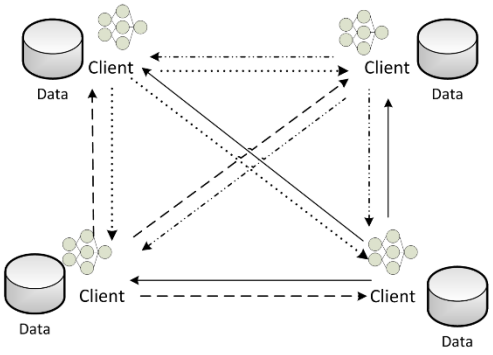

Figure 2. Decentralized FL architecture [7].

This architecture enhances privacy since participants do not have to share individual data with a server. It also increases robustness and resilience to single points of failure, as there is no central entity that can disrupt the entire system. If one of the participants fails, it does not affect the whole FL procedure. Moreover, this approach is scalable by default since it

does not need big changes when the number of participants increases, and the communication bottleneck is removed.

Despite the advantages presented by decentralized architecture, there are some challenges when adopting this type of scheme. Due to the absence of a central server to manage the training process, it becomes impossible to control it and the possibility of suffering client drift [1], imbalance data and synchronization issues when providing model updates exist.

2.2.3 Hierarchical

Hierarchical FL consists of a multi-level architecture where participants are organized into hierarchical groups. Within each group, local model training and aggregation are performed as in traditional FL. Nevertheless, instead of sending individual model updates to a central server, groups of participants send the updates to local aggregators which are responsible for aggregating the group model updates. The local aggregations are then redistributed to a global aggregator which computes the unique global model.

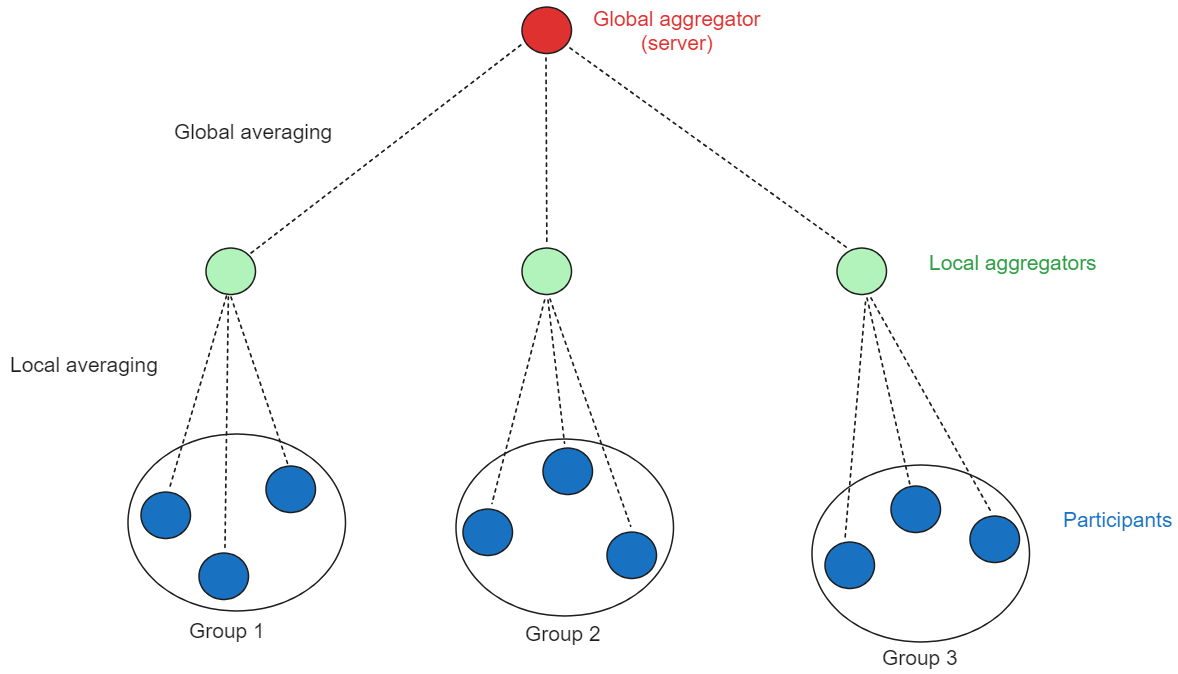

Figure 3. Hierarchical FL overview.

The main advantage of the hierarchical approach is the scalability. Clients can be organized into different groups reducing communication overhead and aggregation

[1] Client drift is defined as the situation when due to data heterogeneity from different FL participants appears a drift between local optimization and global optimization. [34]

complexity compared to centralized architecture. It also provides flexibility in terms of adding or removing nodes easily which makes the system efficient.

The disadvantages of such a system are the communication redundancy and privacy overhead. Model updates must be calculated and communicated multiple times as they are moved to the top of the hierarchy resulting in communication redundancy. Also, privacy mechanisms are intensive tasks in terms of resources and are required every time the model updates are moved between aggregators. Therefore, the privacy protocols must be repeated several times in each round. Finally, synchronization across different levels of hierarchy may increase the complexity.

2.2.4 Vertical and Horizontal FL

FL systems can be categorized based on data distribution scenarios. Vertical and Horizontal FL refers to how data is partitioned across different participants.

Generally, it is assumed that data from all participants present different numbers of samples with the same features which is the case of Horizontal FL, as shown in Figure 4.

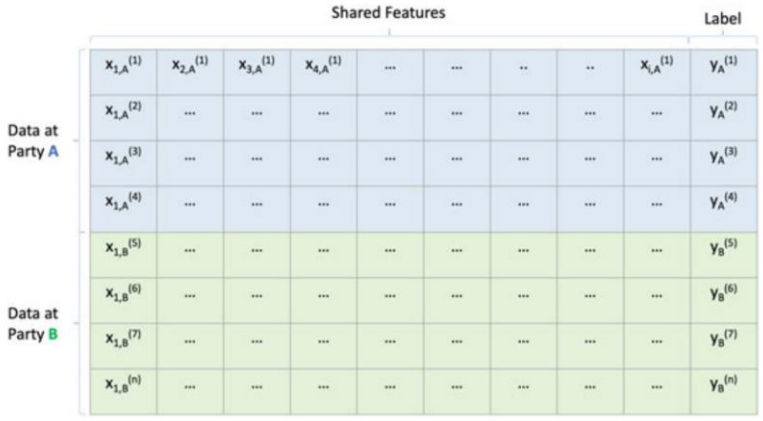

Figure 4. Horizontal FL example [8].

However, in some cases, participants may have different features referring to the same entity. FL systems that can afford these scenarios are known as vertical FL systems. Below is shown an example of this partition.

Figure 5. Vertical FL example [8].

2.2.5 Cross-Silo and Cross-Device FL

Categorization of FL systems can be done by analysing the structure of data. Thus, if data is distributed across multiple devices or data silos.

In Cross-silo FL, multiple autonomous organizations or data silos collaborate to compute a global model without sharing the raw data. Each organization keeps control of its data, ensuring privacy and security. On the other hand, Cross-device FL enables the computation of a global model from multiple devices owned by a single organization. Each device becomes a participant in the FL process.

2.3 Aggregation Algorithms

As described in previous sections, the central server plays a critical role in the FL system. It is responsible for aggregating the local model updates provided by all participating devices or facilities. The process of aggregating these updates is crucial in determining the quality and performance of the resulting global model. There are different aggregation approaches available in the literature. Below are described the main strategies.

2.3.1 Main Aggregation Strategies

Among all aggregation strategies available in the literature, Federated Averaging (FedAvg), Federated Weighted Averaging (FedWeightedAvg) and Federated Stochastic Gradient Descent (FedSGD) are the most widely used algorithms.

The idea of the FedAvg algorithm [1] consists of training a shared global model by averaging the model updates contributed by participants. The algorithm can be formulated as:

$$w_{t+1} = \frac{1}{N} \sum_{i=1:N} w_{t+1}^{i}$$

Where:

- N is the total number of participants in the FL system.
- w_{t+1}^{i} represents the local model weights of the participant *i* at time step *t+1*.

- w_{t+1} consists of global model update at time step $t+1$.

The central server aggregates local model updates using simple averaging resulting in a global model. By averaging the model updates, FedAvg ensures each participant's contribution to influence the global model. As more rounds of training are performed, the global model becomes more accurate and representative of the entire dataset.

