DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

A Systematic Comparison of Federated Machine Learning Libraries

Ahmed Saidani

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

A Systematic Comparison of Federated Machine Learning Libraries

Ein Systematischer Vergleich von Federated Machine Learning Bibliotheken

Author:	Ahmed Saidani
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M. Sc. Tobias Müller
Submission Date:	January 15, 2023

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.01.2023 Ahmed Saidani

Acknowledgments

I would like to express my deepest thankfulness to everyone who supported me during this project directly or indirectly. First and foremost, I would like to thank Prof. Dr. Florian Matthes for all the courses that the sebis chair offer. I participated in many of them during my master's degree and I learned a lot. They made me a better computer scientist as well as helped me prepare for my future career. I would also like to thank him for the supervision of this thesis. Additionally, I would also like to express my deepest gratitude to my Advisor M. Sc. Tobias Müller. I had the chance to work with him for over a year both on my thesis and on a practical lab course. He has been an excellent mentor for me. He was always there for me whenever I needed help or guidance, he offered valuable advice when needed, and motivated me throughout the thesis.

I would also like to thank my manager at CGI, Benjamin Bethke, who has been a great mentor for me in the last two years, as well as, CGI for sponsoring this thesis and making it possible for me to focus on delivering it. Furthermore, I would like to extend my heartfelt thanks to the interviewees for sharing their valuable time and experience with me. Moreover, I would like to thank all the TUM staff and the members of the sebis chair for working so hard to assure that I and my fellow students have the highest quality education possible.

Lastly, I would like to thank my friends and family for emotionally and mentally supporting me for the last six years. This journey has been a rollercoaster ride for me and I wouldn't be anywhere near I am today without their support. With that said, this thesis marks the end of a long journey during which I had the most enriching experience of my life. I am excited to see what journey I will embark on next.

Abstract

In the last few years, consumers have become more than ever aware of their data sovereignty and privacy. In addition, companies need more and more to collaborate but are also careful with sharing their sensitive data. That created a bottleneck for the training of machine learning models that typically needed a lot of data to be trained. Which created the need for privacy-enhancing machine-learning technology like federated machine learning, a new machine learning technique that promises to solve the privacy issues by training the machine learning models on the client side instead of in a central server. Federated machine learning has been growing in relevance in academia but not in the industry. One of the main reasons is that the field is still lacking structure and the infrastructure that will accommodate its adoption. For instance, there are multiple federated machine learning libraries currently available but most of them lack the documentation that explains how to use them. They also lack a lot of features and functionality necessary for building federated learning systems, and they are not ready to be used in production.

This thesis aims to identify the different characteristics of existing federated learning libraries and evaluate their suitability for an industrial application. First, a series of interviews with industry experts will be conducted to understand their needs. Then, a literature review of scientific literature, as well as the official documentation of the libraries and their features will be done to be able to compare them. And finally, a benchmarking tool that compares the non-functional characteristics of the libraries will be developed and demonstrated. By the end of thesis, an overview of the Federated learning community preferences, a comprehensive list of the libraries, their features, and non-functional differences, as well as a benchmark suite will be delivered.

Contents

1. Introduction

The lack of training data is often a bottleneck for ML applications. Problems are getting more complex and solving higher complexity requires more training data. In centralized ML, data needs to be collected from different data points. Training is conducted in a central server. This centralization introduces a multitude of problems. Participants need to share their data with a third party. It makes them lose sovereignty over their data and intellectual property. Training these large models requires also large data centers and large communication bandwidth. Federated learning emerged as a solution to this bottleneck. In federated learning, the central server doesn't train the model with the data collected from the clients but instead chooses a machine learning model and distributes it to the different clients. The clients would then train that model locally using their data and then send a gradient update to the server that reflects the updates made to the client-side model during the training. The server then uses a federated learning aggregation algorithm (e.g FedAvg) to combine the gradient updates received from the different clients to update the global model [12]. This process provides a certain level of privacy by design and keeps the sovereignty of the data on its owner's side since the model will be trained without the data leaving the devices of its owner. This solves the privacy issues of traditional machine learning systems [12].

Since the introduction of Federated Learning by Google researchers in 2016 [12], academics and engineers alike have been discussing the immense potential of the technology and its multiple use cases. This potential might be helpful, especially in privacy-critical industries which work with private data of customers like Finance, Insurance, or Healthcare. In these industries, participants tend to be more careful with who they share their data with, since any data breach could result in a well-being or financial loss. Despite the potential use cases of the technology, real applications have rarely been seen. This is due to many reasons but mainly to the complexity of the development of these systems. This need for a less complex FL systems development has been recognized and some of the biggest tech companies started developing FL libraries. To name some, Google published TFF [52] as part of the Tensorflow Framework, IBM published IBM federated learning [65] and Webank published FATE [56]... This has given FL practitioners a wide choice but also made it harder to pick a library for their project. Since the library differed in terms of functionality, for instance in terms of supported machine learning models, and in terms of quality, for instance, in terms of speed of execution. The scientific literature about federated learning is mainly about the bottlenecks of the technology and its open problems [12]. For instance, the involvement of thousands of clients can lead to scalability and communication issues, and the training of the data on the client-side makes the client responsible for their own data security. Even though there have been some efforts to compare the FL libraries, these comparisons either don't go into

the details of the differences other than the accuracy and the performance or were specific to a use case (e.g IoT) or the choice of the compared libraries wasn't explained. The scientific literature still lacks a structured comparison of FL libraries.

