A Tutorial on Federated Learning from Theory to Practice: Foundations, Software Frameworks, Exemplary Use Cases, and Selected Trends

M. Victoria Luzón, Nuria Rodríguez-Barroso, Alberto Argente-Garrido, Daniel Jiménez-López, Jose M. Moyano, Javier Del Ser, *Senior Member, IEEE*, Weiping Ding, *Senior Member, IEEE*, and Francisco Herrera, *Senior Member, IEEE*

***Abstract*—When data privacy is imposed as a necessity, Federated learning (FL) emerges as a relevant artificial intelligence field for developing machine learning (ML) models in a distributed and decentralized environment. FL allows ML models to be trained on local devices without any need for centralized data transfer, thereby reducing both the exposure of sensitive data and the possibility of data interception by malicious third parties. This paradigm has gained momentum in the last few years, spurred by the plethora of real-world applications that have leveraged its ability to improve the efficiency of distributed learning and to accommodate numerous participants with their data sources. By virtue of FL, models can be learned from all such distributed data sources while preserving data privacy. The aim of this paper is to provide a practical tutorial on FL, including a short methodology and a systematic analysis of existing software frameworks. Furthermore, our tutorial provides exemplary cases of study from three complementary perspectives: i) *Foundations of FL*, describing the main components of FL, from key elements to FL categories; ii) *Implementation guidelines and exemplary cases of study*, by systematically examining the functionalities provided by existing software frameworks for FL deployment, devising a methodology to design a FL scenario, and providing exemplary cases of study with source code for different ML approaches; and iii) *Trends*, shortly reviewing a non-exhaustive list of research directions that are under active investigation in the current FL landscape. The ultimate purpose of this work is to establish itself as a referential work for researchers, developers, and data scientists willing to explore the capabilities of FL in practical applications.**

***Index Terms*—Data privacy, distributed machine learning, federated learning, software frameworks.**

Manuscript received September 8, 2023; revised October 4, 2023 and November 20, 2023; accepted December 21, 2023. This work was partially supported by the R&D&I, Spain grants PID2020-119478GB-I00 and, PID2020-115832GB-I00 funded by MCIN/AEI/10.13039/501100011033. N. Rodríguez-Barroso was supported by the grant FPU18/04475 funded by MCIN/AEI/10.13039/501100011033 and by "ESF Investing in your future", Spain. J. Moyano was supported by a postdoctoral Juan de la Cierva Formación grant FJC2020-043823-I funded by MCIN/AEI/10.13039/5011000 11033 and by European Union NextGenerationEU/PRTR. J. Del Ser acknowledges funding support from the Spanish *Centro para el Desarrollo Tecnológico Industrial* (CDTI) through the AI4ES project, as well as from the Department of Education of the Basque Government (consolidated research group MATHMODE, IT1456-22). Recommended by Associate Editor Zhengcai Cao. *(Corresponding author: Nuria Rodríguez-Barroso.)*

Citation: M. V. Luzón, N. Rodríguez-Barroso, A. Argente-Garrido, D. Jiménez-López, J. M. Moyano, J. Del Ser, W. Ding, and F. Herrera, "A tutorial on federated learning from theory to practice: Foundations, software frameworks, exemplary use cases, and selected trends," *IEEE/CAA J. Autom. Sinica*, vol. 11, no. 4, pp. 824–850, Apr. 2024.

M. Luzón is with the Department of Software Engineering, Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Granada, Granada 18071, Spain (e-mail: luzon@ugr.es).

N. Rodríguez-Barroso, A. Argente-Garrido, D. Jiménez-López, J. Moyano, and F. Herrera are with the Department of Computer Science and Artificial Intelligence, Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Granada, Granada 18071, Spain (e-mail: rbnuria@ugr.es; aargente@ugr.es; dajilo@ugr.es; jmoyano@ugr.es; herrera@ugr.es).

J. Del Ser is with the Department of Communications Engineering, University of the Basque Country (UPV/EHU), and also with TECNALIA, Basque Research & Technology Alliance (BRTA), Spain (e-mail: jdelser@tecnalia.com).

W. Ding is with the School of Information Science and Technology, Nantong University, Nantong 226019, China (e-mail: ding.wp@ntu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JAS.2024.124215

I. Introduction

DEEP learning has revolutionized the field of artificial intelligence (AI) by enabling machines to learn and make decisions like humans through data-driven techniques [1]. The development of high-speed networks such as 5G and advances in edge computing have supported the development of hardware and models capable of processing large amounts of data collected from multiple devices. Consequently, privacy awareness has become a major design driver, shifting the focus from centralized machine learning (ML) to distributed ML. Still, in distributed ML the communication costs far outweigh the compute costs, making the training process inefficient [2]. FL [3], [4] was conceived to address these issues. In essence, FL is a distributed learning paradigm that enables model learning from decentralized data, without the need for collecting data on a central server. Since local data never leaves the device where it was collected, data privacy is guaranteed.

FL has gained significant attention due to its ability to address privacy concerns and improve the efficiency of distributed learning [5]–[7]. Additionally, it is highly scalable as it can accommodate numerous participants, each with their data sources. This can be particularly useful in scenarios with continuous data generation, e.g., Internet of Things (IoT) sensor devices. As a result, FL has become an important AI field, attracting the interest of researchers, developers, and data scientists in the ML community in theoretical and practical studies involving applications that deal with sensitive data. The first successful application of FL was developed by Google to predict user's text input within tens of thousands of Android devices, while keeping data locally on devices [3]. Since then, FL has been applied to a wide range of applications in diverse fields, from industrial engineering to healthcare [8]. It has been also explored for drug discovery from biological and chemical data in real-world cross-national settings [9].

In this context, we can find many overviews and general studies on FL, mostly from a scientific perspective. However, the literature lacks a tutorial on FL that considers essential aspects for the understanding, analysis and use of this research area with a complete view from theory to practice. We provide the following elements: a revision of the key elements, architectures, and categories defined in FL; a design methodology; a systematic analysis of software frameworks from a practical perspective, along with cases of study; and research trends for a double perspective of FL components versus ML approaches. This is indeed the goal of this tutorial paper: to provide a valuable source of information for anyone interested in learning about this cutting-edge research area and in applying it to real-world problems. Specifically, the contributions of this tutorial articulate around three different axis or perspectives:

1) We present a clear insight into the FL field, discussing the foundations, ranging from the description of its main components, from key components to architectures and categories.

2) We provide a practical perspective, including a short methodology, the software frameworks analysis and exemplary cases of study. First, we provide a methodology to design a FL scenario and experiments. Second, we provide an analysis of the functionalities provided by software frameworks for FL deployment, and show the maturity of the field, thorough the simplicity and immediacy of assembling a FL architecture with these frameworks. Third, we provide exemplary cases of study with source code for different ML approaches, such as image classification and sentiment analysis (SA) focused on the data, decision trees in vertical FL from a model analysis focused on interpretability, clustering as non-supervised learning, and differential privacy (DP) as an approach to preserve data integrity and privacy. This collection of cases of study aims to give a broad practical view of the possibilities of FL, including the availability of the code used for them.

3) We analyze the state of the art and prospects of the field, providing a non-exhaustive review of trends under a double prism: i) Trends that are inherent to FL; and ii) Trends in federated ML. The selected trends include attacks and defenses in FL, personalized FL, federated transfer learning, and ML tasks such as semi-supervised FL, anomaly detection, natural language processing (NLP) or SA.

According to these aims, the paper is organized as follows. Section II introduces the concept of FL considering 3 fundamental questions: *Why? What?* and *What for?* Section III explores the foundations, introducing the principal components needed to fully understand the foundations of FL, from key elements to architectures and categories, with a short view of techniques to ensure the data privacy. Section IV examines an ecosystem available in federated scenarios, focusing on federated datasets and a wide range of software frameworks. Section V describes the methodology for designing a FL scenario, depicting a method to put into practice the theoretical concepts introduced previously. Section VI provides a comparison between FL and non-FL scenarios. Section VII discusses different use cases and methods to solve them by using three different software frameworks selected from the study conducted in the previous section. Section VIII elaborates on the selected FL trends. Section IX concludes this tutorial with a summary and outlook. Finally, we include Appendix which shows a list of abbreviations and notations.

II. Federated Learning: Why, What and What for?

Data-driven ML has mastered the AI field [10]. Unfortunately, increasing demands in terms of data volume and variety have resulted in several challenges related to data privacy and the processing of such large amounts of data. Among them, the main ML challenges from which FL emerges are associated with privacy, communication, and data access, which are next discussed shortly:

- *Data Privacy:* In centralized ML, users' data is often collected and stored in a central server, where it can be vulnerable to privacy breaches [11]. This is particularly concerning in some fields such as healthcare [12], finance, and other industries where data privacy is of utmost priority. Moreover, growing concerns about safeguarding of data-privacy manifests in the legal area with, for example, recently published recommendations [13]. Consequently, the development of privacy-preserving AI methods is in urgent demand in such fields.
- *Communication Costs and Latency [14]:* In centralized ML, the raw data is often transmitted to a central server to be processed and used to train ML models [15]. This information exchange can be costly and time-consuming, especially when dealing with large datasets [16]. Furthermore, the increasing amount of data available due to the explosion of IoT sensors [17] and the proliferation of edge devices generating vast amounts of data poses a new challenge related to the storage and preprocessing of data continuously flowing from different sources.
- *Limitations in Data Access [12]:* In some cases, data can be distributed across different institutions or organizations, making it difficult to access or share data between them or with others.

In order to address the above challenges, FL [4] emerges as a distributed ML paradigm aimed at developing a ML model without explicitly sharing any data between any of the participants. It involves a network of clients or data owners $\{C_1, C_2, \ldots, C_n\}$, which takes part in two primary phases:

1) Model training phase, in which each data owner exchanges information without revealing any of their data to collaboratively train a ML model. For that purpose, each data owner trains a local learning model on its data and shares this learning model's information instead of their training data. Then, the trained local models are aggregated to create a trained global learning model (see Fig. 1).

2) Inference phase, where the trained global model is applied to new data instances.

These processes can be either synchronous or asynchronous, depending on the data availability of nodes and the trained

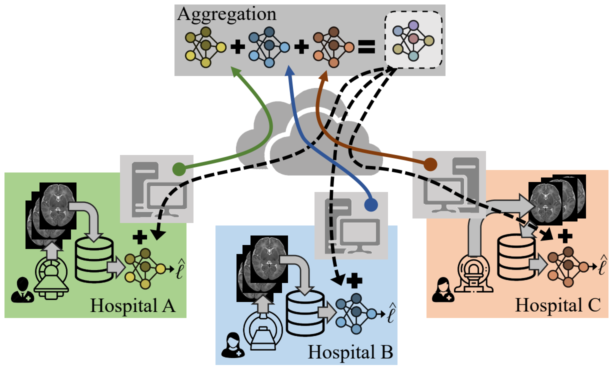

Fig. 1. Generic FL medical use case, where Magnetic Resonance Image data is collected at three different hospitals and modelled locally for a clinical diagnostic application. Model updates are uploaded to a central server (→, →, →) and aggregated to yield a trained global model, which is then delivered downstream (--→) to the hospitals and combined with their local models. As a result, the combined local model leverages knowledge modelled by other hospital for the same clinical task, while keeping local data private.

model. It is important to note that privacy is not the only reason for this approach, as there should also be a fair value-distribution mechanism in place to share the profits gained by the collaboratively trained model.

Once we have described FL as a general concept, a FL scenario can be formally posed as follows. We assume a set of clients or data owners $\{C_1,\ldots,C_n\}$ with their respective local training data $\{D_1,\ldots,D_n\}$. Each of these clients C_i owns a local learning model L_i expressed as the parameters $\{L_1,\ldots,L_n\}$. FL aims to learn a global learning model G, using scattered data across clients through an iterative learning process known as *round of learning*. For that purpose, in each learning round t, each client trains its local model over their local training data D_i^t, resulting in the update of the local parameters L_i^t to $\hat{L}_i^t$. Thereafter, the global parameters G^t are computed by aggregating the trained local parameters $\{\hat{L}_1^t,\ldots,\hat{L}_n^t\}$ using a fixed federated aggregation operator Δ, and the local learning models are updated with the aggregated parameters

$$G^t = \Delta(\hat{L}_1^t,\hat{L}_2^t,\ldots,\hat{L}_n^t)$$
$$L_i^{t+1} \leftarrow G^t, \quad \forall i \in \{1,\ldots,n\}. \tag{1}$$

Updates among the clients and the server are repeated for the learning process until a given stop criteria is met. Thus, the final value of G will sum up the knowledge modelled in the clients.

Finally, we pause at the question *What for?* The design of this distributed learning paradigm allows for training models on data that is not easily collectable or centralized, providing a solution to the problems addressed before because of the following features:

- *Data Privacy:* FL addresses this leakage by allowing the model to be trained on the data where it is allocated, without sharing any information about data to a central server. This way, sensitive data remains on users' devices and is never shared, thus preserving users' privacy.
- *Communication Costs and Latency:* FL addresses this challenge by allowing only model updates to be exchanged between the server and the clients. This approach addresses the latency and high bandwidth issues present in other distributed training processes, where the training algorithm is modified to support distributed computations and the data is downloaded from a central data provider or data silo [18]. Consequently, FL is more efficient and scales better than traditional distributed ML strategies.
- *Data Access:* FL solves this challenge by enabling collaborative ML across different institutions or organizations, without requiring them to share any data [19]. FL effectively overcomes the requirement of a central data provider and enables applications of ML in various domains that are sensitive to data privacy, promoting collaboration and innovation in the field.

FL is driven by the challenges of privacy, communication costs, and data access limitations that are inherent in centralized ML. It offers a promising approach for training ML models on decentralized data, while addressing these challenges and enabling new applications of ML in various domains [8]. The healthcare domain has greatly leveraged the use of FL thorough the development of the so-called Internet of Healthcare Things (IoHT), which allows knowledge from different sources to be combined in order to better determine patient health status and identify possible anticipatory actions [20], [21]. An interesting use case of FL within the IoHT context is medical imaging for COVID-19 detection [22]. Moreover, the industrial engineering domain has found multiple successful applications of FL such as detecting defects in production tasks [23] or malicious attacks detection in communication systems supported by unmanned aerial vehicles [24].

It is worth emphasizing that FL can be utilized in several ways, for instance, it can depend on the roles or the nature of the data of each of the nodes involved. All these aspects are covered in Section III.

III. Federated Learning Foundations: Key Elements, Architectures, Categories and Data Privacy

In this section, we explore the principal components needed to fully understand the FL foundations. In the following, we introduce the principal workflow and key elements of FL in Section III-A (discussing the local training, communication, model aggregation and local models update), to continue with multiple FL architectures (client-server and peer-to-peer) in Section III-B. Then, we introduce the main FL categories according to different criteria in Section III-C (based on data features, labels and sample space). Finally, we briefly discuss the most used techniques to ensure the data privacy in Section III-D.

A. Workflow and Key Elements in Federated Learning

Once FL has been briefly introduced, we can move on to talk about the main workflow of a FL process. In Fig. 2 we show the different steps that compose FL training. In the following, we further explain the steps of the workflow and specify the key elements [25] which arise from each of the steps.

a) Local Training: It starts with the local training of each of the local ML models by each of the data owner nodes. Generally, all these locally trained learning models have a shared

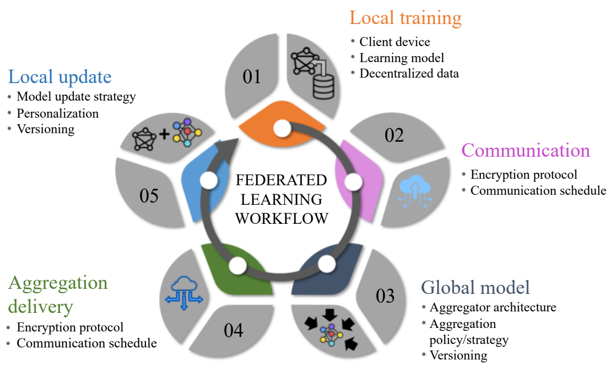

Fig. 2. Block diagram of the FL workflow.

architecture. However, all the aspects concerning training hyperparameters (such as number of epochs, batch size, learning rate) may differ among clients. In this step, the first key elements appear naturally:

• *Decentralized Data:* Data is distributed among different devices or *nodes*, instead of being in a centralized location, which is beneficial when the data privacy and security are a concern. Moreover, such data is inaccessible and not shared with any third-party. The data distribution across the clients can be:

1) Homogeneous or independent and identically distributed (IID): It assumes that the data distribution across the clients is IID, which means that the data of each client follows the same underlying data distribution.

2) Heterogeneous or non independent and identically distributed (non-IID): It assumes that the data distribution across the clients is non-IID, that is, the data of each client follows a different data distribution. Formally, we can distinguish between three types of data distribution heterogeneity [26]: i) Where the feature space of the clients' data are different, but they share the same goal; ii) Where the input space is analogous, but there are differences in the label space according to the data; and iii) When there are differences in both the feature and label spaces.

• *Learning Model:* The training of the learning model is performed on the decentralized data, where each device or node trains its model and contributes to the training process, sharing the weights of its local learning model. It also improves the model due to a better generalization, given that the model can learn from a broader range of data.

• *Clients:* These nodes store data and train models, and are usually referred to as *clients* (see Section III-B).

b) Communication: After local training, the communication enables the coordination and aggregation of model updates generated by the participating nodes, allowing the decentralized training. It plays a crucial role in the protection of the privacy and security of the data when paired with data security techniques like (DP) or (SMC). We highlight the following key elements from this step:

• *Communication Schedule:* The communication can be both synchronous and asynchronous, depending on the configuration. There may also be a central server that handles the collection of all local models, or it may be distributed across multiple nodes in the network.