Despite FedAvg efficiency, this strategy presents some limitations in specific scenarios:

- Data Heterogeneity. FedAvg assumes all participants have similar data distributions and sizes. However, data collected by different participants could be diverse due to variations in participant characteristics. FedAvg may lead to a suboptimal global model in this type of situation.
- Fairness. FedAvg treats all local model contributions equally during the aggregation without considering the data quality or the importance of the FL system. This procedure may result in unequal representation of participants.
- Communication Overhead. Although communication overhead is reduced compared to transmitting raw data to the central server, FedAvg involves exchanging model updates between participants and the central server in each training round. This communication can become a bottleneck, affecting the efficiency and scalability of the system.
- Convergence Rate. The averaging approach may be slower than traditional stochastic gradient descent used in centralized training in terms of convergence rate.

FedWeightedAvg is a variation of FedAvg designed to overcome data heterogeneity and fairness issues. This variation assigns individual weights to each participant during the aggregation of model updates. These weights are commonly based on various factors, e.g., data volume, data quality or device capabilities.

By introducing individual weighting, FedWeightedAvg aims to provide more importance to those participants that present higher data quality or better device capabilities, allowing the local model updates to have higher relevance when computing the global model.

Finally, the FedSGD strategy emerged aiming to apply distributed SGD [9] in a federated approach[2] and can be considered an extension of FedAvg since instead of averaging the full model updates from participants, the central server aggregates only the gradients computed on the local datasets. This variation stands out FedAvg in communication overhead and convergence rate. By aggregating only the gradients computed by the participants, the communication overhead is reduced since gradients are typically smaller (in terms of storage bytes) than the entire model parameters. This is due to different

[2] Distributed vs Federated approach: Main difference is that in distributed training a large, centralized dataset is divided into smaller subsets and distributed across multiple computed nodes while in federated training data remain decentralized and kept locally on participants.

reasons such as the sparse nature of gradients, meaning that many of their components are close to zero and can be further compressed or optimized for storage. Also, FedSGD can achieve faster convergence due to it avoids the averaging approach.

2.3.2 Alternative Strategies

This section briefly describes some of the alternative aggregation strategies present in the literature.

Table 2: Alternative aggregation strategies

Aggregation Strategy	Description
FedPer [10]	This approach proposes to incorporate personalization layers into the model. Thus, the DL model is divided into two parts. The base layer and the personalization layer. Base layers are trained through FedAvg or similar variation while personalization layers are only trained from local data. The objective of this approach is to mitigate the effects of data heterogeneity.
FedMeta [11]	FedMeta aims to handle statistical and systematic challenges in collaborative training ML models in a federated scenario. This method proposes to share a parameterized algorithm (meta learner) instead of a global model. The key idea of this approach is that by sharing meta-information, the global model can be shaped in a way that facilitates faster adaptation.
FedProx [12]	FedProx can be seen as a variation of FedAvg which addresses heterogeneity issues in federated networks by adding a proximal term on each aiming to improve the stability of the method.
FedPAQ [13]	This method relies on three key features: i) periodic averaging where models are updated locally and only periodically averaged at the central server; ii) partial participation where only a fraction of participants train in each round; iii) quantized message-passing of participant updates before uploading them on the central server. This approach reduces the communication overhead issue present in FL scenarios.

2.4 Federated Learning Frameworks

There are several open-source FL frameworks providing tools to implement distributed ML models in different scenarios.

One such framework is Tensorflow Federated (TFF) [14], developed by Google, which offers a flexible and scalable approach, allowing users to build and simulate FL algorithms in Python. Another popular framework is PySyft [15], developed by OpenMined, which focuses on privacy-preserving ML. Federated AI Technology Enabler (FATE) [16] is another open-source framework tailored for industrial applications. It implements secure computation protocols based on homomorphic encryption and multi-party computation.

Flower [17] is one of the most widely used open-source FL frameworks. It is an ML framework agnostic, flexible and easy to use and allows for adding privacy mechanisms. An additional FL open-source framework is PaddleFL [18]. It allows deploying FL systems in large-scale distributed clusters. FedML [19] provides an ML platform that enables secure FL and analytics, among others. The final framework analysed in this deliverable is LEAF [20], a modular benchmarking framework for learning in federated settings.

The following sections provide a brief description of these frameworks with their main features.

2.4.1 TensorFlow Federated (TFF)

TFF[3] aims to enable FL research and experimentation. It allows us to simulate algorithms and test new ones. The interfaces are organized into two main layers: *Federated Learning API* and *Federated Core API*. Federated Learning API offers a set of high-level interfaces to use algorithms included in the framework, while Federated Core API provides low-level interfaces for implementing algorithms by combining Tensorflow with distributed communication.

Its main features are:

- Good integration with Tensorflow
- Provides base classes for FedAVG and FedSDG.
- Allows applying Differential Privacy.

However, TFF presents some limitations: i) does not support vertical data splitting and ii) the FL mode is not implemented.

The architecture of TFF is shown in Figure 6. It presents the strategy layer on the top where the FL algorithms are implemented. Then, the security layer enables the application of the Differential Privacy mechanism. Finally, the model layer presents all methods for creating DL models.

[3] https://github.com/tensorflow/federated

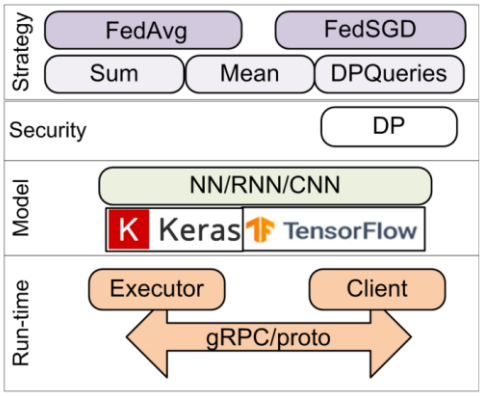

Figure 6. TFF architecture overview [21].

2.4.2 PySyft

PySyft[4], developed by OpenMined, is an FL tool focused on secure DL enabling to apply different mechanisms such as Differential Privacy or Secure multi-party.

This framework allows training FL models developed using both Tensorflow and PyTorch. Also, it offers the possibility to split the data vertically and horizontally. Finally, OpenMined presents an ecosystem that can be used together with PySyft to add functionalities and overcome some limitations. PySyft models can be used on Android and iOS by using KotlinSyft and SwiftSyft projects, respectively. Although it only supports simulation mode, developers can integrate other projects such as PyGrid to support FL mode.

The architecture of PySyft is shown below. The strategy layer allows training models and supports both vertical and horizontal partitions. The security layer enables to apply several security mechanisms. Finally, run-time is compatible with Tensorflow and PyTorch [22] frameworks.

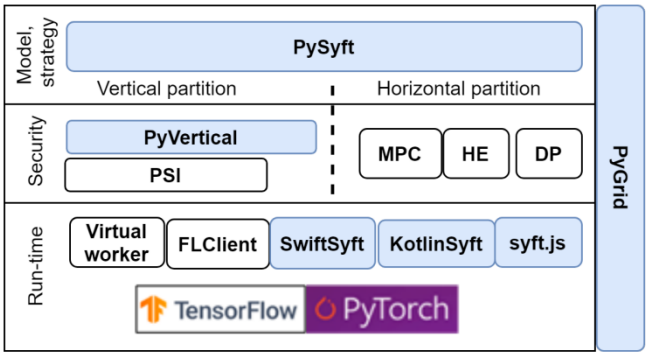

Figure 7. PySyft architecture overview [15].

[4] https://github.com/OpenMined/PySyft

2.4.3 FATE

FATE[5] is an open-source project initiated by WeBank's AI Department to provide a secure computational framework supporting a federated AI ecosystem [16]. It is designed for industrial applications.

It is compatible with main DL frameworks (Tensorflow and PyTorch) and supports horizontal and vertical FL approaches. Regarding security, FATE supports Secure Multi-party Computation (SMPC) protocols and encryption methods (explained in section 3). Furthermore, FATE can be deployed in simulation and federated modes.

One of the most important inconveniences that FATE presents is that developers must modify the source code of FATE to implement custom FL algorithms which adds complexity to adapt FATE to all possible scenarios.

Figure 8 shows the FATE architecture overview. The main components are:

- FATE-Cloud includes a cloud manager responsible for managing the complete Cloud deployment.
- FATE-Board: Visualization tool focused on FL tasks and models.
- FATE-Flow: Task scheduler for managing FL lifecycle.
- FATE-Serving: Scalable online FL model serving.