• *Privacy Protocols:* Although no training data is shared during FL communications, the information shared is susceptible to privacy leaks or corrupting the entire learning process [27]. Hence, communications are one of the weak points of FL regarding susceptibility to attacks. For this reason, it is usually combined with other privacy mechanisms (see Section III-D).

c) Aggregation: The local model updates generated by each node are combined by means of a specific aggregation operator and the result is incorporated to update and create a trained global learning model. The key element in this step is the *aggregation mechanism*, which depends on the task addressed. However, the most common one is *Federated Averaging (FedAvg)* [14] when the ML model can be expressed as a vector of weights. Otherwise, as in clustering, for example, a specific aggregator must be designed to combine the information from each node.

d) Local Update: The last step consists of updating the local models stored in the different nodes with the new global model. The simplest case is to update all local models with this new global model. However, there are different update strategies that consist of combining the local and global models rather than replacing them directly. These approaches are used to achieve features such as personalization of the clients to their local data.

B. Federated Learning Architectures

The combination of the key elements generates multiple FL architectures, that defines their interrelationship [4], both client-server and peer-to-peer:

• **Client-server architecture**. There is a manager node responsible for the coordination and aggregation of model updates named the ***server*** and the rest of nodes which own data and are responsible for training their local models named the ***clients***. This is easy to implement, but it requires a high level of trust in the server. This degree of reliance is its main weakness, as a result it is vulnerable to attacks. We represent this architecture in Fig. 3.

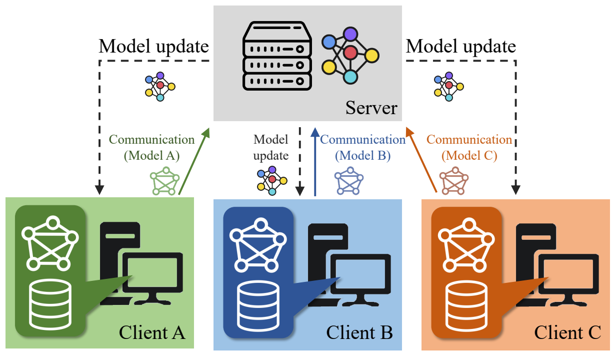

Fig. 3. Representation of client-server FL architecture with 3 clients.

• **Peer-to-peer architecture**. All the nodes own both the training data and aggregate model updates of other nodes. It doesn't require any fixed coordinator of the learning process. This is complex to implement, and the communication costs increase, but the main advantages are an elevated level of security and data privacy. We represent this architecture in Fig. 4.

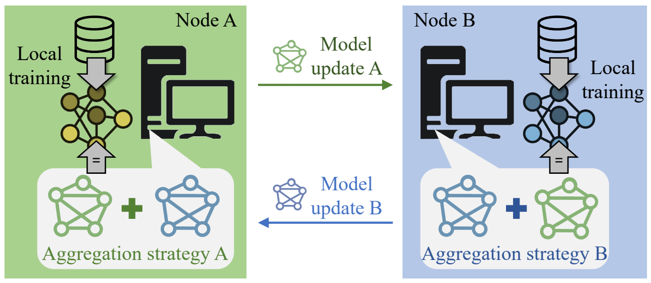

Fig. 4. Representation of peer-to-peer FL architecture.

To the best of our knowledge, the *client-server* architecture is the most common in FL; consequently we will refer to it as the default architecture when FL is being discussed.

C. Federated Learning Categories

There are several categories of FL according to properties of the key elements. We consider the following properties of the decentralized data key element to be the ones that generate the most relevant FL categories:

Data Feature, Label and Sample Space: The decentralized nature of FL may induce bias and heterogeneity in the local data distribution due to different circumstances and factors, such as cultural, ethnic or age differences between the users generating such data. Based on the dimension in which the data is partitioned across clients, there are different categories [4]. We define the following categories in terms of the feature space (X), the label space (Y) and the sample space (I) as follows:

• *Horizontal Federated Learning (HFL):* When data is partitioned across clients based on the samples, which means that each client owns different samples of the overall training dataset. Formally, we can define it as

$$X_i = X_j,\ Y_i = Y_j,\ I_i \neq I_j,\ \forall D_i, D_j,\ i \neq j \tag{2}$$

where the feature and label space of the clients (i, j) is depicted by (X_i, Y_i) and (X_j, Y_j) and it is assumed to be the same, while the samples I_i and I_j are not the same. D_i and D_j depict the data of the clients i and j. It is suitable for training models on data collected from numerous similar devices, such as smartphones or IoT devices.

• *Vertical Federated Learning (VFL):* When data is partitioned across clients based on the features, which means that each client owns the same set of samples, but a different set of features. Formally, we can define it as

$$X_i \neq X_j,\ Y_i \neq Y_j,\ I_i = I_j,\ \forall D_i, D_j,\ i \neq j. \tag{3}$$

It is suitable for training models on data collected for a small number of devices with different feature space. For example, it can be used to predict medical outcomes based on data collected from multiple hospitals, where each hospital has a different set of medical records.

• *Federated Transfer Learning (FTL):* When knowledge is transferred across multiple domains without any overlap between samples or features [28]. Formally, we can define it as

$$X_i \neq X_j,\ Y_i \neq Y_j,\ I_i \neq I_j,\ \forall D_i, D_j,\ i \neq j. \tag{4}$$

In this architecture, it is not assumed that the distribution of training and test data are the same and they are defined in the same feature space. It is usually used in combination with fine-tuning techniques over large models pretrained using a centralized dataset.

Fig. 5 represents the differences between HFL, VFL and FTL according to the data and its features shared between different clients.

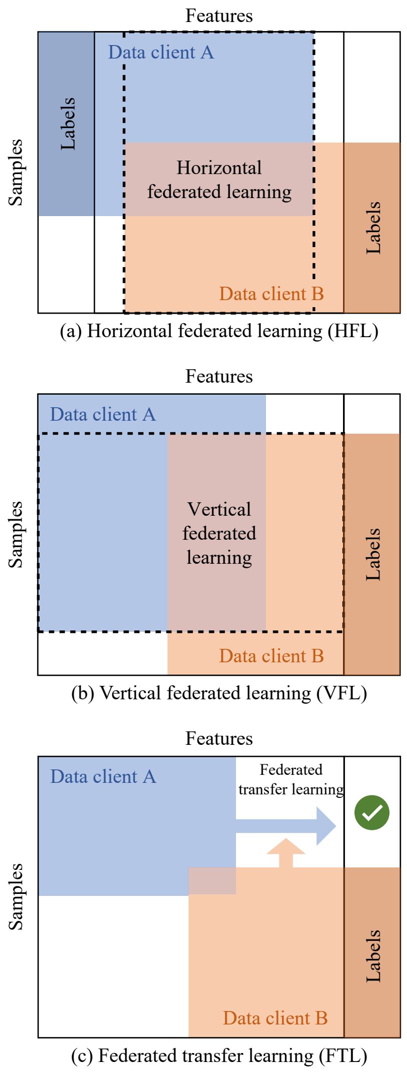
Features
Samples
Labels
Data client A
Horizontal federated learning
Labels
Data client B
(a) Horizontal federated learning (HFL)
Features
Samples
Data client A
Vertical federated learning
Labels
Data client B
(b) Vertical federated learning (VFL)
Features
Samples
Data client A
Federated transfer learning
Labels
Data client B
(c) Federated transfer learning (FTL)

Fig. 5. Representation of HFL, VFL and FTL categories in a client-server FL [4].

D. Data Privacy: Advanced Approaches

FL is built with privacy in mind, that is, clients' data remains private across the FL model training. However, it is possible to break such privacy guarantees through the exchanged models during the learning process, as local client models are prone to memorization of their training dataset. A malicious node can try to recover some part of the private training dataset from other clients, inducing a privacy leakage. Therefore, data privacy techniques are required to enhance the privacy guarantees of a FL model. We consider that these techniques can be deployed in multiple elements of the FL

TABLE I
DATASETS FOR BENCHMARKING FEDERATED SCENARIOS. THE FED. DIST. COLUMN INDICATES IF THE DISTRIBUTION OF THE DATASET IS INHERENTLY FEDERATED. THE REFS. COLUMN INDICATES STUDIES WHERE THE DATASET HAS BEEN USED

Dataset	Task	#Instances	#Clients	Fed. dist.	Category	Refs.
CelebA	Image classification	200 288	9343	Yes	HFL	[37], [38], [39]
Cifar100	Image classification	60 000	–	No	HFL	[40], [41], [42]
Fashion MNIST	Image classification	70 000	–	No	HFL	[43], [44], [45]
FEMNIST	Image classification	805 263	3550	Yes	HFL	[40], [45], [46]
Google landmark v2	Image classification	164 172	1262	Yes	HFL	[47], [48], [49]
iNaturalist	Image classification	155 941	9275	Yes	HFL	[47], [50], [51]
MedMNIST	Image classification	708 069	–	No	HFL	[52], [53], [54]
MNIST	Image classification	70 000	–	No	HFL	[43], [44], [46]
Shakespeare	Text prediction	4 226 150	1129	Yes	HFL	[41], [50], [55]
Reddit	Text prediction	56 587 343	1 660 820	Yes	HFL	[56], [57], [58]
Stack Overflow	Text prediction	168 895 995	585 323	Yes	HFL	[41], [55], [59]
Sentiment140	SA	1 600 498	660 120	Yes	HFL	[40], [46], [50]
Adult	Classification	48 842	–	No	VFL	[60], [61], [62]
Credit2	Classification	30 000	–	No	VFL /FTL	[63], [64], [65]

architecture, which we will elaborate on in the following short sections, considering SMC & homomorphic encryption (HE) and DP.

1) Secure Multiparty Computation & Homomorphic Encryption: SMC is aimed at securing the communications in the FL rounds, mainly focusing on the aggregation procedure. Communication channels are kept safe through HE [29]. SMC often employs HE as a tool to ensure that there are no agents manipulating the communication protocols to either completely deny the communication process or intercept the models exchanged. SMC mostly focuses on computing the aggregation so that sensitive data such as parameters are kept hidden to FL nodes that manipulate them [30]. While these techniques avoid external or internal inference in the FL rounds, the resulting FL model is still vulnerable to attacks that extract information from the aggregated model itself [27]. This motivates the usage of data privacy techniques that modify the learning process to ensure the aggregated FL model is protected as well as the individual clients' models are.

2) Differential Privacy: DP is a data privacy enhancing technique aimed at ensuring the indistinguishability of the data used, that is, it hides the presence of individuals. This is achieved through the addition of calibrated random noise [31]. When applied to FL, DP can be deployed at two stages with diverse privacy guarantees: a) locally training the FL model with DP at the client, known as local differential privacy (LDP) [32], and b) at the aggregation step creating a differentially private version of FedAvg, known as central differential privacy (CDP). LDP provides indistinguishability for clients' data, providing the strongest privacy guarantee at the cost of reduced performance, and CDP ensures indistinguishability in determining whether a client participates in the aggregation step or not, providing a weaker privacy guarantee that improves the performance of the FL model when compared to LDP [33]. There are also some SMC and HE frameworks that integrate DP in their procedures [34].

Overall, data privacy needs to be enhanced in FL tasks. Nevertheless, enhanced data privacy often comes at the cost of worse FL model performance, a trade-off that should be adjusted to each FL scenario.

IV. FEDERATED DATASETS AND STATE-OF-THE-ART SOFTWARE FRAMEWORKS

This section introduces the ecosystem available to design models and studies in federated scenarios. First, the most widely used datasets in the literature to perform FL experiments are presented (see Section IV-A); later, a wide range of state-of-the-art software frameworks for designing such studies, are analyzed from different multiple angles (see Section IV-B).

A. Federated Datasets

Datasets from traditional centralized ML tasks can be reused for simulation purposes by artificially partitioning and sharing the data between the different parties to fit the federated scenario. Nevertheless, there are also some widely used datasets that are considered inherently federated by their features or data distribution. In this regard, it is noteworthy to mention LEAF [35], a benchmarking framework that provides several federated datasets, and TensorFlow Federated (TFF) [36] which also implements some federated datasets.

Table I shows a summary of the most common federated datasets in the literature, including CelebA[1], Cifar100[2], Fashion MNIST[3], FEMNIST$^{\text{IV-A}}$, Google landmark v2$^{\text{IV-A}}$, iNaturalist$^{\text{IV-A}}$, MedMNIST[4], MNIST[5], Shakespeare$^{\text{IV-A}}$, Reddit$^{\text{IV-A}}$, Stack Overflow$^{\text{IV-A}}$, Sentiment140$^{\text{IV-A}}$, Adult[6], and Credit2[7].

The datasets in the table cover different tasks such as computer vision, NLP, and traditional tabular classification, as

[1] https://leaf.cmu.edu/
[2] https://www.tensorflow.org/federated/apidocs/python/tff/simulation/datasets
[3] https://www.kaggle.com/datasets/zalando-research/fashionmnist
[4] https://medmnist.com/
[5] http://yann.lecun.com/exdb/mnist/
[6] https://archive.ics.uci.edu/ml/datasets/adult
[7] https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

TABLE II
STATE-OF-THE-ART SOFTWARE FRAMEWORKS FOR FL (●: FULL SUPPORT; ◑: PARTIAL SUPPORT; ×: NO SUPPORT; –: UNDETERMINED)

	PyS	TFF	FAT	Pad	Flo	Xay	IBM	Sub	OFL	FML	FJx	101	FLb	SFL	EFL	TFL	AFL	NVF
Federated Learning:																		
Horizontal Federated Learning	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Vertical Federated Learning	◑	×	●	●	×	×	×	×	◑	●	×	×	×	×	×	×	×	×
Federated Transfer Learning	×	×	◑	◑	×	×	×	×	×	×	×	×	×	×	×	×	×	●
Support other ML frameworks	●	●	×	×	●	●	●	●	●	●	×	×	◑	×	×	◑	◑	●
Sampling IID or non-IID distribution	●	◑	×	–	×	×	×	×	×	×	×	×	◑	◑	●	●	×	◑
Federated aggregation mechanisms	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑	–	◑	◑	●	◑	●	◑
Adversarial Attacks in FL:																		
Privacy attacks	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Defenses against Privacy attacks	◑	×	◑	◑	×	◑	◑	×	×	×	×	×	×	×	×	×	×	×
Attacks to the federated model	×	×	×	×	×	×	×	×	×	◑	×	●	×	×	×	×	×	×
Defenses against attacks to the model	◑	×	◑	◑	×	◑	◑	×	×	×	×	◑	×	×	×	×	×	×
Differential Privacy (DP):																		
Mechanisms: Exponential, Laplacian...	◑	◑	×	◑	◑	×	×	×	◑	×	×	×	×	×	×	×	◑	●
Subsampling methods to increase privacy	×	◑	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Advanced (DP) Composition	×	●	×	×	×	×	×	●	×	×	×	×	×	×	×	×	×	×
Advanced Properties:																		
Interpretability / Explainability	×	×	◑	×	×	×	◑	×	×	×	×	×	×	×	×	×	×	×
Personalization	×	×	×	×	×	×	×	×	×	×	×	×	◑	×	×	×	×	×
Documentation and tutorials	●	●	●	◑	●	×	●	◑	●	●	●	×	◑	×	×	●	◑	◑
High-level API	×	●	●	–	●	×	●	◑	●	●	●	●	●	◑	●	●	●	◑
Ability to extend the framework	●	●	◑	–	●	–	–	◑	●	●	●	●	◑	×	–	●	●	◑
Actively maintained	●	●	●	●	●	×	●	●	●	●	●	◑	●	×	●	●	●	●

well as different federated scenarios such as HFL, VFL, and FTL. Note that for both VFL and FTL, the most widely used datasets are artificially partitioned but not inherently federated. Besides, the table includes the total number of instances in the dataset and the predefined number of clients in the federated scenario (in those cases where the dataset distribution is inherently federated). These datasets have been used in a substantial number of studies (whose references are included in the last column of the table).

B. State-of-the-Art Frameworks for Federated Learning

When developing experiments for a federated scenario, there are multiple frameworks designed for doing it. We have searched for state-of-the-art open-source frameworks available and selected some important aspects of FL to check whether these frameworks fulfill each aspect or not. Table II shows the reviewed frameworks and whether they fulfill those important aspects in FL[8]. This table may help the users select the framework to use for their experiments with FL. Due to space limitations because of the large number of frameworks, the names of the different frameworks have been shortened in the table. The frameworks reviewed are: PySyft (PyS), TensorFlow Federated TFF, FATE (FAT), PaddleFL (Pad), Flower (Flo), Xaynet (Xay), IBM FL (IBM), Substra (Sub), OpenFL (OFL), FedML (FML), FedJax (FJx), Backdoors 101 (101), FedLab (FLb), SimFL (SFL), easyFL (EFL), TorchFL (TFL), APPFL (AFL), NVFlare (NVF).

a) Degrees of Compliance of Each Framework: Three degrees of compliance have been considered depending on whether an aspect is supported by a framework or not. The green dots indicate that this aspect is fully supported by the framework. The orange ones indicate that the aspect is partially covered in the framework, i.e., it covers some cases but not all. Finally, the red dots mean that this aspect is not supported in the framework. Additionally, the grey dots indicate that we were unable to determine exactly whether the framework supports an aspect.

b) FL Aspects Covered by Each Framework: The table is horizontally partitioned into four groups. First, the main aspects of FL are checked, indicating whether or not the frameworks support the execution of HFL, VFL, or FTL algorithms, if they support common ML frameworks such as TensorFlow, PyTorch, or Scikit-Learn, if they support IID and non-IID data sampling, and if they include a wide range of federated aggregation mechanisms that have been proposed in the literature. Finally, the fourth group checks other advanced properties of the frameworks, such as the interpretability of their models, if they support model personalization on the client side, if they provide comprehensive documentation or a high-level API, their ability to extend the framework with new properties customized by the user, and if they are actively maintained.