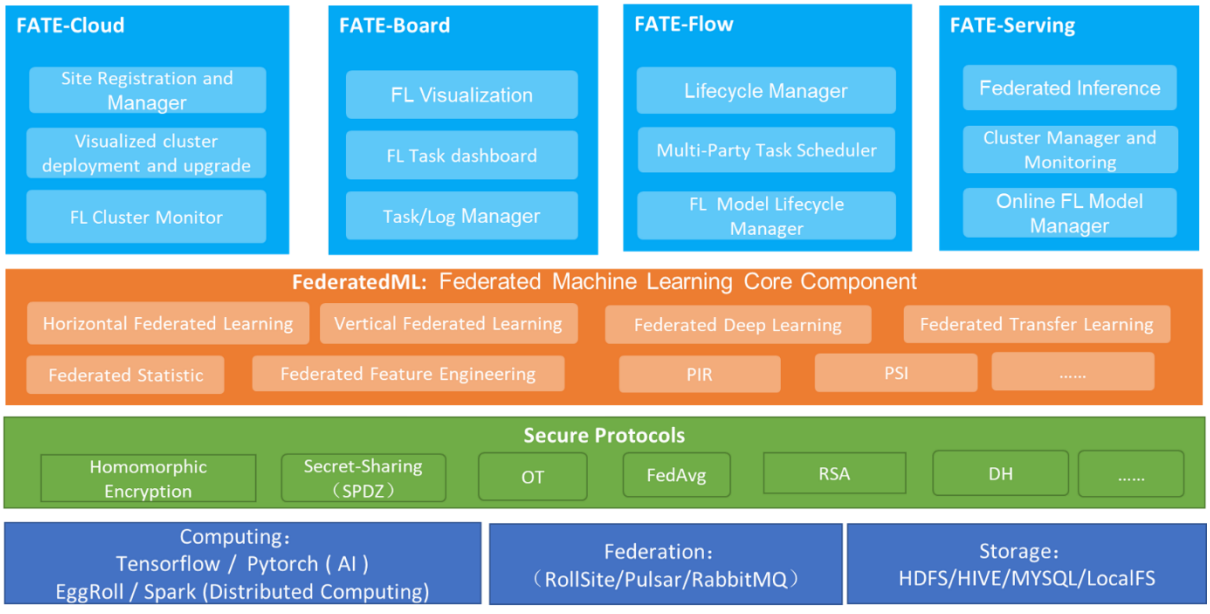

Figure 8. FATE architecture overview [16].

[5] https://github.com/FederatedAI/FATE

2.4.4 Flower

Flower[6] is an open-source framework for FL that simplifies the development and deployment of FL systems. It provides a high-level API for building FL algorithms and systems.

Among key features that stand out:

- Abstraction of FL components.
- ML framework agnostic.
- Support of heterogeneous devices.
- Scalability.
- Flexibility for customization and research.

The flower library facilitates the implementation of various FL algorithms, including FedAVG, FedSGD, and others. It abstracts the communication and coordination aspects of FL, allowing developers to focus on algorithm design and experimentation.

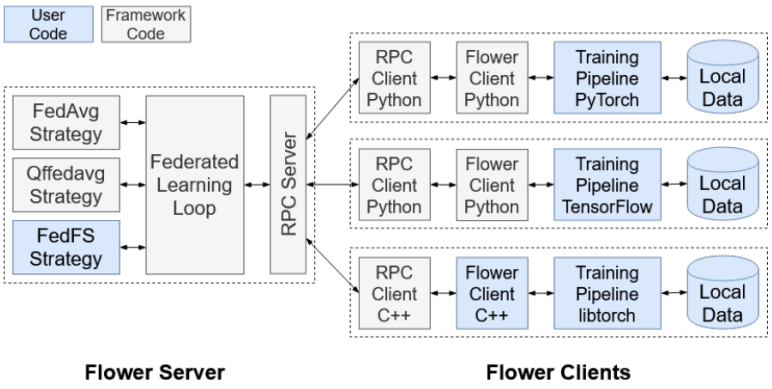

Figure 9. Flower framework architecture [23].

2.4.5 PaddleFL

PaddleFL[7] uses Paddle (PArallel Distributed Deep LEarning) [24], a DL platform that covers DL frameworks, basic model libraries, end-to-end development kits, tools, and components. PaddleFL can process data vertically and horizontally. Algorithm implementation for each approach is decoupled and the algorithms available for each approach are not the same. PaddleFL can be deployed in large-scale clusters. The protocol used for communicating with servers and workers is ZeroMQ [25], a high-performance, asynchronous messaging library that provides sockets for applications or processes.

[6] https://github.com/adap/flower

[7] https://github.com/PaddlePaddle/PaddleFL

The main drawback of PaddleFL is that it uses a little-known DL platform with low support from the community. The PaddleFL architecture overview is depicted in Figure 10.

.

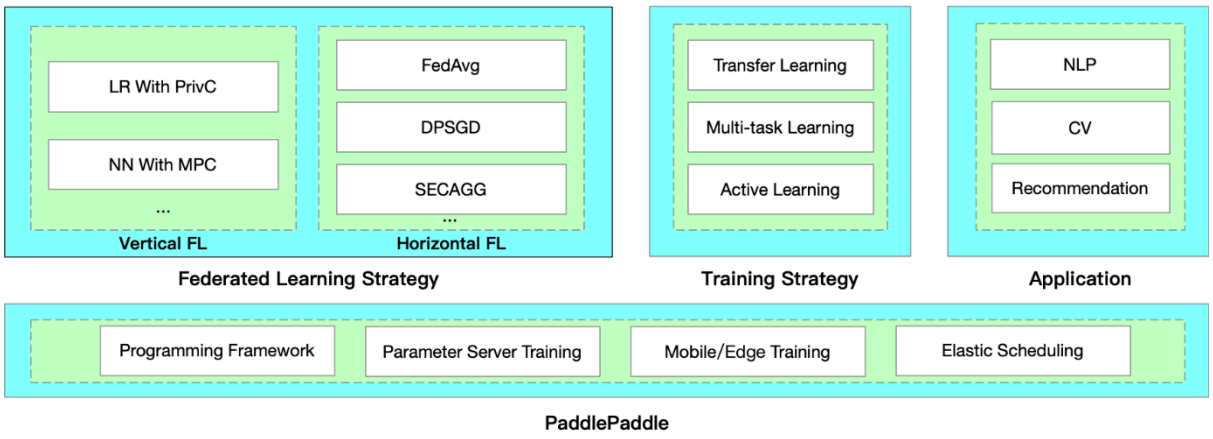

Figure 10. PaddleFL architecture overview [21].

2.4.6 FedML

FedML[8] [26] provides an end-to-end toolkit to enable the development of FL algorithms. The key features of FedML are:

- Support of diverse FL computing paradigms.
- Support of diverse FL configurations.
- Standardized FL algorithm implementations.
- Standardized FL benchmarks.
- Fully open and evolving.

The FedML library organization is depicted in Figure 11 and presents two key components: FedML-API and FedML-core.

FedML-core decouples distributed communication from model training. The distributed communication module is responsible for low-level communication. The backend is based on MPI (Message Passing Interface)[9]. Furthermore, this module offers security functionalities. The model training module is built on PyTorch, so developers can create the trainers according to their needs.

[8] https://github.com/FedML-AI/FedML

[9] https://pypi.org/project/mpi4py/

Fed-API offers several FL strategies, and, with the support of FedML-core, new algorithms can be also implemented. In addition, FedML-API offers an ML system practice that consists of separating the development of models and data from algorithms.

FedML also provides other functionalities for supporting common FL scenarios such as FedML-Mobile and FedML-IoT which are focused on mobile and IoT applications, respectively.

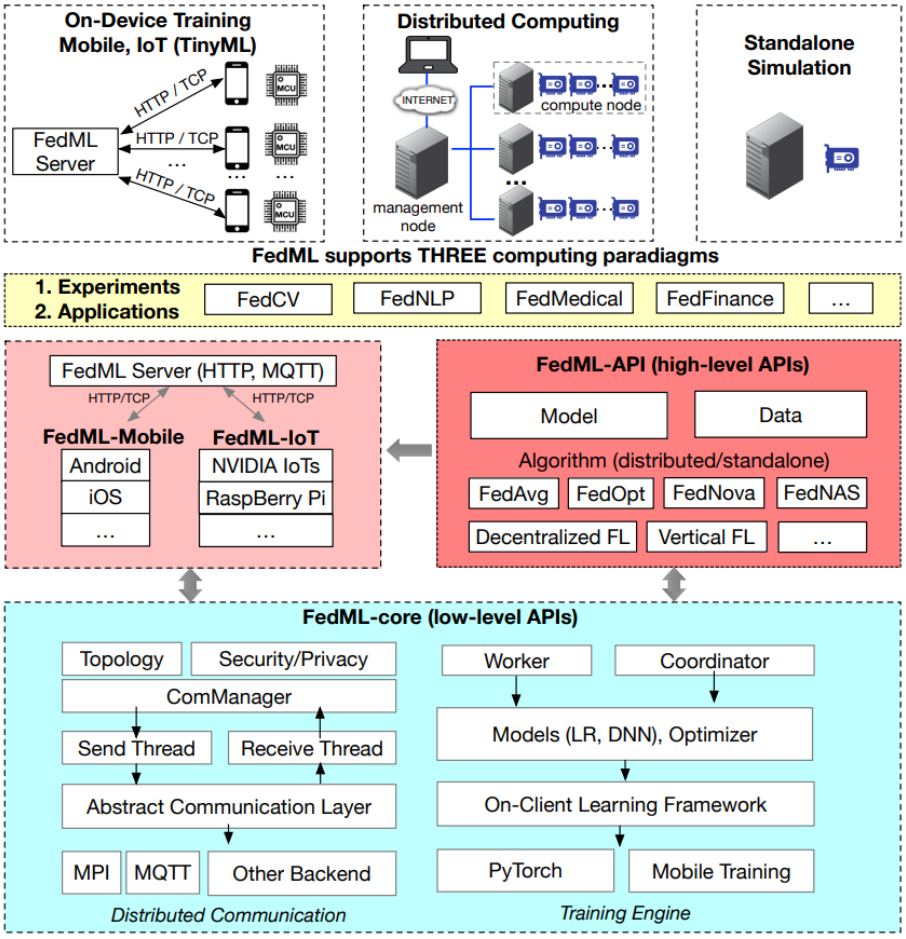

Figure 11. Overview of FedML architecture [26].

2.4.7 LEAF

LEAF[10] is an open-source modular benchmarking FL framework tool. The main features provided by this framework are:

- It enables reproducible science.

[10] https://github.com/TalwalkarLab/leaf

- It provides granular metrics.
- It is modular.
- Creation and curation of federated datasets.

The main drawback of this framework is the lack of documentation and low support from the community.

3 Privacy-Preserving Federated Learning

Federated Learning has emerged as a promising approach to enable collaborative learning across distributed devices without revealing the underlying training data. However, simply retaining data and computations on individual devices, as done in FL, does not fully guarantee privacy. The exchange of model parameters among participants still poses a risk, as these parameters may accidentally contain sensitive information, making the system vulnerable to privacy attacks. For example, in [27] the authors indicate the potential risk of sensitive text patterns, such as credit card numbers, being extracted from a recurrent neural network trained on users' data. Therefore, additional mechanisms are necessary to protect data disclosure from attack strategies, which are subject to privacy-preserving methods in FL.

Generally, privacy preservation techniques are focused on two main objectives: privacy of the training dataset and privacy of the local model parameters which are exchanged with other nodes and/or a centralised server. The primary approaches that can be employed in FL for privacy protection are Differential Privacy, Homomorphic Encryption, Secure Multiparty Computation, and Private Aggregation of Teacher Ensembles (PATE). These approaches are described in the following sections.

3.1 Differential Privacy

Differential Privacy (DP) [28] is a mechanism for privacy preservation that injects random noise into true outputs using rigorous mathematical measures. By using this technique, data patterns corresponding to the model, parameters, or training data become impossible to disclose. However, it is important to consider the trade-off between privacy guarantee and efficiency since adding too much noise and improper randomness can significantly result in lower performance of the model.

Generally, differential privacy can be divided into Local Differential Privacy (LDP) and Global Differential Privacy (GDP).

Local Differential Privacy (LDP) adds random noise to each client's data before transmitting it to a central server or aggregator. The addition of noise happens directly on the individual user or at the data source itself, independently of other users. The central server then aggregates the perturbed data and computes statistical analysis while providing privacy guarantees.

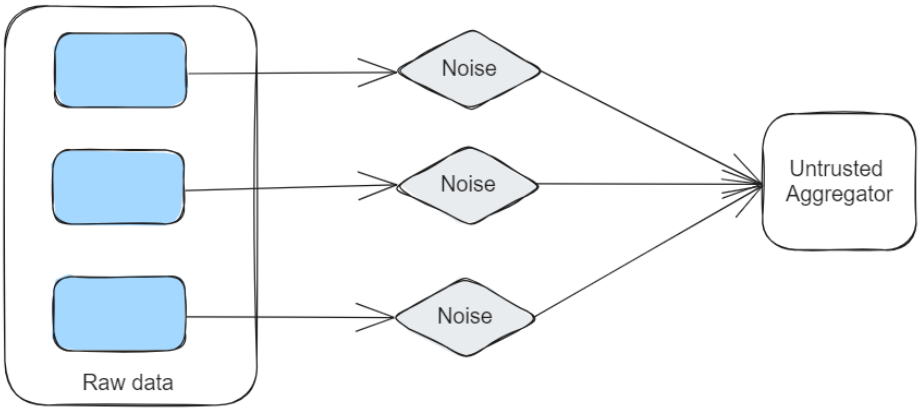

Figure 12: Local Differential Privacy diagram

Global Differential Privacy (GDP) operates by adding random noise to the aggregated results after data is collected and processed at the central server or aggregator. Instead of perturbing individual data points, each user sends their data to the aggregator node without adding noise and the aggregator transforms it. Global private systems tend to be more accurate since all the analysis is implemented on the clean (noise-free) data, and only at the end of the process the noise is added.

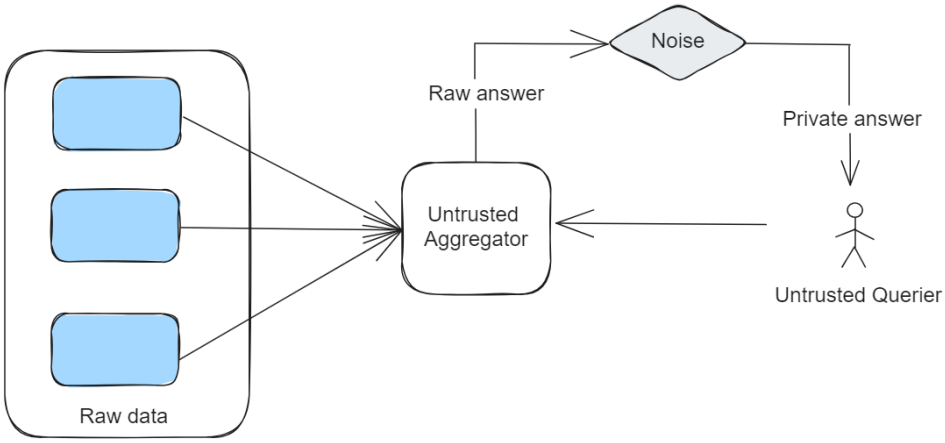

Figure 13: Global Differential Privacy diagram

3.2 Homomorphic Encryption

Homomorphic encryption is another approach for privacy preservation in Federated Learning. This technique allows to perform computation on an encrypted form of data without the need for a secret key to decrypt the data [29]. The operation is equivalent to implementing the computation on the original unencrypted data. In simpler terms, it enables data to remain encrypted while still being usable for certain computations.

The term "homomorphic" derives from algebraic structures and refers to the property of preserving mathematical operations on encrypted data. There are three main types of homomorphic encryption techniques:

Partially Homomorphic Encryption (PHE) allows only specific types of mathematical operations (e.g., addition or multiplication) to be performed on encrypted data. This limits the scope of computations that can be done.

Somewhat Homomorphic Encryption (SHE) can be used with a larger set of mathematical operations on encrypted data. Although, it has practical limitations on the number of operations or their complexity. While it may not support all computations, it is more flexible than PHE.

Fully Homomorphic Encryption (FHE) [30] enables to perform of any function on encrypted data that could be computed on the plaintext data directly. This level of flexibility is incredibly valuable for privacy preservation.

3.3 Secure Multiparty Computation

Secure Multi-party Computation (SMPC) [31] motivation is to compute a function over a dataset owned by multiple parties using their own inputs so that any party learns nothing about others' data. If there are n parties P_1, P_2,..., P_n that own n pieces of private data X_1, X_2,..., X_n, any arbitrary function $f(X_1, X_2,..., X_n) = (Y_1, Y_2,..., Y_n)$ can be applied so that the only information each party has access is the result $(Y_1, Y_2,..., Y_n)$ and its own inputs X_i. SMC often utilizes Zero-Knowledge Proofs (ZKPs) to demonstrate computation integrity and prove that any specific information is revealed. These proofs allow a party to verify that they have followed the rules of the protocol and provided the correct input without disclosing the input itself. Sometimes this property can involve complex operations that are not very efficient and lead to increased computational overhead.