[8] The information in the table is updated as of January 2023.

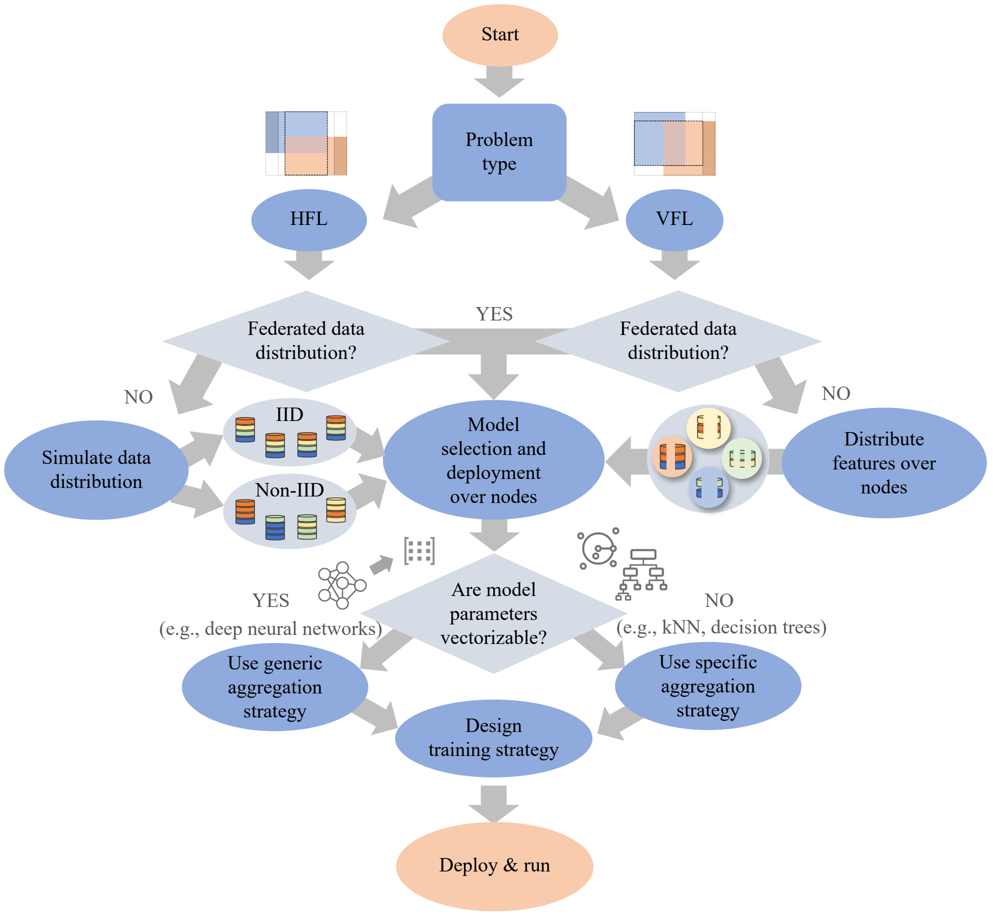
Start
Problem type
HFL
VFL
Federated data distribution?
YES
Federated data distribution?
NO
NO
Simulate data distribution
IID
Non-IID
Model selection and deployment over nodes
Distribute features over nodes
YES
(e.g., deep neural networks)
Are model parameters vectorizable?
NO
(e.g., kNN, decision trees)
Use generic aggregation strategy
Use specific aggregation strategy
Design training strategy
Deploy & run

Fig. 6. Workflow for designing a FL scenario. Read from top to bottom, it begins with the choice of the problem type, HFL or VFL, and continues choosing the federated data distribution until the training strategy is designed, then the FL scenario is ready to be deployed and run.

V. Methodology for Designing a Simulated Federated Learning Scenario

In previous sections, the foundations of FL and the characteristics of existing datasets and frameworks have been reviewed and analyzed. Provided that FL experimentation has a major handicap in that there are no truly federated datasets, i.e., datasets that are hosted on different devices, in this section we provide methodological guidelines for simulating FL scenarios. We stress that, what we do are simulations, in which during experimentation we simulate clients that do not access each other's data. In some situations, this simulation may be closer to reality in the sense that the data may have been collected from different sources, thus distributing it by identifying each of these sources with a client in the federated schema.

Fig. 6 shows a workflow with the main steps to follow to design a FL scenario and experiments. We pay attention to various aspects: the problem to face and its data distribution, the model selection, the training strategy, and evaluation methodologies.

a) Federated Learning Scenario, Problem and Data Distribution: The first step is to analyze the problem at hand, discriminating mainly between HFL or VFL problems (see Section III-C). Not all FL frameworks can deal with every kind of scenario (see Table I). Therefore, the target scenario has a major influence on the framework to be used. If we deal with an HFL problem, frameworks such as TFF, Flower, or PySyft can be used, while FATE or PaddleFL frameworks can be adopted to tackle VFL scenarios. The software choice is entirely at the user's discretion, according to its experience with similar frameworks, the specifications of the problem, and other concerning factors. In the following sections, we discuss and analyse some of them in more detail.

Regardless of the scenario, we should also check whether the data is inherently federated or not. In general terms, those datasets that can be naturally divided into pieces, each belonging to different clients according to their features or data distribution, and can also be considered as inherently federated datasets. For example, the FEMNIST dataset includes alphanumeric characters written by more than 3500 different users; therefore, the data can be distributed so that each client retains the characters written by a single user, as it would occur be in a real-world scenario where characters are collected by a personal device. Likewise, in the Sentiment140 dataset each tweet is annotated with the user who wrote it, thus each client can be identified with a user, giving rise to an inherently federated dataset. Conversely, datasets such as Credit2 have no natural or intrinsic division that makes their partitioning among different clients relevant to be solved via FL. However, such partitions can be made for simulation purposes.

It should also be clarified that datasets that *fully* comply with the FL framework (i.e., they are already distributed among different parties so that the data is not visible outside each of them) are not commonly found for simulation, since the data needs to generally be hosted in a single site for use and distribution. In any of the cases, if the data is inherently federated, it can be directly distributed among clients, enabling the next step of our methodology: model selection.

In the HFL scenario, if the data is not inherently federated, we must split the data by samples. There are two coarse distributions: IID and non-IID (see Section III-A). The former (IID) distributes the data evenly among clients, which can be a feasible scenario in a minority of real-world use cases. The latter (non-IID), however, implies a distribution where each client retains different quantities of data (quantity skew), having various distributions in their feature space (feature distribution skew), having information about entirely different classes in the output variables (label distribution skew) [66], [67], or even a combination of them. In general, non-IID distributions are frequently encountered in real-world problems. In any case, the data distribution should be selected depending on the simulation to be performed. It should be noted that to simulate an IID scenario, just a random partition of the dataset must be performed. However, simulations of non-IID scenarios are instead driven by the approach followed to allocate certain data instances to each federated node, often made according to a Dirichlet distribution, where the imbalance level of each dataset is controlled by a parameter β [66]. In any case, once data is partitioned, it is also distributed among the clients, so that each of them receives different data samples that will be no longer available outside each client.

If facing a VFL problem where the data is not inherently federated, the data is split by features. At this point we note that, in the case of VFL, the non-IID nature of the data arises from the non-overlapping feature distribution among the different clients and the number or types of features available at each client [67]. Once partitioned, each client receives the same set of data samples but with a different set of features, where the class label is only held by one of them.

b) Model Selection: Once the data is distributed among the clients, the next step would be the selection and deployment of the model. Depending on the task, we can choose between supervised learning models, such as decision tree models; unsupervised learning models such as clustering models; or any other traditional ML model or deep learning model. As in any traditional learning approach, the model to be used in the federated scenario will much depend on the problem to be faced. For example, convolutional architectures based on convolutional neural networks (CNN) have been proven to work well for image classification problems [68], recurrent architectures based on recurrent neural networks (RNN) have been widely used for text classification, SA, or temporal prediction problems [69], [70], and other traditional machine learning approaches such as decision trees or support vector machines have been applied for years to simpler problems with tabular data, such as bank credit risk prediction [71]. When there is not a great amount of data available or there is a non-IID data distribution among the clients, it can be interesting to use pre-trained models and fine-tune them for the task to be solved [72], [73]. As a result, we can train large models and take advantage of their knowledge, getting better models and requiring fewer resources than training them from scratch [74].

c) Aggregation Strategy: All the aforementioned models have different adaptations to the FL setting, driven by the aggregation step, where local models are combined. If the model can be expressed as an array of parameters (i.e., deep learning models), a generic and simple aggregator like FedAvg can be chosen. FedAvg performs the weighted average of the collection of local client models, by computing the weighted average of the arrays of parameters, where each array of parameters corresponds to a local client model and the weight factor is the proportion of the client data volume to the total data volume. Note that, there exists more complex variants of FedAvg, designed to alleviate the difficulties associated to non-IID-ness, Byzantine attacks or the lack of personalization [75]. Conversely, if the chosen model cannot be expressed as an array of parameters (such as decision trees or k-means clustering), an advanced and specific aggregator should be selected or implemented ad-hoc to collaboratively learn such a model in a FL scenario.

Further along this methodological step, in general, the same type of model is selected for all clients, and a first version of the model is distributed among them. However, other approaches enable each client to use a model with a different structure, which is then aggregated by the central server usually following an ensemble approach [76], [77]. While the former makes it generally easier to aggregate local models and speeds up the convergence of the global model, the latter overcomes the problem where small participating devices may not have equal access to computing resources, potentially not being capable of running complex models.

d) Training and Evaluation Strategy: The last remaining step of our methodological guidelines is to design the training strategy, which involves several aspects such as defining the number of training rounds (i.e., the number of communications between server and clients to converge to the joint model), the number of clients participating in each round (in some cases, only a fraction of the clients participate in each round), the optimizer, or any other parameter of specific to the model, as well as the evaluation method of the trained model. This evaluation should be considered at two levels: *client level* and *global level*. The former considers a set of test metrics designed to evaluate the performance of a FL model according to the specific needs and target specifications of each client. Given the potentially non-IID distribution of each client, their evaluation is not enough to assess the performance of the whole FL model. Nevertheless, these values in combination with other statistical properties can be devised to select certain clients to aggregate in each step [78]. While the *global level* evaluation also considers a set of test metrics to evaluate the performance of a FL model from a wider perspective, it considers testing the scalability of the chosen FL model, i.e., the FL system performance. Consequentially, in addition to the FL benchmarks presented in LEAF [35], a FL practitioner must be encouraged to benchmark FL system performance [79]. It is common to assume that all clients unconditionally want to participate, meaning that there is no reward for each participant other than the globally trained model. Ideally, each client joining a FL process contains a unique and private local dataset and expects to be fairly rewarded with a better model [3], [9]. However, realistically, potential clients have the agency to decide whether to join or remain in the FL process based on how their individual collaborations will be rewarded. In this sense, from a game theory perspective it is possible to provide each client with a fair payoff [80].

VI. Comparative Study of FL and Non-FL Scenarios: Should FL be Always Considered?

A first question to be addressed is whether FL provides any benefits with respect to isolated local training (non-FL) when the data is distributed among several clients. This section elaborates on this by presenting a performance comparison between FL and non-FL strategies when the data is distributed among several clients, with the aim to show the benefits of collaboratively training a model with FL.

To compare FL and non-FL scenarios, we consider three cases: 1) A single model trained using all the data in a centralized manner, which will be referred to as *centralized model*; 2) A number of models, each one trained by each of the clients using only their local data and not sharing any information with the other parties, referred to as *local models*; and 3) A single shared model trained among all clients but without exposing their data, i.e., a *FL model*. The centralized model is usually considered as an upper-bound theoretical limit for the performance of the federated model, since a better model is usually expected if all the data is gathered. However, note that this model is created only for comparison purposes so implementing it in real-world federated scenarios is not possible, since the data cannot leave each client to create a single centralized dataset. Conversely, it is expected that the FL model performs better than the local models by themselves, since it should benefit from the collaborative training.

For the experiments, we use the MNIST dataset. As noted in Section III-A, we consider both IID and non-IID partitions. In this use case, and to make a fair comparison, we use only one framework, i.e., TFF for the federated model and TensorFlow for the baselines. In this way, we ensure that there are no differences in the architecture or internal implementation of the models. Besides, the model is composed of CNN layers, given their success in image classification tasks [68]. All models are evaluated using the same set of evaluation data which is as well IID or non-IID, according to the experiment at hand.

For the experimental setup in the FL scenario, 10 clients have been considered, whose models have been collaboratively trained for a total of 10 rounds, with 5 local training epochs run before communicating with the central model. The centralized and local models are trained for a total of 50 epochs. Results of the experiments reported in what follows are averaged among 10 executions using different seeds, ensuring the statistical consistency of the comparison. The experiments were executed in a computer with Ubuntu 18 OS, an Intel Xeon E5-2698 CPU and 512GB RAM.

a) IID Scenario: We first analyse the results when the data is IID (see Table III). The table includes the training and testing loss and accuracy metrics, as well as the runtime required to train and test the models. As expected, the centralized model performs slightly better than the federated one. The benefits of collaboratively training a shared model among all clients are clearly stated when comparing the performance of the FL model versus the local ones. Note that the runtime in the federated scenario is higher due to communication overhead. Although the FL model performs better than the local ones when the data is IID, in most real-world scenarios the data distribution is non-IID.

TABLE III
Comparison Among the Federated Model Versus Baselines With IID Data

	Centralized model	Local models (IID)	FL model (IID)
Train loss	0.026	0.064	0.043
Train accuracy	0.992	0.983	0.989
Test loss	0.047	0.113	0.090
Test accuracy	0.985	0.966	0.973
Runtime (s)	140.7	16.1	749.5

b) Non-IID Scenario: In Table IV a comparison among the local and FL models using non-IID data is performed. Note that, in this case, it does not make sense to compare against a centralized model, since we cannot refer to non-IID data if it is not distributed. For such experiments with non-IID data, two different configurations have been considered: 10 clients and 10 FL rounds, as in the previous case, and 20 clients with 20 learning rounds. On average, each client in the MNIST data has less than 100 images, which is insufficient to build a good model.

TABLE IV
Comparison Between the Federated Model Versus Local Ones With Non-IID Partitions

		Local models (non-IID)	FL model (non-IID)
10 clients 10 rounds	Train loss	2.173	0.658
	Train accuracy	0.311	0.851
	Test loss	2.243	1.063
	Test accuracy	0.192	0.640
	Runtime (s)	2.4	27.9
20 clients 20 rounds	Train loss	4.119	0.257
	Train accuracy	0.869	0.928
	Test loss	4.371	0.451
	Test accuracy	0.527	0.852
	Runtime (s)	4.6	75.9

When running the experiment with 10 clients and 10 learning rounds, the local models achieve poor performance, with a test accuracy lower than 0.2. In the same scenario, the FL model achieves acceptable performance, but it is still possible that more data or epochs may be needed to get a better model, given the limited amount of data that each client has in this case. Conversely, when training the models with more clients and for a higher number of epochs, we observe that the FL

TABLE V
SUMMARY OF USE CASES ACCORDING TO THE METHODOLOGY TO DESIGN A FL SCENARIO

Use case	Problem type	Framework	Naturally federated data	Data distribution	Aggregator	Model
UC1	HFL	TFF, Flower, FATE	Yes/No	non-IID/IID	Generic	CNN & Dense deep neural network
UC2	HFL	TFF, Flower	No	IID	Generic	Deep neural network
UC3	VFL	FATE	No	non-IID	Specific	Decision trees
UC4	HFL	TFF	Yes/No	non-IID/IID	Generic	CNN with DP
UC5	HFL	TFF	No	IID	Specific	k-means
UC6	VFL	FATE	No	non-IID	Specific	k-means

model obtains much better performance (with an accuracy over 0.85 in test), and still improves the performance of the local models. Thus, the model performance varies depending on the number of clients or data available, which might be fixed by the environment, and by the number of training rounds performed, which may be adjusted; in any case, the benefits of FL versus isolated local models are clearly stated.

c) Final Conclusions: Considering the above experimental results, we conclude that:

- If we face a scenario where multiple local clients aim to build a model, but they cannot (or do not wish to) share their data, and it is recommended to build a model collaboratively using FL. This recommendation is even more important when the data is non-IID, which is the most common case in real-world scenarios, not only because better results can be obtained (as Tables III and IV clearly expose), but also because the model can be enriched with more knowledge.
- When collaboratively building a model, the average improvement in testing accuracy among the different evaluated scenarios when compared to isolated local model training is around 98%, which is 230% higher than in non-IID scenarios with 10 clients and 10 learning rounds.

VII. FEDERATED LEARNING FOR PRACTITIONERS: EXEMPLARY USE CASES

We proceed by presenting six exemplary use cases to study, from a practical perspective, FL scenarios, several ML problems and demonstrate the process to solving them with three selected frameworks. In all cases, the workflow proposed in Section V is followed. Several scenarios are considered according to the diversity of data and models, as follows.