SMC is an active area of research, and ongoing developments are improving its efficiency and applicability.

3.4 Private Aggregation of Teacher Ensembles (PATE)

A common approach for privacy preservation is the Private Aggregation of Teacher Ensembles (PATE) [32] which introduces the teachers-student scheme. This method combines multiple learning models used as teachers for a student model.

Each **teacher** trains a model with local/private data. The teachers might have different privacy-preserving mechanisms. The predictions made by the teacher models are aggregated adding some noise. The noise introduced during aggregation helps protect individual data privacy. The aggregated predictions are then used as a labelled dataset to train a **student**.

The student learns to predict an output chosen by noisy voting (from the aggregation) among the teachers and cannot directly access an individual teacher or the underlying data or parameters. In Figure 14 the architecture of PATE is displayed.

Initially, multiple teacher models are trained on separate and non-overlapping subsets of the sensitive data. Following this, a student model is trained using auxiliary, unlabelled non-sensitive data. The training process involves using the aggregate output from the ensemble of teacher models, allowing the student to learn and mimic the ensemble's behaviour accurately.

The rationale behind this approach is to ensure that the student model does not rely on specific details of any individual sensitive data point (such as any user's information). As a result, the privacy of the training data is safeguarded, even if potential attackers can observe the internal model parameters of the student.

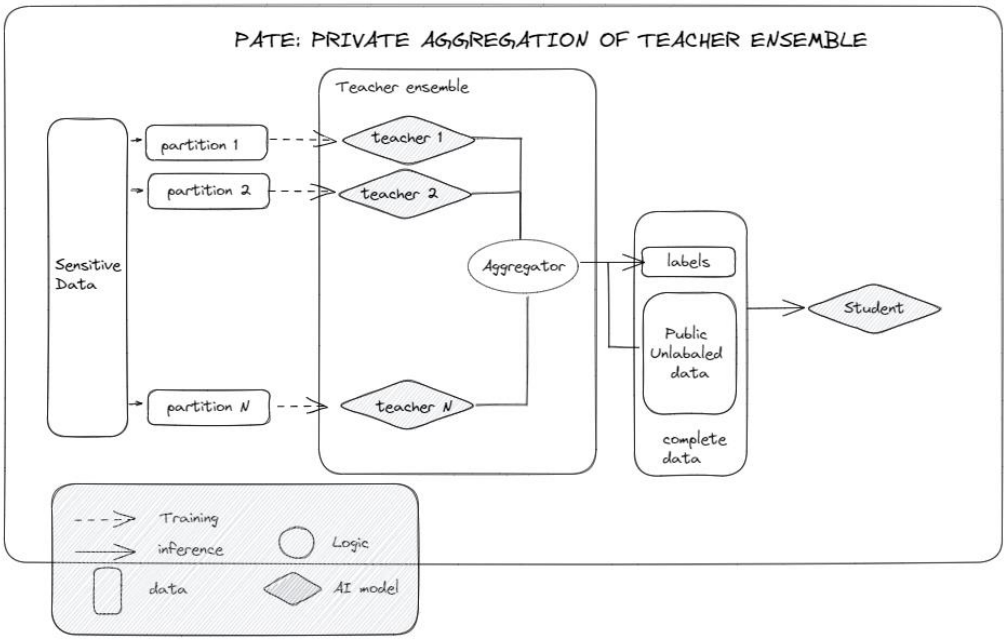

Figure 14: Private Aggregation of Teacher Ensemble diagram

3.5 Comparison of Federated Learning frameworks considering privacy preservation

Sections 2 and 3 provided an initial description and analysis of the Federated Learning paradigm, the main frameworks and privacy mechanisms employed. In Table 3: Main Pros and Cons of the Federated Learning frameworks a comparison of the Federated Learning frameworks is conducted, and their pros and cons are presented.

Table 3: Main Pros and Cons of the Federated Learning frameworks

Framework	Pros	Cons
FATE	1. Production Ready. 2. Provides many FL algorithms. 3. Containerized support with Docker/Kubernetes.	1. No differential privacy algorithms. 2. Developers must modify the source code of FATE to implement custom FL algorithms. 3. Does not use GPUs for training.
Flower	1. ML Framework agnostic.	1. No differential privacy algorithms.

Framework	Pros	Cons
	2. Easy to transform ML pipelines to FL. 3. Supports a great number of clients. 4. Very customizable.	
PaddleFL	1. Multiple privacy-preserving algorithms such as DP, MPC and secure aggregation. 2. High-level interface for some basic and well-known FL aggregators Containerized support with Kubernetes	1. Very small community and contributors. 2. Difficult to use because it uses a little-known DL framework.
FedML	1. No limitations on using edge devices including smartphones and IoT devices. 2. Multi-GPU training support 3. Growing community.	1. No privacy-preserving techniques, only secure aggregation is implemented.
Leaf	1. Contains basic FL mechanisms such as Federated Averaging Aggregator. 2. Very adaptative.	1. Doesn't provide good documentation or tutorials. 2. No benchmarks for privacy preservation in an FL setting.

4 Alchimia Federated Learning

The system requirements and the system architecture of Alchimia will be explained in *D2.1 Requirements and human-centric recommendations* and *D2.2 ALCHIMIA architecture*, respectively. Figure 15 shows the architectural scheme of the project. The features of the architecture that have the most significant impact on the FL framework are the communication between the central server and the clients and the support of different privacy-preserving techniques.

The communication between the central server and participants is based on Apache Kafka, a distributed event-streaming platform for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications [33]. Thus, the FL framework must support this type of communication to fulfil this requirement.

Regarding privacy-preserving support, the framework should be able to apply the main techniques, e.g., PATE, Differential Privacy or Secure Sum.

Therefore, the FL framework to use in this project must offer a high degree of flexibility so that all the constraints are overcome.

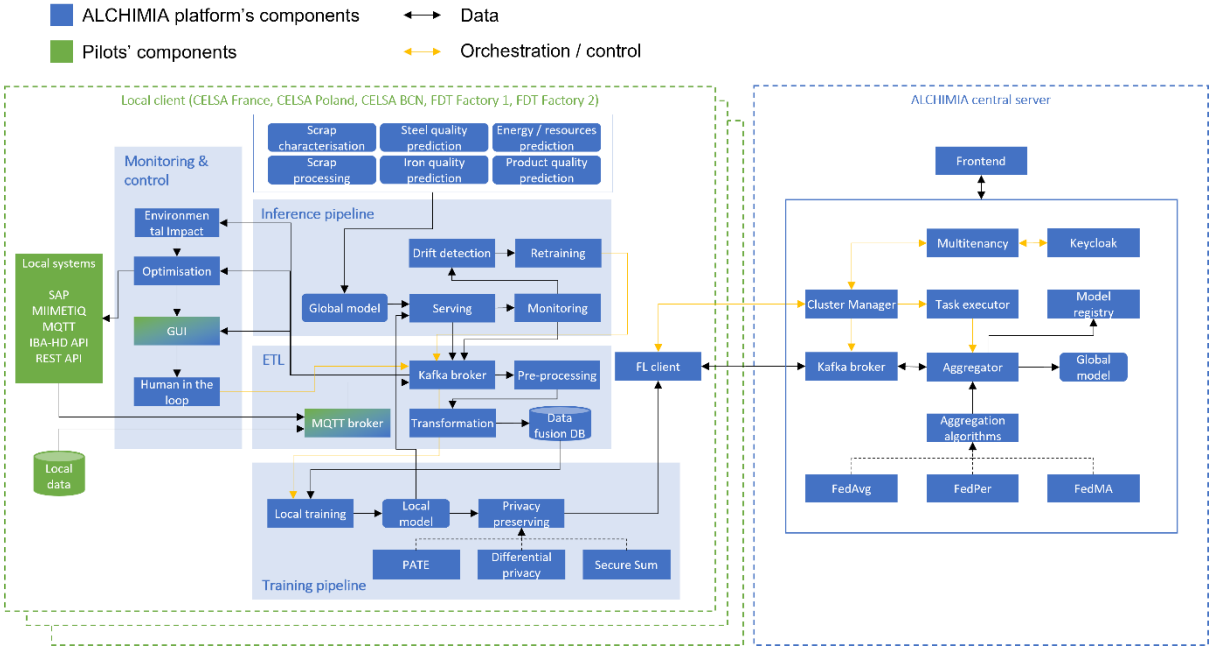

Figure 15. Alchimia system architecture.