First, Section VII-A demonstrate how to perform an image classification using deep learning models in a HFL setting (Use Case 1 (UC1)). Next, Section VII-B delves into a SA classification problem using deep learning models in HFL (Use Case 2 (UC2)). Section VII-C showcases how to perform VFL classification using decision trees, with a traditional ML algorithm interpretable by design (Use Case 3 (UC3)). Section VII-D introduces a method to train FL models with DP (Use Case 4 (UC4)). Finally, Section VII-E illustrates the training of unsupervised clustering models in HFL (Use Case 5 (UC5)), whereas VII-F concludes the section by showing the training of clustering models in a VFL setting (Use Case 6 (UC6)).

Before describing them, we pay attention to the three selected frameworks that we consider relevant for further analysis: TFF, Flower and FATE.

- *TFF:* an open-source ML platform that extends the widely known TensorFlow to perform federated ML. TensorFlow is a reference platform in deep learning, which has great popularity and support from the community and its developers. Therefore, users who are familiar with TensorFlow and want to start developing models for FL will find it much easier to do so in TFF than in any other platform.
- *Flower:* Flower has gained popularity recently, given that it is much simpler and more familiar to use than other platforms. Although it was launched more recently than TFF, it has a community supporting it and it is actively maintained. The general characteristics of both TFF and Flower are very similar, but TFF partially supports the creation of non-IID data partitions and offers slightly more support for DP. However, it should also be noted that, while TFF allows definitions of any deep learning model that could be created in TensorFlow and Keras, Flower also allows using models written in PyTorch, which is another major tool for deep learning.
- *FATE:* Finally, since neither TFF nor Flower consider VFL scenarios within their frameworks, we have selected FATE, which does. Compared to the rest of the frameworks that consider VFL, FATE is the one that offers the most options and has the most complete and clear documentation. As subsequently presented, the implementation of FATE's code is not based on any other prior framework (as TFF and Flower were on TensorFlow, Keras or PyTorch), but defines its procedure based on a pipeline where different components are added, compiled and ran.

Unless otherwise indicated in any of the use cases, the same experimental setup as in Section VI is used.

The most important code excerpts required to complete the use cases are presented in the paper. The whole use cases are available as Python notebooks on the following website: https://github.com/ari-dasci/S-TutorialFL. Notebooks are fully documented so that the user can run and reproduce the results, or even modify them in a way that maximizes the learning and understanding of the different frameworks and the FL workflow itself.

A summary of the use cases is presented in Table V, according to the different aspects described in the methodology for designing a FL scenario, such as the problem type, being either HFL or VFL; if the data used has an inherent federated distribution; whether the dataset is split as IID or non-IID among the clients; which kind of aggregator (generic or specific) is used; and which kind of model is used to solve the task at hand.

A. UC1: Image Classification Using Deep Learning in HFL

In recent years, different deep learning techniques have been used to solve a plethora of problems. Besides, image classification is one of the most common challenges in ML. In this case, we demonstrate the process of performing image classification using the MNIST dataset.

Following the methodology described in Fig. 6:

1) Problem Type: The first step is to determine which problem we tackle. In this case, the problem is reported as HFL, i.e., each client holds different data patterns following the same input feature set. As the three selected frameworks (TFF, Flower, and FATE) can deal with HFL problems, all of them are used.

2) Federated Distribution: In this use case, we deal with two different data distributions: i) the intrinsic non-IID partition is considered, so that each client receives digits written by a unique user and ii) the whole dataset is divided into simulated IID partitions, so the data in each client follows a similar distribution.

3) Model Selection: Two different deep learning networks are used to solve the problem: one including a CNN layer, which is widely used for image classification [68], and another using only densely connected neural layers, which is less complex.

4) Aggregation Strategy: As deep network parameters are vectorizable, a generic aggregator such as FedAvg is used.

5) Training and Evaluation Strategy: Finally, to be as close as possible to real-world settings where there are usually no shared data fragments (i.e., no global test set is available), the evaluation is carried out by assessing the model over the local test data sets at each client, reporting the average value of the performance metrics. Such test sets will also have either an IID or non-IID distribution, according to the type of partitioning that was also used for training.

Firstly, we demonstrate the process of solving the use case with TFF. In TFF, there are two main ways to load the data: i) loading those datasets that TFF provides specifically designed for their platform, which are already federated as non-IID partitions, or ii) loading any other dataset and distributing it among the clients. In Listing 1, the non-IID partitioned MNIST data is loaded from TFF (each instance is already assigned to a client); while in Listing 2, the traditional MNIST dataset is loaded from *tensorflow datasets* package and transformed into a dataframe with a random client id, so that the IID partition can be created later. After loading the data as presented, it must be preprocessed to match the model required input structure and distributed among the different clients. Note that in this case, we present the two possibilities presented in the HFL scenario in Section V: either having an inherently federated dataset (Listing 1) or simulating the partitioning and distribution of the data among clients (Listing 2).

To define the models, a method that returns a Keras model must be created. In Listing 3 we show how to create a CNN; a similar process should be followed to create any other network architecture. For this specific problem, the data is first reshaped, so it is represented as a 28×28 pixel matrix. Later, a two-dimensional CNN layer with 32 filters, a kernel size of

```
mnist_train, mnist_test =
 ↪  emnist.load_data(only_digits=True)
```

Listing 1. UC1: image classification using deep learning in HFL in TFF. Loading TFF's MNIST dataset.

```
# Load MNIST from tfds (train partition)
mnist_train = tfds.load('mnist', split='train')

# Transform the data to a dataframe
mnist_train_df = tfds.as_dataframe(mnist_train)

# Random list of ids to add to the dataframe
ids_train = ... # Random values in [0, NCLIENTS-1]
mnist_train_df['id'] = ids_train
```

Listing 2. UC1: image classification using deep learning in HFL in TFF. Loading MNIST from other sources.

```
def create_keras_CNN():
  model = models.Sequential([
    layers.Reshape((28, 28, 1), input_shape=(28, 28)),
    layers.Conv2D(32, kernel_size=(5, 5),
     ↪  activation="relu", padding="same", strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2,
     ↪  padding='valid'),
    layers.Flatten(),
    layers.Dense(10, activation="softmax"),
  ])
  return model

def model_fn():
  keras_model = create_keras_CNN()

  return tff.learning.from_keras_model(
    keras_model,
    input_spec=train_data[0].element_spec,
    loss=losses.SparseCategoricalCrossentropy(),
    metrics=[metrics.SparseCategoricalAccuracy()]
  )
```

Listing 3. UC1: image classification using deep learning in HFL in TFF. Creating a CNN model.

5×5 pixels, and a ReLU activation function is created, which ends with a dense layer with 10 output units, one for each output class. In addition, a method that creates the model for the federated scenario, including not only the network to use but also the input specification, loss, and evaluation metrics should be created. In this case, categorical cross-entropy loss and accuracy are used, given the output nature of the problem. A more in-depth analysis on CNNs for image classification can be found in [68].

To train in TFF, a training strategy should be first defined. As the model can be expressed as an array of parameters, a generic aggregator as FedAvg is employed (see Section V). Since an unweighted FedAvg method is used in this case, each client has the same importance in the learning process. Nevertheless, a weighted FedAvg approach could also be used, where the contribution of each client to the global model is biased by the amount of data available locally. Besides, Adam is used as the optimizer; note that different learning rates may be set for the client and the server. Once the process is initialized, the model is collaboratively trained for several rounds. The choice of the number of rounds can be driven by the specific requirements of the problem, such as runtime constraints or the convergence and performance achieved by the model. This process is presented in Listing 4.

```
# Define training process
training_process = build_unweighted_fed_avg(
  model_fn,
  client_optimizer_fn=lambda:
   ↪  optimizers.Adam(learning_rate=0.001),
  server_optimizer_fn=lambda:
   ↪  optimizers.Adam(learning_rate=0.01)
)

train_state = training_process.initialize()
for round_num in range(1, NROUNDS+1):
  result = training_process.next(train_state, train_data)
```

Listing 4. UC1 of image classification using deep learning in HFL in TFF. Training in the federated scenario.

In contrast, Flower provides neither non-IID partitioned datasets nor mechanisms to create such partitions. As we do not have an inherently federated dataset, the MNIST dataset is loaded from PyTorch's torchvision module, and later it is IID partitioned among the different clients, following a simple uniform random data distribution (Listing 5).

```
# Load train from torchvision and transform to tensor
mnist_train = MNIST("./dataset", train=True,
 ↪  transform=transforms.ToTensor())

# Split into NCLIENTS partitions to simulate the
 ↪  individual datasets
train_lengths = [len(mnist_train) // NCLIENTS] * NCLIENTS
train_splits = random_split(mnist_train, train_lengths)

# Create DataLoaders for each client
train_data = []
for i in range(NCLIENTS):
    train_data.append(DataLoader(train_splits[i]))
```

Listing 5. UC1: image classification using deep learning in HFL in Flower. Loading the MNIST dataset.

As previously mentioned, Flower allows using either Keras or PyTorch models. To show an alternative to the previous case, in Flower we use PyTorch models. In Listing 6, an example method to create a CNN model is presented. We note that although the code snippet is different from that of TFF, the structure of the network is the same: a two-dimensional CNN layer with 32 filters and kernel size of 5×5 pixels, ReLU activation function, and a final dense layer with 10 output units. A similar process should be followed to create other architectures.

```
class CNN_Net(nn.Module):
   def __init__(self) -> None:
      super(CNN_Net, self).__init__()
      self.cnn1 = nn.Conv2d(in_channels=1,
       ↪  out_channels=32, kernel_size=5, stride=1,
       ↪  padding=2)
      self.relu1 = nn.ReLU()
      self.maxpool1 = nn.MaxPool2d(kernel_size=2)
      self.fc1 = nn.Linear(32 * 14 * 14, 10)

   def forward(self, x: torch.Tensor) -> torch.Tensor:
      out = self.cnn1(x)
      out = self.relu1(out)
      out = self.maxpool1(out)
      out = self.fc1(out.view(out.size(0), -1))
      return out
```

Listing 6. UC1: image classification using deep learning in HFL in Flower. Creating a CNN model.

In Flower, the user must define the train and test methods (as if for a centralized scenario). To adapt such methods to the federated scenario, first a FlowerClient class is defined, so each client in the federated simulation is an instance of such class and has its data and copy of the network. To allow Flower to create client instances and simulate the model distribution (see Section V), it is necessary to create a method such as the one shown in Listing 7.

```
def client_fn(cid: str) -> FlowerClient:
   # Load model
   net = CNN_Net().to(DEVICE)

   # Note: each client gets a different train/test data
   trainloader = train_data[int(cid)]
   testloader = test_data[int(cid)]

   # an FLower client represents a single participant
   return FlowerClient(net, trainloader, testloader)
```

Listing 7. UC1: image classification using deep learning in HFL in Flower. Method to create client instances.

Flower's FedAvg implementation is used in this example to train the model in the federated scenario. Note that more metrics can be reported if the user defines their strategy (as shown in the notebook). To start the simulation, we must indicate the method to create the clients, how many clients are involved, the configuration for the server (including the number of rounds), and the strategy to follow, as presented in Listing 8.

```
fl_sim = fl.simulation.start_simulation(
   client_fn=client_fn,
   num_clients=NCLIENTS,
   config=fl.server.ServerConfig(num_rounds=NROUNDS),
   strategy=fl.server.strategy.FedAvg(),
)
```

Listing 8. UC1: image classification using deep learning in HFL in Flower. Training in the federated scenario.

To conclude with the implementation of this use case, we show how to perform image classification in FATE. The first thing to note about FATE is that, although running on a single machine, the user must set up different clients. For that purpose, each client is assigned a different id to simulate the real-world scenario, and they are assigned as either *guest* or *host* roles. In this case, many hosts can be configured, we define the first client as the guest and the rest as hosts.

In addition to those roles, there is also an *arbiter* role, which orchestrates the learning process. FATE is based on the creation of a pipeline, so once created, the id of each participant is indicated (Listing 9).

FATE does not provide datasets, but they can be loaded from external sources. For such a purpose, the user must configure a Reader component, as shown in Listing 10, where the data path is specified. In this case, the dataset is not directly loaded from the external source within the code, but MNIST images have been downloaded and placed in a directory divided into folders for each client and train/test partition, so that FATE can read them.

It should also be noted that, in the HFL scenario, FATE only allows using networks comprising PyTorch's dense layers. Thus, in this case, Listing 11 demonstrates the implementation of only the dense architecture in FATE, but not the one comprising CNN layers. This network is simpler than the

```
guest = 10000
hosts = [10001, 10002, 10003, 10004]
arbiter = 9999

# Initialize the pipeline
pipeline = PipeLine()
# Set participants information
pipeline.set_roles(
  guest=guest,
  host=hosts,
  arbiter=arbiter
)
```

Listing 9. UC1: image classification using deep learning in HFL in FATE. Create a pipeline and set roles.

```
# Create table with data for client 0
train_datas[0] = {
  "name": "mnist_train_0",
  "namespace": "experiment"
}

# Bind the data into the pipeline
pipeline.bind_table(name=train_datas[0],
 ↪  path='/path/to/data/c0_train')

# Create Reader for training data
reader_train = Reader(name="reader_train")

# Set the data of each client
reader_train.get_party_instance(role='guest',
 ↪  party_id=guest).component_param(table=train_datas[0])

# Repeat the same process for the rest of clients
```

Listing 10. UC1: image classification using deep learning in HFL in FATE. Creating a Reader component.

```
model = t.nn.Sequential(
   t.nn.Linear(784, 32),
   t.nn.ReLU(),
   t.nn.Linear(32, 10),
   t.nn.Softmax(dim=1)
)
```

Listing 11. UC1: image classification using deep learning in HFL in FATE. Creating a dense network model.

previous ones, considering the input as a sequence of $28 \times 28 = 784$ input units, which are transformed into 32 hidden layer units with a linear or dense layer, uses a ReLU activation function, and ends again with 10 output units, one for each different class.

To define the training process, the HomoNN class is used (Listing 12). It includes the definition of the model to use, loss, optimizer, dataset, and training strategy. The parameters are the same as those used in previous cases.

After creating each of the components (such as the nn_0 in Listing 12), they must be added to the pipeline as shown in Listing 13. The training process is started once the pipeline is compiled and called the fit() method. The creation of such pipeline is further described in the corresponding notebook.

Once we describe how to implement the solution to the use case in each of the platforms, we show some results obtained from their execution. In Table VI we report the training and testing loss and accuracy obtained by the models (both the convolutional and dense architectures; except for FATE, which does not support CNNs), as well as the required runtime to perform the federated training and evaluation. Besides, note that FATE does not report loss value for testing data. To

```
nn_0 = HomoNN(
   name='nn_0',
   model=model,
   loss=t.nn.CrossEntropyLoss(),
   optimizer=t.optim.Adam(model.parameters(), lr=0.001),
   dataset=dataset_param,
   trainer=TrainerParam(
     trainer_name='fedavg_trainer',
     epochs=NEPOCHS * NROUNDS,
     aggregate_every_n_epoch=NEPOCHS,
     batch_size=BATCH_SIZE
   )
)
```

Listing 12. UC1: image classification using deep learning in HFL in FATE. Configure the federated training.

```
pipeline = PipeLine()

# Add all the corresponding components
pipeline.add_component(reader_train)
pipeline.add_component(
    nn_0,
    data=Data(train_data=reader_train.output.data)
)
# ...

# Compile the pipeline and train
pipeline.compile()
pipeline.fit()
```

Listing 13. UC1: image classification using deep learning in HFL in FATE. Configure pipeline and train.

TABLE VI
RESULTS OF UC1 WITH THE IMAGE CLASSIFICATION TASK USING DEEP LEARNING IN HFL FOR THE THREE FRAMEWORKS AND TWO NETWORK ARCHITECTURES. N/A VALUES INDICATE THAT NEITHER THE CNN COULD BE BUILT IN FATE NOR COULD THE TESTING LOSS BE REPORTED IN FATE

		TFF	Flower	FATE
Dense	Train loss	0.192	0.002	1.489
	Train accuracy	0.947	0.991	0.968
	Test loss	0.276	0.006	N/A
	Test accuracy	0.922	0.965	0.947
	Runtime (s)	53.6	198.5	1516.7
CNN	Train loss	0.043	0.000	N/A
	Train accuracy	0.989	1.000	N/A
	Test loss	0.090	0.002	N/A
	Test accuracy	0.973	0.987	N/A
	Runtime (s)	749.5	337.1	N/A

date, FATE was unable to deal with more than 4 hosts, so FATE's experiments were executed with only 5 clients instead of 10.

It can be first observed that the accuracy results between the different frameworks are similar, and their differences may be given by implementation differences in the models. Moreover, results corresponding to TFF and Flower show that, as expected, the convolutional architecture performs better for image recognition purposes compared to a simpler architecture. In fact, it should be noted that only 10 global learning rounds have been used for the experiments, but better results may be obtained if it were executed for a higher number of rounds. According to execution times, there is no clear difference between TFF and Flower, while TFF's dense network runs faster than Flower's, Flower is faster at executing the CNN. However, it is clearly shown that FATE is much slower than the rest of the frameworks. Therefore, according to the presented results, Flower might be the best option using a convolutional architecture for image classification, obtaining the best performance much faster than TFF. In contrast, FATE does not seem to be a suitable option for HFL image classification purposes. However, and as previously stated, such a selection might also be biased by other factors; for example, an expert user of TensorFlow might choose TFF over Flower if the runtime is not of crucial importance, since the learning curve for creating the FL models will be much lower.

B. UC2: Sentiment Analysis Using Deep Learning in HFL

NLP problems have also topped the literature topics in ML, even more with the growth of deep learning models in the last decade. In this use case we show how to perform SA with the Sentiment140 dataset, using a pretrained network instead of building it from scratch.

Following the proposed methodology:

1) Problem Type: The first step is to check the problem under consideration. As in the previous case, the problem in UC2 is also HFL, where each client holds different tweets that are private and cannot be accessed by other clients. Having analysed the limitations of FATE according to the network architectures (only dense layers are allowed), in this use case, we only consider TFF and Flower frameworks to solve the problem.

2) Federated Distribution: For this use case, a simulation of an IID partition among clients is made, and a pretrained deep learning network is used, which is specifically designed for text processing tasks. The weights of this pretrained model are publicly available[9].

3) Model Selection: Fine-tuning a model that has been specifically proposed for text processing tasks is a promising starting point, as we rely on a model that has been successfully tested on similar modelling problems. The fact that this model has already been pretrained over similar data makes the convergence much faster and therefore better performance levels can be reached within shorter execution times, which becomes even more important in FL environments due to the need for reducing the communication overhead.

4) Aggregation Strategy: As would happen with any other deep learning network, the parameters are vectorizable, so a generic aggregator is used.

5) Training and Evaluation Strategy: As in the previous use case, and as would be in most real-world cases, the model is evaluated using local and private test datasets, where each client reports the metric values over their own data.

First, we analyse how to solve the use case in TFF, highlighting the main differences with the previous use case. In this case, the data is downloaded from *tfds*, as in Listing 2, the percentage of data used both at training and testing phases is also provided (see Listing 14). Note that the Sentiment140 dataset has 1.6 million instances, a fraction of it is used so that the experiments can be executed in a reasonable time. Then,

[9] https://tfhub.dev/google/nnlm-en-dim128-with-normalization/2

```
# Load train and test partitions of sentiment140 from tfds
sent140 = tfds.load('sentiment140', split=['train[:1%]',
 ↪  'test[:10%]'])
sent140_train, sent140_test = sent140[0], sent140[1]

# Transform train to dataframe and keep desired columns
sent140_train_df =
 ↪  tfds.as_dataframe(sent140_train)[['text', 'polarity']]
sent140_train_df['text'] =
 ↪  sent140_train_df['text'].apply(lambda x:
 ↪  text_processing(x))

# Create a random list of ids and set to the dataframe
ids_train = [i for i in range(NCLIENTS) for _ in
 ↪  range(len(sent140_train)//NCLIENTS)]
random.Random(seed).shuffle(ids_train)
sent140_train_df['id'] = ids_train

# Do the same process for the test data
```

Listing 14. UC2: SA using deep learning in HFL in TFF. Loading and processing and distributing a percentage of Sentiment140 dataset as IID data.

the columns according to the tweet text and its polarity are selected and the data is transformed to a dataframe, so it can be later converted to a dataset format required by TFF. The tweets should also be processed using the text_processing() method that removes punctuation marks and converts them to lowercase, among others, as seen in the corresponding notebook. In order to make IID partitions, a random list of ids is created so that the instances are randomly distributed among the clients. Thus, each client has access to a different portion of the training data.

The Sentiment140 dataset has three classes by default: 0 (negative), 2 (neutral), and 4 (positive). In previous use case we already dealt with a multi-class problem, so in this case we are transforming it to a binary one. For that purpose, we remove the neutral instances, so the aim is to differentiate between negatively and positively polarized tweets. Such a process is made as in Listing 15, where tweets with a neutral polarity are removed. Tweets with a positive polarity (i.e., a value of 4) are given class value of 1, whereas negatively polarized tweets are assigned to class 0.

```
def delete_neutral_ops(df):
    df = df.loc[df['polarity']!=2]
    df['polarity'] = df['polarity'].replace(4, 1)
    return df

sent140_train_df = delete_neutral_ops(sent140_train_df)
sent140_test_df = delete_neutral_ops(sent140_test_df)
```

Listing 15. UC2: SA using deep learning in HFL in TFF. Transform the multi-class problem to a binary one.

The model can be defined as in Listing 16. In this case, the create_keras_model() method creates a network that is based on a pretrained model, whose weights are frozen during training. If the user would like to also fine-tune it, the parameter should be changed to True, however it will make the training phase last longer. Besides adding such pretrained layers, we include an extra layer with 16 neurons and ReLU activation function, as well as a final layer with only one output unit. Since this is a binary classification problem, a single output unit is enough to predict the polarity of the tweet. Despite using the above model, any other model from *tfhub* or other repositories might be used. Other architecture created by the

```
def create_keras_model():
  # Load pretrained model from tfhub
  hub_layer = hub.KerasLayer(
    "https://tfhub.dev/google/...",
    input_shape=[],
    dtype=tf.string,
    trainable=False
  )

  # Create model with fixed pretrained model
  # Also add some dense layers
  model = tf.keras.Sequential()
  model.add(hub_layer)
  model.add(tf.keras.layers.Dense(16, activation='relu'))
  model.add(tf.keras.layers.Dense(1))
  return model


def model_fn():
  keras_model = create_keras_model()

  return tff.learning.models.from_keras_model(
    keras_model,
    input_spec=train_data[0].element_spec,
    loss=losses.BinaryCrossentropy(),
    metrics=[metrics.BinaryAccuracy()]
  )
```

Listing 16. UC2: SA using deep learning in HFL in TFF. Creating a model using pretrained layers.

user himself from scratch could be also utilized as well, although it would not benefit from the advantages described above when using a pretrained and problem-specific model. Moreover, the model_fn() method presents some differences regarding the previous use case, mainly being the loss and the metrics in use. In this case, binary cross-entropy loss and accuracy are used instead of their categorical counterparts, due to the binary nature of the classification problem.

The rest of the process, including the configuration of the training strategy and finally running the FL algorithm, is performed as in the previous TFF's use case.

In this use case, we also show how to solve the SA in Flower; however, minimum differences exist in the solutions already presented. The data is loaded, processed and binarized as in TFF (see Listing 14 and Listing 15). In Section VII-A, we mentioned that Flower can handle either Keras or PyTorch models; while in UC1 we used a PyTorch model; in this case we use a Keras model so that we show how to use both of them. Therefore, the model is created as previously presented in Listing 16. The client_fn() method seen in Listing 7 remains the same, being the main difference that the create_keras_model() method is used instead of CNN_Net(). Finally, the training strategy is defined as in Listing 8. The whole process can be observed in the corresponding notebook.

Once we describe how to perform SA with both TFF and Flower, we present some results obtained from the execution of the corresponding experiments. Table VII reports the training and testing loss and accuracy obtained by the models, as well as the required runtime to perform the federated training and evaluation. In this case, although the data is obtained from the same source and the network architecture is the same (based on a pretrained Keras model), as well as the learning strategy, the results are different. It is interesting that Flower needs only around 40% of the time required by TFF to train the model, being a much faster option in this case. Besides, the testing accuracy is higher in Flower's implementation too, making it a better option for performing federated SA.

TABLE VII
RESULTS OF UC2: (SA) USING DEEP LEARNING IN HFL IN TFF AND FLOWER

	TFF	Flower
Train loss	0.563	0.002
Train accuracy	0.723	0.992
Test loss	0.494	0.006
Test accuracy	0.889	0.966
Runtime (s)	470.2	186.5

C. UC3: Decision Trees in VFL

In contrast to previous use cases, we shift to a vertically partitioned data scenario, following the workflow:

1) Problem Type: The data is vertically partitioned, we face a VFL problem.

2) Federated Distribution: As in most real-world VFL cases, in this case we deal with a tabular dataset. Specifically, we use the Credit2 dataset for bank credit risk prediction. From the three selected frameworks, only FATE provides support for vertically partitioned data, so neither TFF nor Flower are considered in this case. The data in this scenario does not have an inherent federated distribution (see Table I), so we simulate it as a non-overlapping attribute-skewed non-IID partition [67].

3) Model Selection: Although most of the work in FL relies on deep learning models, in this use case we show how to use traditional ML models such as decision trees, which are particularly suited to solve this problem. Indeed, decision trees have been used with considerable success throughout the literature [71]. We use the FATE's SecureBoost [81] implementation over vertically partitioned tabular data.

4) Aggregation Strategy: As decision trees cannot be expressed as an array of parameters, SecureBoost defines a specific aggregation scheme so that clients generate a single global model together.

5) Training and Evaluation Strategy: For the evaluation, each client holds its portion of test data; thus it is not available for either the rest of clients or any central server. In this case, although the test instances must refer to the same individuals, each client holds their unique and private features for each instance.

In VFL, features are distributed among the clients, however only one of them owns the label feature. In FATE, the client with the labels is referred to as the *guest*, while the other is referred to as the *host*. Furthermore, to use SecureBoost, the arbiter role is not needed. Given that the dataset is not inherently federated, it is artificially partitioned and distributed for the VFL scenario by FATE. In Listing 17 a CSV file is uploaded for each client. Any dataset in this format can be uploaded and used in FATE. Each CSV file contains a different number of client-specific attributes, following a non-overlapping feature skew non-IID partition, as well as an id for each instance, so the instances between different clients can be matched. Such data alignment must be made before training the model by adding an Intersection object to the pipeline (Listing 18). This allows each party to identify the data instances participating in the training, without any further information besides their id.

```
# Path to the FATE repository in the local machine
data_base = "/path/to/data/repository/FATE/"

# Add the data to the FATE framework before reading it
pipeline.add_upload_data(
   file=os.path.join(data_base,
    ↪  "default_credit_hetero_guest.csv"),
   table_name=guest_train_data["name"],
   namespace=guest_train_data["namespace"],
   extend_sid=True
)
# Add also host data

# Upload data
pipeline.upload()

# Define Reader component and add to the pipeline
reader_0 = Reader(name="reader_0")
# Add components to the pipeline
pipeline.add_component(reader_0)
```

Listing 17. UC3: decision trees in VFL in FATE. Loading dataset.

```
# Define DataTransform and Intersection components
data_transform_0 = DataTransform(name="data_transform_0",
 ↪   with_match_id=True)
intersection_0 = Intersection(name="intersection_0")

pipeline.add_component(data_transform_0,
 ↪   data=Data(data=reader_0.output.data))
pipeline.add_component(intersection_0,
 ↪   data=Data(data=data_transform_0.output.data))
```

Listing 18. UC3: decision trees in VFL in FATE. Data intersection by id.

The model and training strategy is defined as in Listing 19. Here, the different parameters of the model are set up: the objective or loss metric, the encryption type, and other parameters for the trees such as the number of trees and maximum depth. Then, the model component is added to the pipeline where, the input data of the model is the one received by the previous clients' data.

```
hetero_secureboost_0 = HeteroSecureBoost(
   name="hetero_secureboost_0",
   objective_param={"objective": "cross_entropy"},
   encrypt_param={"method": "paillier"},
   num_trees=10,
   tree_param={"max_depth": 3}
)

# Add to the pipeline
pipeline.add_component(hetero_secureboost_0,
 ↪   data=Data(train_data=intersection_0.output.data))
```

Listing 19. UC3: decision trees in VFL in FATE. Creating SecureBoost model.

The final steps to start the learning process are to compile and fit, as previously seen in Listing 13. Besides, the FATE board allows visualisation of some results, such as the structure of the built trees. Note that when analysing the trees, they only contain detailed information about the guest or host parties, depending on who is accessing it, that is, each guest party cannot observe information about other parties, and vice versa. For example, Fig. 7 shows the host who is accessing the tree; thus, in the nodes that are partitioned by its attributes, the full information of the node is presented, while for those that are partitioned by guest's attributes, no information is provided to prevent leakage of private information.

D. *UC4: Introducing Differential Privacy*

Previous use cases did not increase privacy during the training process, as discussed in Section III-D. In this use case, we show how to use DP techniques when training, thus enhancing data privacy on the client side.

Following the workflow for designing a FL scenario (see Fig. 6), the DP process leads either to the phase of model selection and deployment across nodes phase, or the design of the training strategy. The former refers to models that intrinsically protect the privacy of clients and their data by their design, while the latter refers to additional mechanisms to protect such privacy, which are independent of the model in use. In the literature, independent mechanisms are more common, since they can be applied to a wider range of methods.

The problem to face in this use case is HFL, and TFF is the only framework in our selection that implements DP mechanisms. The other frameworks are therefore discarded. The rest of the workflow is very similar to UC1 but including DP in the training process. Specifically, the widely used DP with adaptive clipping proposed in [82] is used, as it is the only DP method available in TFF. For the data, both intrinsic non-IID data distribution and simulated IID partition are included in the experimental setup. Having analysed the results in UC1, only the CNN model is considered, using a generic FedAvg aggregation strategy.

Differences with UC1 are scarce, as it is simple to incorporate DP into the learning process. As presented in Listing 20, the only change to introduce the DP mechanism is to define a DP aggregator when designing the learning strategy in the last step of the workflow. This aggregator receives as parameters a multiplier for the Gaussian noise, as well as the number of clients expected to participate in each round. Then, the aggregator is set up when creating the federated average process. In each round, clients send their parameters by inducing a certain amount of noise on their transmitted values, which further protects their local information.

In Fig. 8, we show the variation in performance in terms of testing accuracy and testing loss of the model trained using DP with the noise multiplier values. For that purpose, we run it with both IID and non-IID data partitions, and the reported results are averaged among 10 executions. It is shown, as expected, that the higher the noise introduced in the communication to increase privacy, the poorer the performance. According to this experiment, a value of 0.1 for the Gaussian noise multiplier still maintains predictive performance in terms of accuracy at the same level, in the IID and the non-IID scenario, while strengthening the local privacy. By further increasing the noise, performance degrades, but it may recover by increasing the number of learning rounds. Consequentially, increasing the noise multiplier above this threshold would depend on the problem requirements and the privacy concerns of the users involved in the scenario. Futhermore, it should be noted that the runtime remains constant, disregarding the level of induced noise. Finally, although in UC1 and UC2 Flower appeared to be the better option, several factors must be considered when choosing a framework. If privacy protection in the communications is a design factor to be pri-

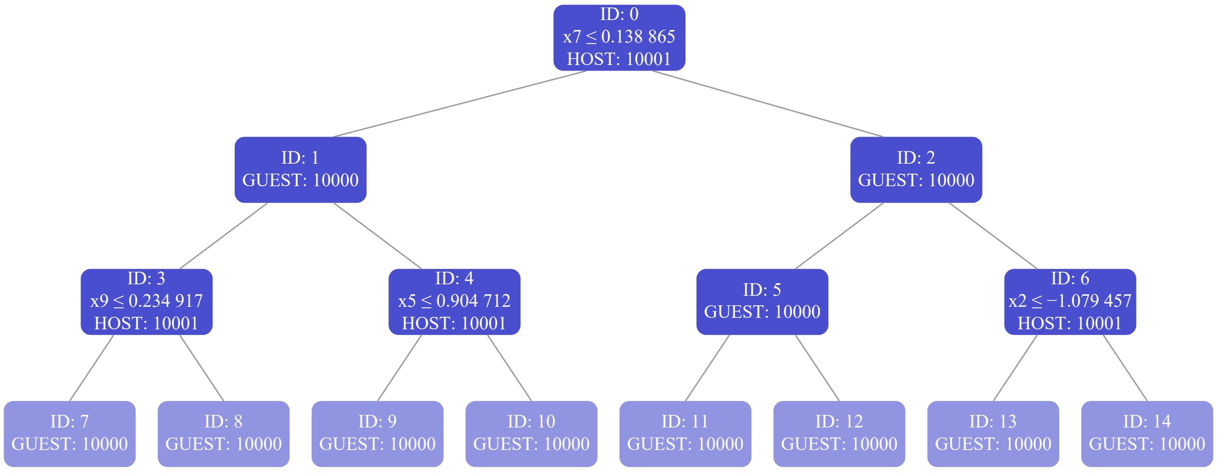

Fig. 7. Host's view of a decision tree built by SecureBoost for UC3.

```
# Aggregator with Differential Privacy
aggregation_factory = dp_aggregator(noise_mult, NCLIENTS)

# Include aggregator in the training process
training_process = build_unweighted_fed_avg(
    model_fn,
    client_optimizer_fn=lambda:
     ↪  optimizers.Adam(learning_rate=0.001),
    server_optimizer_fn=lambda:
     ↪  optimizers.Adam(learning_rate=0.01),
    model_aggregator=aggregation_factory
)
```

Listing 20. UC4: introducing DP in TFF. Training process including DP mechanisms.

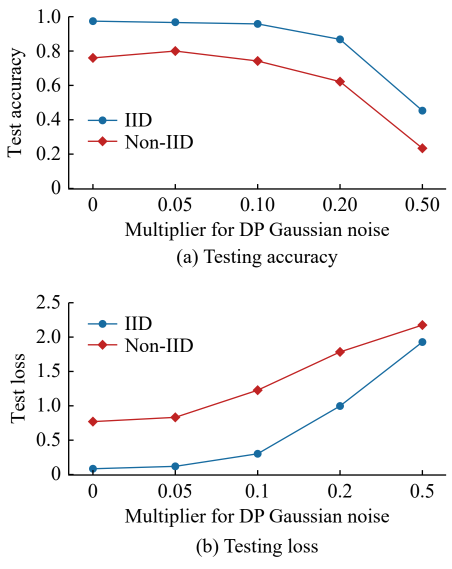

Fig. 8. Test results of UC4 as DP noise increases.

oritized and DP is hence necessary, to date TFF provides this functionality and can be a suitable option.

E. UC5: Clustering With K-Means in HFL

In addition to training predictive supervised models, unsupervised techniques can be adapted to FL according to the workflow presented:

1) Problem Type and Federated Distribution: In this use case we face an HFL problem, where the MNIST dataset is IID partitioned. TFF is used, since it is the only one of the three FL frameworks providing models for clustering tasks in FL scenarios.