In section 3, a comparison among the principal open-source FL frameworks has been provided. However, none of them aligns perfectly with the requirements of the project, either due to inadequate support for all privacy-preserving techniques, the inability to establish communication between server and participants through Apache Kafka or another platform based on pub/sub pattern, or due to their lack of flexibility.
For these reasons, the FL system will be built upon the Atos FL framework. A basic FL framework that needs to extend its functionalities to support Kafka-based communication and privacy techniques, but given its flexibility, it stands as the optimal choice for meeting the constraints of this project. These framework will be extended and tailored to the needs of Alchimia project and use-cases.

4.1 Design

The main open-source FL frameworks available have been described in the previous sections. However, no one fits all use cases in general. Therefore, the design of the AtosFL framework is presented in this section. With the development of this framework, the aim is to implement a flexible tool that can be adjusted to any FL scenario.

4.1.1 Pipe and Filter Pattern

The main architectural pattern of AtosFL is *pipe and filter*. It is a software design pattern used to process data by breaking down a complex task into a series of smaller, independent processing steps called filters. These filters are connected between them through pipes.

Each filter in the pipeline performs a specific operation on the data. The output of the filter becomes the input of the following filter until data is processed by all filters. Figure 16 depicts the pipe and filter pattern overview.

Figure 16. Pipe and filter pattern scheme.

The key features of using this architecture are the following:

- Loose and flexible coupling components. Thus, different components can be implemented and combined among them in different ways without making any assumption on how they are being used.
- Independency because of loose coupling. This way filters can be changed without modifications in other filters.
- Allows parallel processing.
- Filters can be considered "black boxes". Users do not need to know the inner workings.
- Reusability of filters.

As described in previous sections, FL is an approach that enables ML training across multiple participants while keeping data decentralized and secure. In a centralized FL scenario, each participant should perform at least the following steps:

1. Load local data.
2. Train the local model using local data.
3. Extract local weights.
4. Send local weights to the central server.
5. Receive global weights from the central server.
6. Update local weights with global weights.

This process can be represented as a pipeline as shown in Figure 17. Therefore, it can be implemented following the pipe and filter pattern.

Figure 17. FL Client pipeline.

Regarding the central server perspective, these are the minimal steps needed:

1- Receive local weights from participants.

2- Aggregate local weights to compute global weights.

3- Broadcast global weights to all participants.

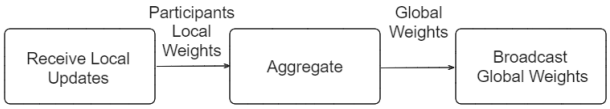

Figure 18. FL server pipeline.

4.1.2 Components

The core component of the framework is called *Pod*. A Pod implementation ignores the distributed aspect of the FL logic by only focusing on small pieces of functionality, e.g., the weights aggregation of the federated server.

A Pod defines wires to interact with the exterior, this entity is the listener design pattern. There are two types of wires: input and output. The main difference is that the pod provides a handler for the input wire which executes a function and then forwards the output through the output wire if needed to connect to the next pod. Therefore, pods can connect between them using input and output wires.

Therefore, everything can be described in terms of Pods. Custom functionalities to address scenario requirements can be provided by extending pods. Thus, defining new wires and assigning default handlers to them. For example, in a client-server federated approach, client and server pods could be present.

The server Pod may define an input wire called */update* that receives the parameter vectors trained by the participants. Once all participants have provided their parameters, the server can perform the aggregation and compute the global parameter vector. This global vector could be forwarded through an output wire called */broadcast* that gets connected to the participants.

The client Pod could present an input wire called */update* that receives the global parameter vector from the server. When this interface is triggered, the client can update its local model and perform a training round with local data and then send the local parameters through an output wire called */trained* which connects to the server.

Using this approach, Pods can be implemented to cover communication protocols to enable participants to share information among themselves (decentralized FL) or with the central server (centralized FL). AtosFL supports HTTP communication protocol. However, more protocols support can be added, especially Apache Kafka.

Regarding privacy-preserving, this concept refers to maintaining the local model parameters of the participants private, as described in section 3. Different privacy

algorithms can be implemented following the POD philosophy. For example, the Differential Privacy method could be supported by implementing a Pod that receives the local weights and adds noise to them before sending the local updates to the server.

The following section provides a description of the implementation of *Kafka Pod*.

4.1.3 Communication

Kafka was selected as the underlying communication protocol due to its robustness, scalability, and real-time data streaming capabilities. Kafka's robust storage mechanism ensures that messages are stored over time, allowing clients to retrieve missed updates when they come back online. This is particularly useful in scenarios where nodes may experience intermittent connectivity.

The integration of Kafka within AtosFL involves the following components:

1. Producers: Producers publish messages to specific Kafka topics representing different communication channels.
2. Consumers: Consumers subscribe to relevant topics and receive messages in real time.
3. Topics: Kafka topics serve as communication channels, organizing and categorizing messages for different aspects of the FL process such as model updates, data availability, or synchronization signals.

Both clients and the server function as producers and consumers. Clients are responsible for transmitting their individual local models, whereas the server sends the aggregated model as part of the communication process. A high-level diagram of the communication is displayed in Figure 19.

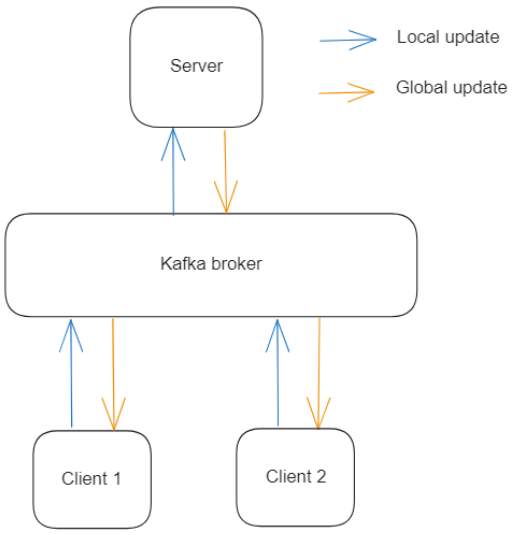

Figure 19: Kafka-based communication overview

Kafka Pod must communicate with other local pods (e.g., client pod) and with the Kafka broker so that it can subscribe to topics and publish messages. Therefore, this pod needs wires to forward messages to local pods and to the broker.

The Kafka Pod interfaces that are at least necessary in a common FL scenario from the perspective of FL clients are as follows:

- Topic subscription to receive global weights.
- Local output interface to forward global weights to the client Pod.
- Local input interface to receive local updates from the client Pod.
- Output interface to publish the local update to the Kafka broker.

From a server perspective, the necessary interfaces are:

- Topic subscription to receive local updates from each participant.
- Local output interface to forward the updates to the server Pod.
- Local input interface to receive global weights from the server Pod.
- Output interface to publish global weights to the Kafka broker.

Figure 20 shows the concept.

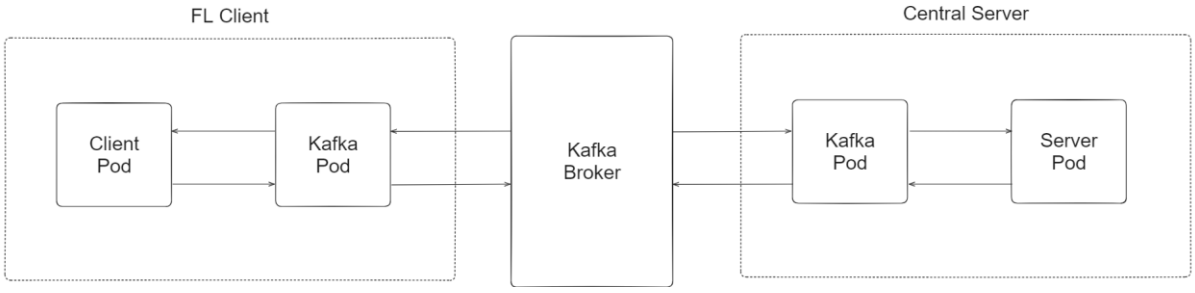

Figure 20. Kafka support through AtosFL.