2) Model Selection: As for the learning model, TFF offers an implementation of k-means; since such a model cannot be expressed as an array of parameters, a specific aggregator had to be designed in the TFF implementation to adapt the method to the federated environment.

3) Aggregation Strategy: At each round, TFF's k-means aggregator receives the centroids that each client computed locally, as well as the number of data points assigned to each centroid. Subsequently, the server computes the new centroids as a weighted combination of the client's centroids, where weights are related to the number of data points represented by every client centroid.

4) Training and Evaluation Strategy: In this case, to evaluate the clustering model, no testing data is used, but different unsupervised internal measures are computed to assess its performance, such as the analysis of the centroids. To the date, the considered frameworks do not offer many options for evaluating federated clustering models.

The data loading and partitioning is performed as in Listing 2. The model is created as in Listing 21, where the training strategy is also set up, considering as main parameters the number of clusters and the shape of the input data. The input data shape is set to 784 (input images of 28×28 pixels), while the number of clusters is set to the number of different digits in the dataset. After defining the model, the federated process is initialized and run for several rounds.

It should be highlighted that, while offering the k-means method, the options to further manipulate it beyond its training phase are very limited. Among these options, we can show the number of instances assigned to each cluster or the coordinates of the clusters' centroids (Listing 22).

Besides printing the centroids, they can also be plotted. In Fig. 9 the centroids of a classic centralized k-means are compared to the federated version participating 10 clients; in both cases, the algorithm ran for 10 iterations. It can be observed that the outcomes are similar in both cases: most of the centroids represent a clear digit, while there are others where it is not clear which digit are representing; such centroids comprise instances from different classes.

```
# Create k-means model
fed_process = build_fed_kmeans(
    num_clusters = 10,
    data_shape = (784,)
)

# Initialize and run federated process
fed_state = fed_process.initialize()
for round_num in range(1, NROUNDS+1):
   result = fed_process.next(fed_state, client_data)
```

Listing 21. UC5: clustering with k-means in HFL in TFF. Defining k-means model.

```
# Assignation of data examples to each cluster
print(result.state.finalizer)

# Centroids
for c in result.state.global_model_weights:
   print(c)
```

Listing 22. UC5: clustering with k-means in HFL in TFF. Printing some results.

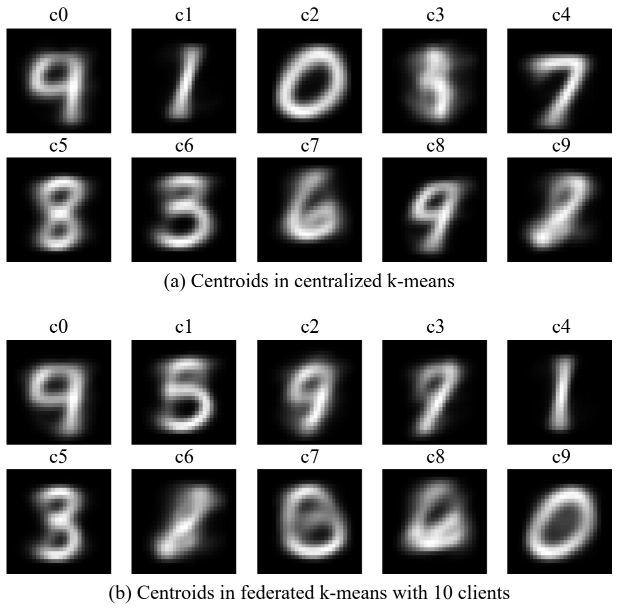
c0
c1
c2
c3
c4
c5
c6
c7
c8
c9
(a) Centroids in centralized k-means
c0
c1
c2
c3
c4
c5
c6
c7
c8
c9
(b) Centroids in federated k-means with 10 clients

Fig. 9. Comparison of k-means centroids using either the centralized or the federated version in UC5.

F. UC6: Clustering With K-Means in VFL

Clustering methods might be applied to vertically partitioned scenarios too. Following the workflow in Fig. 6:

1) Problem Type and Federated Distribution: In UC6 we face a VFL scenario over the Credit2 dataset, which is not inherently federated distributed. Therefore, a non-IID partition is simulated.

2) Model Selection: Once each client holds its specific data, FATE's implementation of k-means is used.

3) Aggregation Strategy: As in the previous use case, a specific aggregator for k-means models is used.

4) Training and Evaluation Strategy: Similar to UC5, the evaluation is performed by using internal clustering metrics, such as the compactness of the clusters and the distance between them; therefore, the evaluation is carried out on the same data on which the clustering process was conducted.

The process to perform clustering in a VFL scenario is similar to the training presented in UC3 (see Section VII-C). The main differences are the definition of the k-means model and its training strategy, as presented in Listing 23. The number of clusters is set to the number of classes in the dataset, and a maximum of 100 iterations are performed.

```
# Parameters for the k-means model
param = {
   "k": 2,
   "max_iter": 100
}

hetero_kmeans_0 = HeteroKmeans(name='hetero_kmeans_0',
 ↪   **param)
```

Listing 23. UC6: clustering with k-means in TFF. Printing some results.

In order to demonstrate the proper operation of k-means in a federated scenario, the Davies-Bouldin Index (DBI) [83] is analysed, and a minimized clustering metric evaluates the compactness of each cluster as well as the distance to the remaining clusters. Fig. 10 plots the DBI of both the centralized k-means execution using all the attributes in one party, and the VFL k-means using two different parties. Note that the k-means automatically stops execution if convergence is observed or the clusters do not change for many iterations. The results indicate that DBI in both cases reaches the same minimum value, however the centralized version requires fewer iterations. Therefore, it is demonstrated that equally competitive results can be obtained even if data access is restricted.

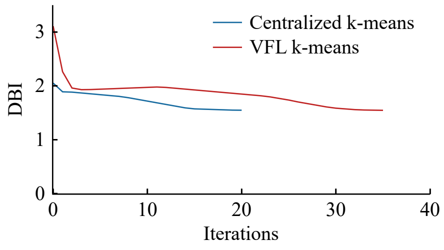

Fig. 10. Convergence comparison between centralized and VFL k-means in UC6.

G. Lessons Learned From a Practical Perspective

Based on the extensive study of the use cases and the analysis of datasets and software frameworks, the following lessons have been learned, which can be quite useful for practitioners:

• The precision of FL is competitive when compared to the performance of centralized scenarios, highlighting the non-IID scenario which is the most common case in real-world problems. Therefore, FL can be deemed an excellent option for distributed data silos, not only to preserve privacy, but also for good performance.

• FL can be applied to different state-of-the-art models, including deep neural networks (*deep learning*), unsupervised learning (e.g., clustering) or interpretable models (such as decision trees), among other ML approaches. We can find distributed implementations of FL for almost all ML approaches.

• DP can be added to a FL scenario to ensure data privacy without increasing running times, at the cost of a performance penalty. This has been observed in Fig. 8 (showing the degradation of performance).

- There are several software frameworks for implementing FL scenarios. However, hardly any of them include functionalities beyond the basics. Thus, there is still room for research before software frameworks with a larger range of models and functionalities become available.
- Among those tools analysed in depth in the experimental use cases, it has been clearly demonstrated that, although FATE offers some mechanisms to solve problems in HFL scenarios, it offers far fewer options than the rest of the frameworks and yields significantly worse performance. However, its applicability to VFL scenarios has been proven to be excellent. In contrast, when facing HFL problems, the choice between TFF and Flower may depend on several factors, such as i) the users' experience in TensorFlow and PyTorch; ii) the need for introducing data privacy in the process, where TFF includes DP mechanisms, but Flower does not; iii) the performance of the models, where Flower models seems to outperform those provided by TFF; or iv) by any other of the characteristics that distinguish them, noted in Table II.

VIII. Selected Trends in Federated Learning Studies and Machine Learning Approaches

FL has the potential to transform the way we approach ML, making it more efficient, secure, and privacy-preserving. In this section, we will explore some trends in FL, including those that arise naturally from its design and those that are carried over from the ML challenges. Such selected trends in FL studies are graphically summarized in Fig. 11 and discussed in the rest of the section.

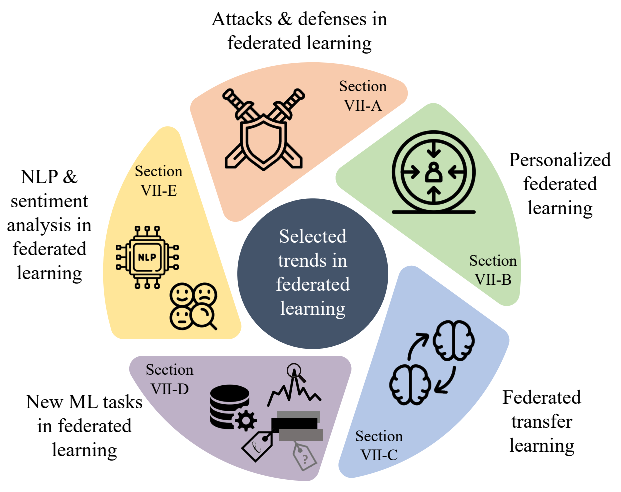
Attacks & defenses in federated learning
Section VII-A
Personalized federated learning
Section VII-B
Federated transfer learning
Section VII-C
New ML tasks in federated learning
Section VII-D
NLP & sentiment analysis in federated learning
Section VII-E
NLP
Selected trends in federated learning

Fig. 11. Selected trends in FL discussed in this manuscript.

A. Attacks and Defences in Federated Learning

As with any ML paradigm, FL is vulnerable to adversarial attacks [27]. The vast majority of the attacks in ML are built upon the manipulation of training data by third-parties [84]. FL, by definition, is exempt from such attacks as training data is inaccessible. However, it is exposed to numerous adversarial attacks, either from clients or the server, as well as from third parties in communications.

There is a wide range of possibilities when it comes to categorizing attacks according to different criteria such as the origin of the attack, the attacker's knowledge, the objective of the attack, and so forth [85], [86]. The main attack types are *attacks to the model*, with the aim of modifying the behaviour of the federated model, and *privacy attacks*, which attempt to infer some information about the training data allocated among the clients, namely: Property Inference Attacks [87], Feature Reconstruction Attacks [88] and Membership Inference Attacks [89].

Defensive strategies have been developed to address these types of attacks [90]–[92]. The nature of these defences, like that of the attacks, is diverse, and categorizations can be found based on various classification criteria. According to the place where the defence is allocated, we distinguish between: 1) Server defenses, which assume that the server is reliable and are usually based on robust aggregation operators, anomaly detection or the application of DP; 2) Client defenses, which assume that at least a portion of the clients is benign and are commonly based on the application of DP; and 3) Communication channel defenses, which embrace secure implementations of FL as SMC (see Section III-D1).

B. Personalized Federated Learning

FL introduces several advantages regarding the centralized ML such as generalization, privacy, and reduction of the communications. However, this solution does not address two desirable features of a model such as producing poor convergence on highly imbalanced data, and the customization of the models to the different clients' specifications. On one hand, the convergence of FL models degrades significantly when data imbalance is present. This performance degradation is attributed to the so-called client drift phenomenon [93]. Since the global model is averaged from multiple clients, it may not generalize well in a client whose data distribution has outliers. Thus, having a single model is often insufficient for practical applications of FL having imbalanced datasets from multiple clients. Personalized federated learning (PFL) [94] arises as a solution to both challenges without losing sight of the attributes provided by FL.

There are several proposals to achieve PFL. In [95] the following categorization of strategies for PFL is proposed:

- *Global Model Personalization:* The performance of PFL is directly dependent on the generalization performance of the global model, which is the reason why many PFL approaches aim to improve the performance of the global model under data imbalance. Within this category, a distinction is made between data-based approaches, based on mitigating the client drift problem by reducing the statistical heterogeneity among the clients' training data, and between model-based approaches, based on learning a global model robust enough for future customization on clients. Data-based approaches directly address the data imbalance present in FL at the client level [96], or select a subset of clients with minimal class imbalance [97]. In contrast to data-based approaches, which incur a loss of valuable information associated with the inherent diversity of client behaviors, model-based approaches try to improve the adaptation performance of the local model by means of regularization techniques [98].

● *Learning Personalized Models:* It is designed to address the customization problem applying different learning paradigms in the FL setting [43]. Within this category, we distinguish between architecture-based approaches, which provide a personalized model architecture to each client, and similarity-based approaches, which aim to take advantage of client relationships to improve performance. Architecture-based approaches focus on training shared layers between clients and some personalized layers locally at every client [99]. Another architecture-based approach is to train different architectures using a public dataset based on the updated consensus, and then to fine-tune each local model using the private dataset [100]. In similarity-based approaches, a personalized model is learned for each client, with related clients learning similar models through multitask learning to consider pairwise client relationships [101].

C. Federated Transfer Learning

When local datasets do not share sufficient common features or samples, FL may struggle. This is where FTL comes into play. Several researchers have applied FTL in real-world applications.

A secure FTL framework is formulated in [28], as a technique that integrates transfer learning into FL to construct a model from two datasets with different samples and feature spaces. This approach enables the transfer of knowledge from a pretrained model on a source dataset to a target dataset that has different feature spaces or data distributions. The FedSteg framework [102] uses FTL to detect hidden information in images for secure image steganalysis. Reference [103] utilized FTL without sharing vocabulary for privacy-preserving NLP applications in cancer registries. Reference [104] utilized knowledge distillation to train computationally affordable CNNs for edge devices. The authors proposed the Group Knowledge Transfer framework, which optimizes the client and server model alternatively with knowledge distillation loss. The larger server model takes features from the edge to minimize the gap between periodically transferred ground truth and soft label predicted by the edge model, while the small model distils knowledge from the larger server model using private data and soft labels transferred back from the server.

To summarize, FTL is a promising trend that combines the strengths of transfer learning and FL, suitable for real-world applications with privacy concerns or limited data availability.

D. Machine Learning Tasks With Federated Learning

ML techniques have revolutionized the way we approach data analysis and decision-making. In this section, we explore their extension to FL and how they can improve its performance.

a) Semi-Supervised Federated Learning: This builds upon the idea of effectively using unlabelled data to enhance training. The usage of unlabelled data is also motivated by FL itself, as it imposes strong privacy requirements, which makes large-scale labelling unfeasible. However, there are also scenarios where labeling costs are significantly reduced due to being mostly automated. A remarkable example is language modelling, where labelling is achieved through user typing behavior [14].

Popular semi-supervised learning techniques based on Consistency Regularization [105] and Pseudo-labelling [106] can be naively applied to FL. However, further adaptation is required to notably increase the performance of such techniques. Similar adaptations have been developed in parallel by some authors, showing great success in the process [107]. Additional approaches based on knowledge distillation techniques have also been proposed [108].

While the performance gap between supervised FL and semi-supervised FL has been notably narrowed [109], there is still room for improvement, since the field of semi-supervised FL has received little attention in the literature. The FL ecosystem can greatly benefit from the latest advances in this field, as the sources of unlabelled data are constantly growing.

b) Federated Learning for Anomaly Detection: Anomaly detection is the task of identifying unusual data points or patterns that do not conform to the expected or normal behaviour and may indicate unusual events that may require further investigation [110]. The relationship between FL and anomaly detection becomes essential mainly when dealing with IoT environments. However, IoT devices have been increasing their computing power recently, to the point that they can deploy ML models by themselves. Therefore, to preserve the privacy of sensor data and to reduce communication overheads, many of these anomaly detection methods have been adapted to federated scenarios.

The work on Intrusion Detection Systems, whose aim is to detect attempts to compromise the integrity, confidentiality, or availability of networks, is extensive in federated scenarios [111]. FL has also been used in financial problems [112], which is a very common field in anomaly detection. In this case, it is essential to preserve data privacy to the maximum extent possible since different private organizations could be cooperating to detect financial frauds while using sensitive data. Another major research field that benefits from the combination of FL and anomaly detection is Predictive Maintenance [113]. In many industrial settings, the machinery is monitored and sensorized so that, it allows the construction of models to detect anomalous behaviour that may be related to an early or future breakdown of such machinery.

Not only have anomaly detection methods been adapted to FL environments, but also some anomaly detection mechanisms have been incorporated in certain cases to improve the security of existing FL algorithms. The main use case has been the detection and defence from poisoning attacks, by detecting clients that may be operating anomalously regarding the rest of the participants, thus discarding their updates or completely removing them from the training process. [114], [115].

E. Natural Language Processing and Sentiment Analysis in Federated Learning

NLP and SA are two critical fields in ML that have numerous applications in areas such as customer service, marketing, and social media analysis [116]. In the FL context, NLP and SA are used to analyse natural language text data distributed across multiple devices or servers without the need for data to be centralized [117]. This approach allows for the creation of robust models without the need to share sensitive data, thereby addressing privacy concerns. However, the distributed nature of FL presents several challenges for NLP and SA. For example, due to variations in the data across different devices, the model must be able to learn from data with different distributions and handle noisy data. Furthermore, data imbalance and heterogeneity across different devices pose additional challenges for NLP and SA [118].

Despite these challenges, NLP and SA have been successfully implemented in FL [8]. One approach is to use transfer learning techniques to pretrain the model on large, centralized datasets before fine-tuning it on the distributed data. This approach has been shown in Section VII-B to be effective in improving model performance on distributed data, where clients use a pretrained model to solely train the last layers of the model, converging faster than training the whole model from scratch. Another approach is to use DP techniques to protect the privacy of individual data points during training. Another nascent approach is to apply contrastive representation learning to handle crowds [119], in which FL can help to deal with numerous individuals to contribute to data labelling and annotation tasks, which is a bottleneck in NLP. However, these techniques can lead to decreased model accuracy, and more research is needed to address this issue.

In summary, NLP and SA are critical fields that have numerous applications in ML, and their implementation in FL offers a promising solution for privacy-preserving ML. Despite the challenges posed by the distributed nature of FL, successful implementations have been achieved through transfer learning and DP techniques. Ongoing research is needed to address the remaining challenges and to further advance the field of NLP and SA in FL.

IX. Conclusions

As has been shown in this tutorial, FL has become a crucial field for training machine learning models across decentralized environments. In situations when data privacy is a hard requirement, FL allows ML models to be trained locally while leveraging the knowledge gathered from other nodes of the decentralized network without any need for centralized data transfer. This reduces the vulnerability of sensitive data and mitigates the risk of interception by third parties.

The instructive view on FL provided in this tutorial encompasses FL foundations from key elements to architectures and categories, a design methodology, eighteen software frameworks, six exemplary use cases, and selected trends. This material can be useful for developers and researchers willing to gain a comprehensive understanding of this field. Moreover, the use cases covering different possible real-world scenarios have been thoroughly described, and we have discussed how to solve them with three different frameworks (TFF, Flower and FATE). This implementation guide aims to support practitioners when designing and building reliable and secure ML models that respect data privacy and that provide accurate and useful results for a wide range of distributed computing applications and environments.

On an overarching note, FL has grown in maturity over the years, due to the large number of studies and different frameworks proposed to solve FL problems. However, there is a widespread consensus around the promising path that lies ahead for this research area. New challenges and opportunities will surely arise for future research, which are regularly discussed in prospective surveys on the topic [120]–[126].

Beyond the realm of research, recent data privacy regulations (including the Data Governance Act [127] and the Artificial Intelligence Act [128] of the European Parliament, among others) have emphasized the significance of *privacy* as one of the fundamental requirements for trustworthy and responsible AI. In this regard, FL can be identified as a pivotal technology to ensure that sensitive data remains confidential in high-risk scenarios throughout the entire AI life cycle. This regulatory context elevates FL as a field of utmost relevance for realizing trustworthy and responsible AI [129], highlighting the need for reference material as the tutorial contributed in this manuscript. Trends in responsible AI, stressing privacy-preserving techniques and security enhancements, are closely aligned with the need for safeguarding sensitive user data and ensuring the ethical treatment of individuals' information. These needs lie at the heart of FL. As FL becomes increasingly integrated into practical problems arising from different sectors, aligning its development with responsible AI guidelines not only fosters trust and societal acceptance on this decentralized machine learning approach, but also reinforces its responsible and sustainable growth in the future.

Appendix
Abbreviations, Notations and Framework URLs

In this Appendix, we include the list of abbreviations used throughout the manuscript, as follows:

UC1	Use Case 1
UC2	Use Case 2
UC3	Use Case 3
UC4	Use Case 4
UC5	Use Case 5
UC6	Use Case 6
AI	Artificial Intelligence
FedAvg	Federated Averaging
FL	Federated Learning
IoT	Internet of Things
IoHT	Internet Of Healthcare Things
IID	Independent and Identically Distributed
ML	Machine Learning
non-IID	Non Independent and Identically Distributed
HFL	Horizontal Federated Learning
VFL	Vertical Federated Learning
FTL	Federated Transfer Learning
SMC	Secure Multiparty Computation
HE	Homomorphic Encryption
DP	Differential Privacy
LDP	Local Differential Privacy

CDP	Central Differential Privacy
TFF	TensorFlow Federated
PyS	PySyft
FAT	FATE
Pad	PaddleFL
Flo	Flower
Xay	Xaynet
IBM	IBM FL
Sub	Substra
OFL	OpenFL
FML	FedML
FJx	FedJax
101	Backdoors 101
FLb	FedLab
SFL	SimFL
EFL	easyFL
TFL	TorchFL
AFL	APPFL
NVF	NVFlare
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
NLP	Natural Language Processing
DBI	Davies-Bouldin Index
PFL	Personalized Federated Learning
SA	Sentiment Analysis

We also provide a table of the notation employed in formal definitions in Table VIII.

TABLE VIII
NOTATION EMPLOYED IN FORMAL DEFINITIONS

Term	Definition
$\{C_1, ..., C_n\}$	Set of data owners (also known as clients).
D_i	Training data of client C_i.
L_i	Local learning model (expressed as parameters) of client C_i.
G	Global learning model.
t	Round of learning.
L_i^t	Parameters of local model L_i at the t-th round before training.
$\hat{L}_i^t$	Parameters of local model L_i at the t-th round after training.
G_t	Parameters of the global learning model at the t-th round.
X	Feature space.
Y	Label space.
I	ID space.

Furthermore, we state that each software framework reviewed were accessed using the following URLs as of October 3rd, 2023:

- TensorFlow Federated https://github.com/tensorflow/federated
- PySyft https://github.com/OpenMined/PySyft
- FATE https://github.com/FederatedAI/FATE
- PaddleFL https://github.com/PaddlePaddle/PaddleFL
- Flower https://github.com/adap/flower
- Xaynet https://github.com/xaynetwork/xaynet
- IBM FL https://github.com/IBM/federated-learning-lib
- Substra https://github.com/Substra/substra
- OpenFL https://github.com/intel/openfl
- FedML https://github.com/FedML-AI/FedML
- FedJax https://github.com/google/fedjax
- Backdoors 101 https://github.com/ebagdasa/backdoors101
- FedLab https://github.com/SMILELab-FL/FedLab
- SimFL https://github.com/Xtra-Computing/SimFL
- easyFL https://github.com/EasyFL-AI/EasyFL
- TorchFL https://github.com/vivekkhimani/torchfl
- APPFL https://github.com/APPFL/APPFL
- NVFlare https://github.com/NVIDIA/NVFlare

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[2] T. Goldstein, "Challenges for machine learning on distributed platforms (invited talk)," in *Proc. 32nd Int. Symp. Distributed Computing*, New Orleans, USA, 2018, pp. 2:1–2:3.

[3] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," arXiv preprint arXiv: 1610.05492, 2016.

[4] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*. Cham, Germany: Springer, 2019.

[5] J. Zhu, J. Cao, D. Saxena, S. Jiang, and H. Ferradi, "Blockchain-empowered federated learning: Challenges, solutions, and future directions," *ACM Comput. Surv.*, vol. 55, no. 11, p. 240, Nov. 2023.

[6] D. Chen, X. Jiang, H. Zhong, and J. Cui, "Building trusted federated learning: Key technologies and challenges," *J. Sens. Actuator Netw.*, vol. 12, no. 1, p. 13, Feb. 2023.

[7] R. Zong, Y. Qin, F. Wu, Z. Tang, and K. Li, "Fedcs: Efficient communication scheduling in decentralized federated learning," *Inf. Fusion*, vol. 102, p. 102028, Feb. 2024.

[8] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Comput. Ind. Eng.*, vol. 149, p. 106854, Nov. 2020.

[9] European Commission, "Machine learning ledger orchestration for drug discovery," 2019. [Online]. Available: https://cordis.europa.eu/project/id/831472.

[10] B. Mahesh, "Machine learning algorithms–A review," *Int. J. Sci. Res.*, vol. 9, no. 1, pp. 381–386, Jan. 2020.

[11] D. Wang, W. Yao, T. Jiang, G. Tang, and X. Chen, "A survey on physical adversarial attack in computer vision," arXiv preprint arXiv: 2209.14262, 2022.

[12] K. Abouelmehdi, A. Beni-Hssane, H. Khaloufi, and M. Saadi, "Big data security and privacy in healthcare: A review," *Procedia Comput. Sci.*, vol. 113, pp. 73–80, Dec. 2017.

[13] European Commission, "High-level expert group on artificial intelligence, ethics guidelines for trustworthy AI," European Union, 2019. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/expert-group-ai.

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artificial Intelligence and Statistics*, Fort Lauderdale, USA, 2017, pp. 1273–1282.

[15] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *Proc. IEEE Custom Integrated Circuits Conf.*, Austin, USA, 2017, pp. 1–8.

[16] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, "Big data analytics: A survey," *J. Big Data*, vol. 2, no. 1, p. 21, Oct. 2015.

[17] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big IoT data analytics: Architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, Mar. 2017.

[18] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep

unsupervised learning using graphics processors," in *Proc. 26th Ann. Int. Conf. Machine Learning*, Montreal, Canada, 2009, pp. 873–880.

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

[20] K. K. Coelho, M. Nogueira, A. B. Vieira, E. F. Silva, and J. A. M. Nacif, "A survey on federated learning for security and privacy in healthcare applications," *Comput. Commun.*, vol. 207, pp. 113–127, Jul. 2023.

[21] A. Chaddad, Q. Lu, J. Li, Y. Katib, R. Kateb, C. Tanougast, A. Bouridane, and A. Abdulkadir, "Explainable, domain-adaptive, and federated artificial intelligence in medicine," *IEEE/CAA J. Autom. Sinica*, vol. 10, no. 4, pp. 859–876, Apr. 2023.

[22] S. Naz, K. T. Phan, and Y.-P. P. Chen, "A comprehensive review of federated learning for COVID-19 detection," *Int. J. Intell. Syst.*, vol. 37, no. 3, pp. 2371–2392, Mar. 2022.

[23] X. Han, H. Yu, and H. Gu, "Visual inspection with federated learning," in *Proc. 16th Int. Conf. Image Analysis and Recognition*, Waterloo, Canada, 2019, pp. 52–64.

[24] N. I. Mowla, N. H. Tran, I. Doh, and K. Chae, "Federated learning-based cognitive detection of jamming attack in flying ad-hoc network," *IEEE Access*, vol. 8, pp. 4338–4350, 2020.

[25] N. Rodríguez-Barroso, G. Stipcich, D. Jiménez-López, J. A. Ruiz-Millán, E. Martínez-Cámara, G. González-Seco, M. V. Luzón, M. A. Veganzones, and F. Herrera, "Federated learning and differential privacy: Software tools analysis, the sherpa.ai FL framework and methodological guidelines for preserving data privacy," *Inf. Fusion*, vol. 64, pp. 270–292, Dec. 2020.

[26] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, and S. Barro, "Non- II D data and continual learning processes in federated learning: A long road ahead," *Inf. Fusion*, vol. 88, pp. 263–280, Dec. 2022.

[27] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara, "Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges," *Inf. Fusion*, vol. 90, pp. 148–173, Feb. 2023.

[28] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 70–82, Jul.–Aug. 2020.

[29] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. USENIX Ann. Technical Conf.*, 2020, pp. 33.

[30] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *Proc. ACM SIGSAC Conf. Computer and Communications Security*, 2020, pp. 1253–1269.

[31] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. 3rd Theory of Cryptography Conf.*, New York, USA, 2006, pp. 265–284.

[32] T. Wang, X. Zhang, J. Feng, and X. Yang, "A comprehensive survey on local differential privacy toward data statistics and analysis," *Sensors*, vol. 20, no. 24, p. 7030, Dec. 2020.

[33] M. Naseri, J. Hayes, and E. De Cristofaro, "Local and central differential privacy for robustness and privacy in federated learning," in *Proc. 29th Annu. Network and Distributed System Security Symp.*, San Diego, USA, 2022, pp. 24–28.

[34] R. Eriguchi, A. Ichikawa, N. Kunihiro, and K. Nuida, "Efficient noise generation to achieve differential privacy with applications to secure multiparty computation," in *Proc. 25th Int. Conf. Financial Cryptography and Data Security*, 2021, pp. 271–290.

[35] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," in *Proc. 33rd Conf. Neural Information Processing Systems*, Vancouver, Canada, 2019.

[36] The TensorFlow Federated Authors (Google), "Tensorflow federated," 2018.

[37] N. Rodríguez-Barroso, E. Martínez-Cámara, M. V. Luzón, and F. Herrera, "Backdoor attacks-resilient aggregation based on robust filtering of outliers in federated learning for image classification," *Knowl.-Based Syst.*, vol. 245, p. 108588, Jun. 2022.

[38] L. Qu, Y. Zhou, P. P. Liang, Y. Xia, F. Wang, E. Adeli, L. Fei-Fei, and D. Rubin, "Rethinking architecture design for tackling data heterogeneity in federated learning," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, New Orleans, USA, 2022, pp. 10051–10061.

[39] Y. Zheng, S. Lai, Y. Liu, X. Yuan, X. Yi, and C. Wang, "Aggregation service for federated learning: An efficient, secure, and more resilient realization," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 2, pp. 988–1001, Mar-Apr. 2023.

[40] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting shared representations for personalized federated learning," in *Proc. 38th Int. Conf. Machine Learning*, 2021, pp. 2089–2099.

[41] Z. Charles, Z. Garrett, Z. Huo, S. Shmulyian, and V. Smith, "On large-cohort training for federated learning," in *Proc. 35th Conf. Neural Information Processing Systems*, 2021, pp. 20461–20475.

[42] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large CNNs at the edge," in *Proc. 34th Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 14068–14080.

[43] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on Non- II D data with reinforcement learning," in *Proc. IEEE Conf. Computer Communications*, Toronto, Canada, 2020, pp. 1698–1707.

[44] Z. Lian, W. Wang, and C. Su, "COFEL: Communication-efficient and optimized federated learning with local differential privacy," in *Proc. IEEE Int. Conf. Communications*, Montreal, Canada, 2021, pp. 1–6.

[45] N. Rodríguez-Barroso, E. Martínez-Cámara, M. V. Luzón, and F. Herrera, "Dynamic defense against byzantine poisoning attacks in federated learning," *Future Gener. Comput. Syst.*, vol. 133, pp. 1–9, 2022.

[46] Y. Zhang, M. Duan, D. Liu, L. Li, A. Ren, X. Chen, Y. Tan, and C. Wang, "CSAFL: A clustered semi-asynchronous federated learning framework," in *Proc. Int. Joint Conf. Neural Networks*, Shenzhen, China, 2021, pp. 1–10.

[47] T.-M. H. Hsu, H. Qi, and M. Brown, "Federated visual classification with real-world data distribution," in *Proc. 16th European Conf. Computer Vision*, Glasgow, UK, 2020, pp. 76–92.

[48] D. Caldarola, B. Caputo, and M. Ciccone, "Improving generalization in federated learning by seeking flat minima," in *Proc. 17th European Conf. Computer Vision*, Tel Aviv, Israel, 2022, pp. 654–672.

[49] S. K. Prashanthi, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the performance of accelerated Jetson edge devices for training deep learning models," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, p. 44, Dec. 2022.

[50] O. Marfoq, C. Xu, G. Neglia, and R. Vidal, "Throughput-optimal topology design for cross-silo federated learning," in *Proc. 34th Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 19478–19487.

[51] K. R. Jayaram, A. Verma, G. Thomas, and V. Muthusamy, "Just-in-time aggregation for federated learning," in *Proc. 30th Int. Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Nice, France, 2022, pp. 1–8.

[52] G. Kaissis, A. Ziller, J. Passerat-Palmbach, T. Ryffel, D. Usynin, A. Trask, I. Jr. Lima, J. Mancuso, F. Jungmann, M.-M. Steinborn, A. Saleh, M. Makowski, D. Rueckert, and R. Braren, "End-to-end privacy preserving deep learning on multi-institutional medical imaging," *Nat. Mach. Intell.*, vol. 3, no. 6, pp. 473–484, May 2021.

[53] D. Manna, H. Kasyap, and S. Tripathy, "MILSA: Model interpretation based label sniffing attack in federated learning," in *Proc. 18th Int. Conf. Information Systems Security*, Tirupati, India, 2022, pp. 139–154.

[54] H. Kasyap and S. Tripathy, "Privacy-preserving decentralized learning framework for healthcare system," *ACM Trans. Multimed. Comput., Commun., Appl.*, vol. 17, no. 2S, p. 68, Jun. 2021.

[55] N. Agarwal, P. Kairouz, and Z. Liu, "The skellam mechanism for differentially private federated learning," in *Proc. 35th Neural Information Processing Systems*, 2021, pp. 5052–5064.

[56] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. 23rd Int. Conf. Artificial Intelligence and Statistics*, Palermo, Italy, 2020, pp. 2938–2948.

[57] F. Lai, Y. Dai, S. S. V. Singapuram, J. Liu, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking model and system performance of federated learning at scale," in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, USA, 2022, pp. 11814–11827.

[58] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "FedAT: A high-performance and communication-efficient federated

learning system with asynchronous tiers," in *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, St. Louis, USA, 2021, pp. 60.

[59] K. Singhal, H. Sidahmed, Z. Garrett, S. Wu, J. Rush, and S. Prakash, "Federated reconstruction: Partially local federated learning," in *Proc. 35th Conf. Neural Information Processing Systems*, 2021, pp. 11220–11232.

[60] X. Li, Y. Hu, W. Liu, H. Feng, L. Peng, Y. Hong, K. Ren, and Z. Qin, "OpBoost: A vertical federated tree boosting framework based on order-preserving desensitization," *Proc. VLDB Endowment*, vol. 16, no. 2, pp. 202–215, Oct. 2022.

[61] T. Qi, F. Wu, C. Wu, L. Lyu, T. Xu, H. Liao, Z. Yang, Y. Huang, and X. Xie, "FairVFL: A fair vertical federated learning framework with contrastive adversarial learning," in *Proc. 36th Conf. Neural Information Processing Systems*, New Orleans, LA, USA, 2022, pp. 7852–7865.

[62] D. Cha, M. D. Sung, and Y.-R. Park, "Implementing vertical federated learning using autoencoders: Practical application, generalizability, and utility study," *JMIR Med. Inf.*, vol. 9, no. 6, p. e26598, Jun. 2021.