5 Example

An initial example was conducted making use of the AtosFL framework. The example focuses on predicting the steel grade produced by knowing the scrap characteristics (quality and quantity). The steel grade prediction is identified within *Use Case 3: Determine Predictive Results offline* of CELSA. The model that was developed serves as a mock-up of the models that will be developed for the Alchimia project.

This example has been conducted simulating the Federated Learning training process. The dataset has been divided by the number of clients selected for the training. This approach allows us to achieve meaningful insights and assess the feasibility of Federated Learning for the Alchimia project.

5.1.1 Dataset

The dataset used was provided by CELSA. It contains data from three months measured at the CELSA plant located in France. Among the multiple data that were provided, two *CSV* files were used for this example. The files contain the information related to the charged scrap and the steel analysis.

In the preprocessing stage, some transformations were applied to the data such as merging the files by casting ID, transforming the steel quality into category format, and aggregating the scrap types by casting ID. Finally, the variables that were selected for training the model are displayed in Figure 21.

The columns in the dataset are:

- **QUALITY:** Steel quality. Qualities are considered as integer numbers from 0 to 23.
- **WS0010- WS0100:** Amount of scrap in kilograms (kg). Each column represents a different quality of scrap.

	QUALITY	WS0010	WS0020	WS0022	WS0030	WS0031	WS0032	WS0033	WS0040	WS0060	WS0100
0	22	0.0	0.00	0.0	79.150	0.000	0.0	10.14	48.59	11.120	0.00
1	22	0.0	0.00	0.0	33.660	27.390	0.0	0.00	0.00	0.000	8.77
2	22	0.0	0.00	0.0	51.340	23.989	0.0	0.00	0.00	0.000	10.78
3	22	0.0	0.00	0.0	57.409	0.000	0.0	0.00	0.00	0.000	11.50
4	22	0.0	0.00	0.0	60.509	24.499	0.0	8.31	11.77	0.000	10.66
...	...	...	...	...	...	...	...	...	...	...	...
1059	0	0.0	0.00	0.0	87.950	27.350	0.0	7.28	24.37	10.400	5.95
1060	0	0.0	53.71	0.0	130.829	26.040	0.0	6.50	20.98	10.830	0.00
1061	0	0.0	25.34	0.0	41.270	0.000	0.0	6.86	6.86	11.770	0.00
1062	0	0.0	0.00	0.0	84.451	21.020	0.0	4.15	30.12	14.671	6.08
1063	0	0.0	0.00	0.0	77.450	27.040	0.0	7.31	32.20	16.700	0.00

Figure 21: Example dataset

The columns of the training dataset are divided into target classes (quality) and the corresponding features.

The distribution of the classes is displayed in Figure 22. Notice that the imbalance of the classes may affect the performance of the model.

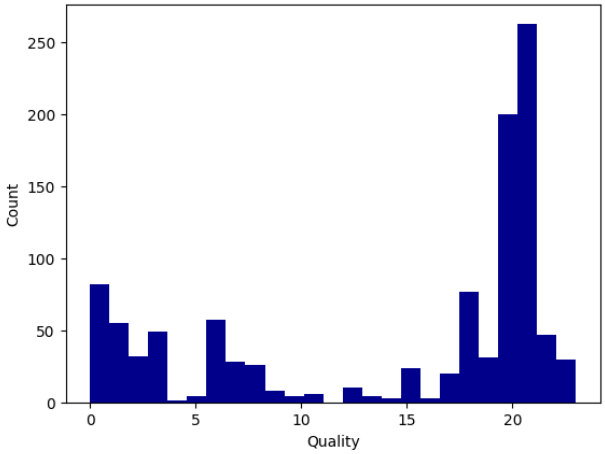

Figure 22: Distribution of quality classes

The number of clients selected for the Federated Learning training is set to 2, therefore the dataset is equally partitioned by this number. This partitioning was performed while trying to maintain the distribution characteristics of the original dataset, although, in future examples, unbalanced datasets will also be tested since they simulate the reality of different data available at each plant.

5.1.2 Federated Learning strategy

The AtosFL framework supports various strategies such as FedAvg and FedWeightedAvg. The strategy that was used in this approach is FedAvg. As explained in Section 2.3, the central server aggregates local model updates using simple averaging resulting in a global model. FedAvg ensures each participant's contribution to influence the global model.

The number of rounds performed in the training is 10. This implies that each local model has been trained 10 times and aggregated at the central server generating a global model. For each training, a number of 100 epochs has been selected and a batch size of 128. Finding the optimal number of rounds for Federated Learning training leads to a more accurate and representative model of the entire dataset.

5.1.3 Results

In this section, we present the outcomes and insights extracted from the FL example conducted. The purpose of this example was to evaluate the effectiveness of the AtosFL framework ensuring the collaborative training of a model across distributed nodes.

To compare the results of the model with and without the FL approach, initially, the model was trained in a standard way without making use of the FL framework. After, the same model was trained using the Federated Learning framework with a small group of client nodes.

The model consists of a neural network created using the Keras library, which provides a high-level interface for designing and training these types of models.

The neural network comprises 3 fully connected (Dense) layers. Figure 23 displays a diagram of the architecture of the network. The first two layers use the Rectified Linear

Unit (ReLu) activation function $f(x) = max(0, x)$. ReLu is one of the most widely used activation functions due to its simplicity and computational efficiency and often achieves better performance.

The last layer of the network which serves as the output of the model uses the SoftMax activation function. SoftMax is commonly used for multi-class classification problems because it converts the network's output values into probability values that sum up to 1. The number of outputs of the network is configured as 24 which is the number of steel quality classes present in the dataset.

The loss function used for training the model is the Sparse Categorical Cross-entropy and the optimizer algorithm selected is Adaptive Moment Estimation (Adam). Last, for measuring the performance of the model the accuracy metric is monitored while training.

In Figure 24 the code definition of the network is shown.