[63] X. Chen, S. Zhou, B. Guan, K. Yang, H. Fao, H. Wang, and Y. Wang, "Fed-EINI: An efficient and interpretable inference framework for decision tree ensembles in vertical federated learning," in *Proc. IEEE Int. Conf. Big Data*, Orlando, USA, 2021, pp. 1242–1248.

[64] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2090–2103, Aug. 2020.

[65] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 70–82, Jul.–Aug. 2020.

[66] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non- II D data silos: An experimental study," in *Proc. IEEE 38th Int. Conf. Data Engineering*, Kuala Lumpur, Malaysia, 2022, pp. 965–978.

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

[68] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of image classification algorithms based on convolutional neural networks," *Remote Sens.*, vol. 13, no. 22, p. 4712, Nov. 2021.

[69] W. Li, L. Zhu, Y. Shi, K. Guo, and E. Cambria, "User reviews: Sentiment analysis using lexicon integrated two-channel CNN–LSTM family models," *Appl. Soft Comput.*, vol. 94, p. 106435, Sept. 2020.

[70] S. Wang, J. Cao, and P. S. Yu, "Deep learning for spatio-temporal data mining: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 8, pp. 3681–3700, Aug. 2022.

[71] S. Bhatore, L. Mohan, and Y. R. Reddy, "Machine learning techniques for credit risk evaluation: A systematic literature review," *J. Banking Financ. Technol.*, vol. 4, pp. 111–138, 2020.

[72] A. Hilmkil, S. Callh, M. Barbieri, L. R. Sütfeld, E. L. Zec, and O. Mogren, "Scaling federated learning for fine-tuning of large language models," in *Proc. 26th Int. Conf. Applications of Natural Language to Information Systems*, Saarbrücken, Germany, 2021, pp. 15–23.

[73] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Proc. 34th Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 198.

[74] D. Liu and T. Miller, "Federated pretraining and fine tuning of BERT using clinical notes from multiple silos," arXiv preprint arXiv: 2002.08562, 2020.

[75] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, "Model aggregation techniques in federated learning: A comprehensive survey," *Future Gener. Comput. Syst.*, vol. 150, pp. 272–293, Jan. 2024.

[76] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Proc. 34th Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 198.

[77] X. Gong, A. Sharma, S. Karanam, Z. Wu, T. Chen, D. Doermann, and A. Innanje, "Preserving privacy in federated learning with ensemble cross-domain knowledge distillation," in *Proc. 36th AAAI Conf. Artificial Intelligence*, 2022, pp. 11891–11899.

[78] B. Soltani, Y. Zhou, V. Haghighi, and J. C. S. Lui, "A survey of federated evaluation in federated learning," in *Proc. 32nd Int. Joint Conf. Artificial Intelligence*, Macao, China, 2023, pp. 758.

[79] F. Lai, Y. Dai, S. S. V. Singapuram, J. Liu, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking model and system performance of federated learning at scale," in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, USA, 2022, pp. 11814–11827.

[80] V. Siomos and J. Passerat-Palmbach, "Contribution evaluation in federated learning: Examining current approaches," in *Proc. 1st NeurIPS Workshop on New Frontiers in Federated Learning*, 2021.

[81] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, Nov.–Dec. 2021.

[82] G. Andrew, O. Thakkar, H. B. McMahan, and S. Ramaswamy, "Differentially private learning with adaptive clipping," in *Proc. 35th Conf. Neural Information Processing Systems*, 2021, pp. 17455-17466.

[83] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979.

[84] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Comput. Sci. Rev.*, vol. 34, p. 100199, Nov. 2019.

[85] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-Y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," in *Proc. 34th Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 1348.

[86] M. S. Jere, T. Farnan, and F. Koushanfar, "A taxonomy of attacks on federated learning," *IEEE Secur. Privacy*, vol. 19, no. 2, pp. 20–28, Mar-Apr. 2021.

[87] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *Proc. ACM SIGSAC Conf. Computer and Communications Security*, Toronto, Canada, 2018, pp. 619–633.

[88] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "Updates-Leak: Data set inference and reconstruction attacks in online learning," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 1291–1308.

[89] D. Wu, S. Qi, Y. Qi, Q. Li, B. Cai, Q. Guo, and J. Cheng, "Understanding and defending against white-box membership inference attack in deep learning," *Knowl.-Based Syst.*, vol. 259, p. 110014, Jan. 2023.

[90] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and P. S. Yu, "Privacy and robustness in federated learning: Attacks and defenses," *IEEE Trans. Neural Networks Learn. Syst.*, 2022. DOI: 10.1109/TNNLS.2022.3216981

[91] Y. Liang, Y. Li, and B.-S. Shin, "Auditable federated learning with byzantine robustness," *IEEE Trans. Comput. Soc. Syst.*, 2023. DOI: 10.1109/TCSS.2023.3266019

[92] K. Zhang, C. Keliris, T. Parisini, B. Jiang, and M. M. Polycarpou, "Passive attack detection for a class of stealthy intermittent integrity attacks," *IEEE/CAA J. Autom. Sinica*, vol. 10, no. 4, pp. 898–915, Apr. 2023.

[93] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. 37th Int. Conf. Machine Learning*, 2020, pp. 5132–5143.

[94] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3347–3366, Apr. 2023.

[95] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 34, no. 12, pp. 9587–9603, Dec. 2023.

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

[97] M. Yang, X. Wang, H. Zhu, H. Wang, and H. Qian, "Federated learning with class imbalance reduction," in *Proc. 29th European Signal Processing Conf.*, Dublin, Ireland, 2021, pp. 2174–2178.

[98] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Nashville, USA, 2021, pp. 10708–10717.

[99] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," arXiv preprint arXiv:

1912.00818, 2019.

[100] D. Li and J. Wang, “FedMD: Heterogenous federated learning via model distillation,” arXiv preprint arXiv: 1910.03581, 2019.

[101] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, “Personalized cross-silo federated learning on non-IID data,” in *Proc. AAAI Conf. Artificial Intelligence*, 2021, pp. 7865–7873.

[102] H. Yang, H. He, W. Zhang, and X. Cao, “FedSteg: A federated transfer learning framework for secure image steganalysis,” *IEEE Trans. Network Sci. Eng.*, vol. 8, no. 2, pp. 1084–1094, Apr.–Jun. 2021.

[103] M. Alawad, H.-J. Yoon, S. Gao, B. Mumphrey, X.-C. Wu, E. B. Durbin, J. C. Jeong, I. Hands, D. Rust, L. Coyle, L. Penberthy, and G. Tourassi, “Privacy-preserving deep learning NLP models for cancer registries,” *IEEE Trans. Emerging Topics in Computing*, vol. 9, no. 3, pp. 1219–1230, Jul.–Sep. 2021.

[104] C. He, M. Annavaram, and S. Avestimehr, “Group knowledge transfer: Federated learning of large CNNs at the edge,” in *Proc. 34th Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 14068–14080.

[105] D. Berthelot, N. Carlini, I. Goodfellow, A. Oliver, N. Papernot, and C. Raffel, “MixMatch: A holistic approach to semi-supervised learning,” in *Proc. 33rd Conf. Neural Information Processing Systems*, Vancouver, Canada, 2019, pp. 5049–5059.

[106] D.-H. Lee, “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks,” in *Proc. Workshop on Challenges in Representation Learning*, Atlanta, USA, 2013, pp. 896.

[107] W. Jeong, J. Yoon, E. Yang, and S. J. Hwang, “Federated semi-supervised learning with inter-client consistency & disjoint learning,” in *Proc. 9th Int. Conf. Learning Representations*, 2021, pp. 1–15.

[108] S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto, “Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-IID private data,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 191–205, Jan. 2023.

[109] C. Zhang, L. Zhu, D. Shi, J. Zheng, H. Chen, and B. Yu, “Semi-supervised feature selection with soft label learning,” *IEEE/CAA J. Autom. Sinica*, 2022. DOI: 10.1109/JAS.2022.106055

[110] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, Jul. 2009.

[111] S. Agrawal, S. Sarkar, O. Aouedi, G. Yenduri, K. Piamrat, M. Alazab, S. Bhattacharya, P. K. R. Maddikunta, and T. R. Gadekallu, “Federated learning for intrusion detection system: Concepts, challenges and future directions,” *Comput. Commun.*, vol. 195, pp. 346–361, Nov. 2022.

[112] M. Schreyer, T. Sattarov, and D. Borth, “Federated and privacy-preserving learning of accounting data in financial statement audits,” in *Proc. 3rd ACM Int. Conf. AI in Finance*, New York, USA, 2022, pp. 105–113.

[113] A. Bemani and N. Björsell, “Aggregation strategy on federated machine learning algorithm for collaborative predictive maintenance,” *Sensors*, vol. 22, no. 16, p. 6252, Aug. 2022.

[114] C. Richards, S. Khemani, and F. Li, “Evaluation of various defense techniques against targeted poisoning attacks in federated learning,” in *Proc. IEEE 19th Int. Conf. Mobile Ad Hoc Smart Syst.*, Denver, USA, 2022, pp. 693–698.

[115] S. Shi, C. Hu, D. Wang, Y. Zhu, and Z. Han, “Federated anomaly analytics for local model poisoning attack,” *IEEE J. Sel. Areas Commun.*, vol. 40, no. 2, pp. 596–610, Feb. 2022.

[116] A. Rajput, “Natural language processing, sentiment analysis, and clinical analytics,” in *Innovation in Health Informatics: A Smart Healthcare Primer*, M. D. Lytras and A. Sarirete, Eds. Amsterdam, Netherlands: Elsevier, 2020, pp. 79–97.

[117] B. Y. Lin, C. He, Z. Ze, H. Wang, Y. Hua, C. Dupuy, R. Gupta, M. Soltanolkotabi, X. Ren, and S. Avestimehr, “FedNLP: Benchmarking federated learning methods for natural language processing tasks,” in *Findings of the Association for Computational Linguistics*, Seattle, United States, 2022, pp. 157–175.

[118] M. Liu, S. Ho, M. Wang, L. Gao, Y. Jin, and H. Zhang, “Federated learning meets natural language processing: A survey,” arXiv preprint arXiv: 2107.12603, 2021.

[119] H. Yang, X. Li, and W. Pedrycz, “Learning from crowds with contrastive representation,” *IEEE Access*, vol. 11, pp. 40182–40191, Jan. 2023.

[120] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” *Knowl.-Based Syst.*, vol. 216, p. 106775, Mar. 2021.

[121] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. DÓliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecnỳ, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramér, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, “Advances and open problems in federated learning,” *Found. Trends® Mach. Learn.*, vol. 14, no. 1–2, pp. 1–210, Jun. 2021.

[122] K. Zhang, X. Song, C. Zhang, and S. Yu, “Challenges and future directions of secure federated learning: A survey,” *Front. Comput. Sci.*, vol. 16, no. 5, p. 165817, Feb.–Dec. 2022.

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

[124] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara, “Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges,” *Inf. Fusion*, vol. 90, pp. 148–173, Feb. 2023.

[125] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, “A survey on federated learning: Challenges and applications,” *Int. J. Mach. Learn. Cybern.*, vol. 14, no. 2, pp. 513–535, Feb. 2023.

[126] R. Al-Huthaifi, T. Li, W. Huang, J. Gu, and C. Li, “Federated learning in smart cities: Privacy and security survey,” *Inf. Sci.*, vol. 632, pp. 833–857, Jun. 2023.

[127] Regulation (EU) 2022/868 of the European Parliament and of the Council of 30 May 2022 on European data governance and amending Regulation (EU) 2018/1724 (Data Governance Act) (Text with EEA relevance), PE/85/2021/REV/1, 2022.

[128] Proposal for a Regulation of the European Parliament and of the Council Laying down harmonised rules on Artificial Intelligence (Artificial Intelligence Act) and amending certain Union Legislative Acts, COM/2021/206 final, 2021.

[129] N. Díaz-Rodríguez, J. Del Ser, M. Coeckelbergh, M. López de Prado, E. Herrera-Viedma, and F. Herrera, “Connecting the dots in trustworthy artificial intelligence: From AI principles, ethics, and key requirements to responsible AI systems and regulation,” *Inf. Fusion*, vol. 99, no. C, p. 101896, Nov. 2023.

M. Victoria Luzón is currently a Full Professor in the Software Engineering Department at the University of Granada, Spain. She is a Member of the research group “Soft Computing and Intelligent Information Systems” and of the Andalusian Interuniversity Institute in Data Science and Computational Intelligence (DaSCI Institute). She received the B.Sc. degree in computer science from the University of Granada in 1992. She received the Ph.D. degree in industrial engineering from the University of Vigo in 2001. Her main research interests include sentiment analysis, natural language processing and the study of novel learning techniques such as federated learning.

Nuria Rodríguez-Barroso received the B. Sc. degrees in computer science and mathematics from the University of Granada in 2018 and the M. Sc. degree in computer science from the University of Granada in 2019. Finally, he received the Ph.D. degree in computer science from the University of Granada in 2023. She is actually working as a Posdoc in the University of Granada. Her main research interests include adversarial attacks and defences in federated learning as well as other applications of federated learning. She is also interested in natural language processing, sentiment analysis and trusworthy AI.

Alberto Argente-Garrido received the M.Sc. degree in computer science from the University of Granada, Spain, in 2018, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Artificial Intelligence. His current research interests include federated learning, decision trees, explainability and data science.

Daniel Jiménez-López is a Ph.D. candidate in the University of Granada working in the development of federated learning platforms in Python, as well as in the development of techniques to safeguard the privacy and integrity of deep learning models. Consequently, differential privacy and federated learning are him main research fields.

Jose M. Moyano received the Ph.D. degree in computer science from the University of Córdoba (Spain) and Virginia Commonwealth University (USA) in 2020. He is currently a Postdoctoral Researcher with the Department of Computer Science and Artificial Intelligence at the University of Granada. His research interests include the development of multi-label learning models, specifically based on ensembles, as well as federated learning.

Javier Del Ser (Senior Member, IEEE) defended the first doctoral thesis (Cum Laude) in control engineering and industrial electronics at the University of Navarra (2006), and the second doctoral thesis in information and communication technologies (also Cum Laude and awarded the Extraordinary PhD Award) at the University of Alcala de Henares (2013). He is a Research Professor in artificial intelligence at TECNALIA RESEARCH & INNOVATION, and an Adjunct Professor at the Department of Communications Engineering of the University of the Basque Country (UPV/EHU). He is also a distinguished Professor at the University of Granada (Spain). His research interests are in artificial intelligence, machine learning and deep learning applied to practical modeling and optimization tasks for problems arising from different sectors, including industry, health, telecommunications, transportation, energy, and mobility, among others. He has authored more than 430 scientific contributions to date, including 180 JCR-indexed journal articles. He has supervised 16 doctoral theses and participated in more than 50 projects and contracts. He has been listed within the top 2% most influential AI researchers worldwide by the Stanford University and has also been part of the team that developed the R&D strategy in Artificial Intelligence for the Government of Spain in 2019.

Weiping Ding (Senior Member, IEEE) received the Ph.D. degree in computer science from Nanjing University of Aeronautics and Astronautics, in 2013. From 2014 to 2015, he is a Postdoctoral Researcher at the Brain Research Center, National Chiao Tung University, Hsinchu, Taiwan, China. In 2016, He was a Visiting Scholar at National University of Singapore, Singapore. From 2017 to 2018, he was a Visiting Professor at University of Technology Sydney, Australia. He is a Full Professor with the School of Information Science and Technology, Nantong University, and also the Supervisor of Ph.D postgraduate by the Faculty of Data Science at City University of Macau, China. His main research directions involve deep neural networks, multimodal machine learning, and medical images analysis. He ranked within the top 2% Ranking of Scientists in the World by Stanford University (2020–2023). He has published over 250 articles, including over 100 IEEE Transactions papers. His eighteen authored/co-authored papers have been selected as ESI Highly Cited Papers. He has co-authored four books. He has holds 28 approved invention patents, including two U.S. patents and one Australian patent. He serves as an Associate Editor/Editorial Board member of *IEEE Transactions on Neural Networks and Learning Systems*, *IEEE Transactions on Fuzzy Systems*, *IEEE/CAA Journal of Automatica Sinica*, *IEEE Transactions on Intelligent Transportation Systems*, *IEEE Transactions on Intelligent Vehicles*, *IEEE Transactions on Emerging Topics in Computational Intelligence*, *IEEE Transactions on Artificial Intelligence, Information Fusion, Information Sciences*, *Neurocomputing*, *Applied Soft Computing*. He is the Leading Guest Editor of Special Issues in several prestigious journals, including *IEEE Transactions on Evolutionary Computation*, *IEEE Transactions on Fuzzy Systems*, and *Information Fusion*. Now he is the Co-Editor-in-Chief of both *Journal of Artificial Intelligence and Systems* and *Journal of Artificial Intelligence Advances*.

Francisco Herrera (Senior Member, IEEE) received the M.Sc. degree in mathematics in 1988, and the Ph.D. degree in mathematics in 1991, both from the University of Granada, Spain. He is a Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada and Director of the Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI). He's an Academician in the Royal Academy of Engineering (Spain). He has been the Supervisor over 60 Ph.D. students. He has published more than 600 journal papers, receiving more than 144 000 citations (Scholar Google, H-index 179). He has been nominated as a Highly Cited Researcher (in the fields of computer science and engineering, respectively, 2014 to present, Clarivate Analytics). He acts as Editorial Member of a dozen of journals. His current research interests include among others, computational intelligence, information fusion and decision making, trustworthy artificial intelligence and data science (including data preprocessing, prediction and big data).