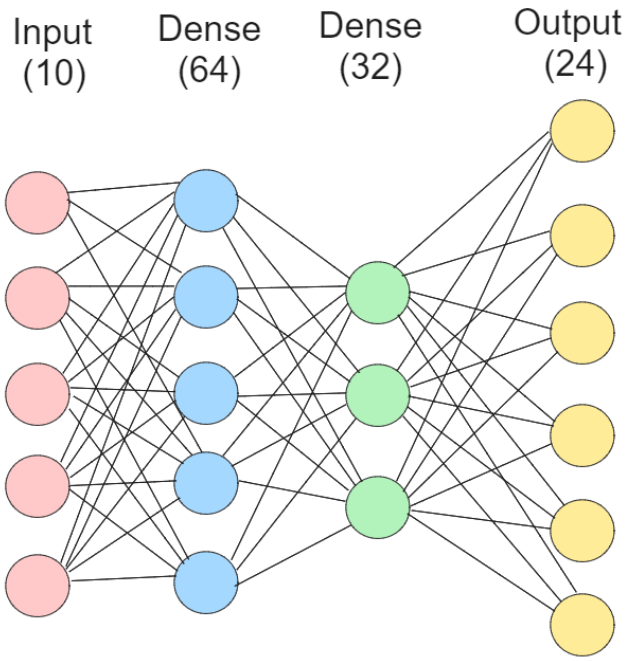

Figure 23: Neural network diagram

```
model = Sequential()
model.add(Dense(units=64, input_dim=10,  activation='relu'))
model.add(Dense(32, activation="relu"))
model.add(Dense(24, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Figure 24: Neural network definition

For developing the model without the Federated Learning approach, the complete dataset was used in the training stage. The training accuracy achieved with 100 epochs is 46 %.

For the FL approach, over a series of training rounds, the model's performance improved gradually. Allowing the model to learn from the data within the different nodes. In the last round, the model reached a training accuracy of 54 %. In Figure 25 a screenshot with the training logs is exhibited, notice the progression of the accuracy on each node is printed.

Taking into consideration that in the non-federated training process there is no more than one training round, the performance of the model could be higher if the number of epochs increases (as it is happening when FL rounds are added).

A conclusion from the limited-scale example is that as we added more clients to the training process, the communication protocol continued to facilitate effective coordination and model updates. This indicates the capability of adding extra nodes in more complex scenarios.

Lastly, the Kafka-based communication protocol demonstrated efficiency in transmitting updates between clients and the server.

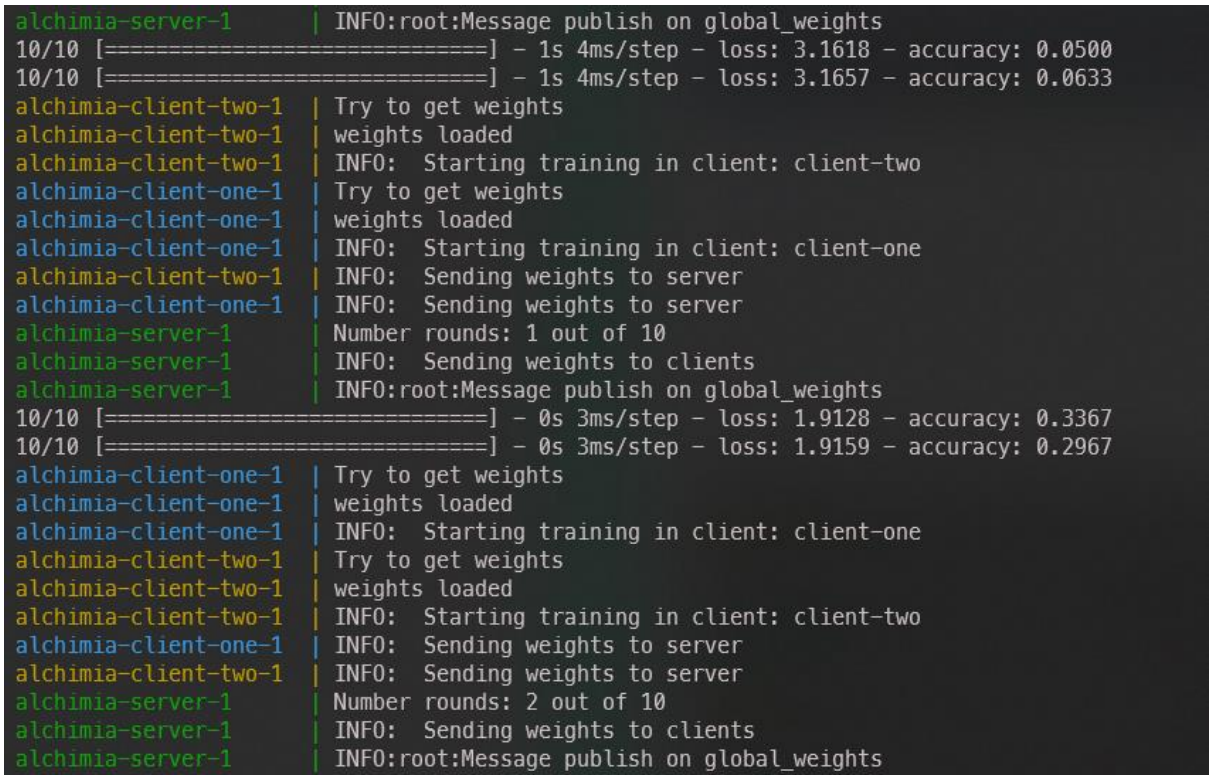

Figure 25: Federated Learning training logs

6 Conclusions

This deliverable provided an analysis of the state of the art in the field of Federated Learning with the challenges still present when deploying an FL system. Moreover, a study of privacy-preserving techniques is designed to enhance privacy protection of sensitive information and prevent risks of attacks. Moreover, a brief description of the main open-source FL frameworks is given together with a comparison among them considering the supported functionalities, communication protocols and the offered privacy-preserving techniques.

The report also presents the FL framework that is going to be developed during the project. The design of the framework is focused on being as flexible as possible and that it can be easily adjusted to any FL scenario. In addition, the results of a demo where the application of the FL framework are provided.

During the next steps of Alchimia's activities towards the design, development, and test of the whole FL system the following tasks will be addressed:

- Complete FL framework development.
- Integration of FL framework according to the specifications provided in *D2.2 ALCHIMIA architecture*.
- Deployment of the FL system following requirements provided in *D2.1 Requirements and human-centric recommendations*.

7 References

[1] E. M. D. R. S. H. B. A. y. A. H. Brendan McMahan, «Communication-Efficient Learning of Deep Networks from Decentralized Data,» 2016.

[2] S. T. M. E. G. L. L. Vale Tolpegin, «Data Poisoning Attacks Against Federated Learning Systems,» 2020.

[3] P. K. V. J. M. D. W. P. Anshuman Suri, «Subject Membership Inference Attacks in Federated Learning,» 2022.

[4] Y. L. Y. Z. X. Z. Yupeng Jiang, «Mitigating Sybil Attacks on Differential Privacy based Federated Learning,» 2018.

[5] M. L. L. L. Yue Zhao y D. C. V. C. Naveed Suda, «Federated Learning with Non-IID Data,» 2018.

[6] Z. A. D. F. C. G. L. E. A. José Ángel Morell, «Optimising Communication Overhead in Federated Learning Using NSGA-II,» 2022.

[7] E. N. A. S. Elena Fedorchenko, «Comparative Review of the Instruction Detection Systems Based on Federated Learning: Advantages and Open Challenges,» 2022.

[8] N. B. Heiko Ludwig, Federated learning: A comprehenive Overview of Methods and Applications, San Jose, CA, USA: Springer, 2021.

[9] G. S. C. R. M. K. C. M. D. Q. V. L. M. Z. M. M. R. A. S. P. T. K. Y. A. Y. N. Jeffrey Dean, «Large Scale Distributed Deep Networks,» 2012.

[10] V. A. A. K. S. S. C. Manoj Ghuhan Arivazhagan, «Federated Learning with Personalization Layers,» 2018.

[11] M. L. Z. D. Z. L. X. H. Fei Chen, «Federated Meta-Learning with Fast Convergence and Efficient Communication,» 2018.

[12] A. K. S. M. Z. M. S. A. T. V. S. Tian Li, «FEDERATED OPTIMIZATION IN HETEROGENEOUS NETWORKS,» 2020.

[13] A. M. H. H. A. J. R. P. Amirhossein Reisizadeh, «FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization,» 2019.

[14] «TensorFlow Federated: Machine Learning on Decentralized Data,» [En línea]. Available: https://www.tensorflow.org/federated?hl=es-419.

[15] «PySyft's documentation,» [En línea]. Available: https://openmined.github.io/PySyft/.

[16] «FATE Documentation,» [En línea]. Available: https://fate.readthedocs.io/en/latest/.

[17] «Flower Documentation,» [En línea]. Available: https://flower.dev/.

[18] «PaddleFL Documentation,» [En línea]. Available: https://paddlefl.readthedocs.io/en/latest/.

[19] «FedML Documentation,» [En línea]. Available: https://doc.fedml.ai/.

[20] S. M. K. D. P. W. T. L. J. K. H. B. M. V. S. A. T. Sebastian Caldas, «LEAF: A Benchmark for Federated Settings,» 2018.

[21] E. Y. D. F. ,. E. S. E. N. E. F. M. N. Ivan Kholod, «Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis,» 2020.

[22] «Pytorch Documentation,» [En línea]. Available: https://pytorch.org/.

[23] D. J. B. P. P. B. d. G. J. F.-M. T. T. X. Q. T. P. Y. G. N. D. L. Akhil Mathur, «ON-DEVICE FEDERATED LEARNING WITH FLOWER,» 2021.

[24] «PadePadle Documentation,» [En línea]. Available: https://github.com/PaddlePaddle/Paddle.

[25] «ZeroMQ Documentation,» [En línea]. Available: https://zeromq.org/get-started/.

[26] S. L. J. S. Chaoyang He, «FedML: A Research Library and Benchmark for Federated Machine Learning, 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,» 2020.

[27] C. L. Ú. E. J. K. a. D. S. N. Carlini, «The secret Sharer: Evaluating and testing unintended memorization in neural networks,» *Proceedings of the 28th USENIX,* p. 267–284, 2019.

[28] C. K. K. M. F. M. I. N. M. Dwork, «Our Data, Ourselves: Privacy Via Distributed Noise Generation,» de *Advances in Cryptology - EUROCRYPT 2006*.

[29] K. S. S. W. F. G. Y. G. Nguyen Truong, «Privacy preservation in federated learning: An insightful survey from the GDPR perspective,» *Computers & Security,* vol. 110, p. 102402, 2021.

[30] C. Gentry, A fully homomorphic encryption scheme, 2009.

[31] A. C.-C. Yao, «How to generate and exchange secrets,» de *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, Toronto, Canada, 1986.

[32] M. A. Ú. E. I. G. K. T. Nicolas Papernot, «Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data».

[33] «Apache Kafka documentation,» [En línea]. Available: https://kafka.apache.org/.

[34] M. S. L. R. S. S. G. S. M. a. M. H. Farshid Varno, «AdaBest: Minimizing Client Drift in Federated Learning via Adaptive Bias Estimation,» 2022.