This thesis aims to compare the FL libraries qualitatively and quantitatively in a structured and well-rounded manner. We conduct a qualitative comparison by opposing the functionalities and features of each library. Also, we compare the quantitative capabilities by benchmarking non-functional requirements such as scalability, performance, and efficiency. Furthermore, this thesis aims to provide a benchmarking suite to be used in the future and to reproduce its results. We aim to accomplish this structured comparison by answering the following three research questions (RQs):

RQ1: What are the functional and non-functional requirements relevant for a federated learning library, and what are the most important metrics to benchmark them?

RQ2: What are the different federated libraries available, and how do they differ in terms of functionality?

RQ3: How could a modular software application that benchmarks the different federated learning libraries using the metrics be developed?

2. Background

2.1. Machine Learning

Machine Learning is a subdiscipline of artificial intelligence. To be able to define what machine learning is, the definition of artificial intelligence needs to be established first. The term artificial intelligence was coined by Professor John Mccarthy in 1955 [1]. He defined it as "The science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable." As such, intelligent beings should be able to complete tasks that typically require cognitive intelligence.

Learning from past experiences is a characteristic of cognitively intelligent animals such as humans and the same tasks that require cognitive intelligence to be completed, also require the ability to learn and draw conclusions from past experiences [2]. The parallel to that in machines is dubbed "Machine Learning". Machine Learning is the ability of a machine to gain knowledge and understanding from data. It covers a wide range of methods and processes for designing and implementing algorithms based on statistical models that enable machines to draw conclusions from data, as well as, predict outcomes of certain events based on the patterns discovered in that data. The main premise is that the more a machine learning system is trained the better its performance will be [2].

Multiple machine-learning techniques have been developed throughout the years. Most of them fall into one of three categories, reinforcement learning in which agents learn to interact with the data by rewarding certain behavior and punishing others (e.g DQN and DDPG), unsupervised learning which is used to explore the data (e.g clustering and dimensionality reduction), or supervised learning which is used to predict future outcomes based on the data (e.g regression and classification)[3]. Only supervised and unsupervised learning techniques are relevant to this thesis. Figure 1 illustrates the taxonomy of Machine Learning that are relevant.

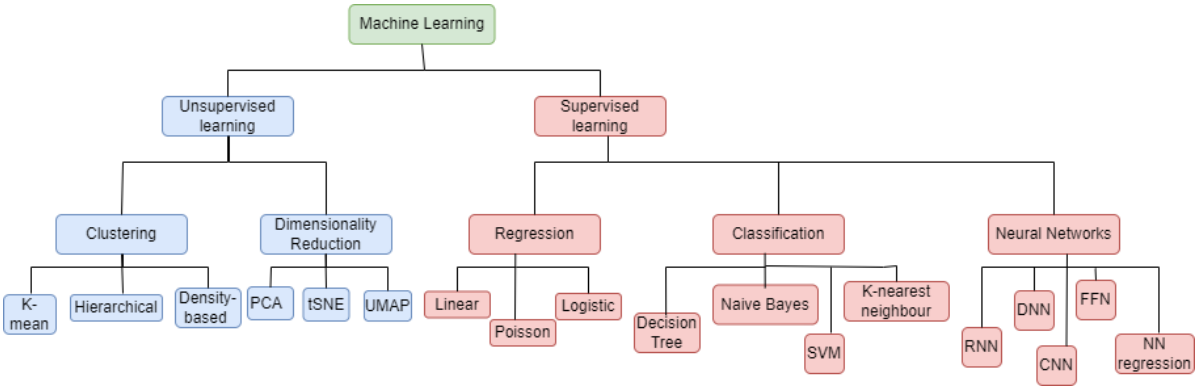


Figure 1.: Taxonomy of the different Machine Learning methods (source: own work, based on [3]).

2.1.1. Foundations of Machine learning

Machine learning systems use stochastic methods to fulfill their functions. To explain the difference between the different types of machine learning (supervised and unsupervised learning) and what federated learning is. Also, some statistics and machine learning terminology need to be defined. For instance, Machine learning models train on instances of labeled and unlabeled data by optimizing their loss function to make predictions. The data instances have multiple features across different dimensions. To improve the predictions of a Machine Learning model, the parameters of the model could be configured. Thus terms like a model, a parameter, a feature, an instance, a dimension, labeled and unlabeled data, and loss (cost) function also need to be defined [7]:

- **Model** is the output of the machine learning algorithm after being trained on its given data. A model is expected to be given input data and gives back an output which is a prediction based on the patterns it discovered while being trained on that data [7].
- **Dataset** is a collection of data that represents the observations captured on a certain population. The rows of the table are called instances, while the columns are called features. The data in a given dataset could be either labeled or unlabeled [7].
- **Attributes** is an umbrella term for all the variables presented in a dataset. Features and Instances are both considered attributes [7].
- **Parameter** is a variable internal to the model that can be configured to make the predictions of that model more accurate. The parameter is configured automatically by the model when being trained [7].
- .**Dimensions** are used to describe the number of features in a dataset. For instance, a dataset that has n features is a n-dimensional dataset and can be represented on an n-dimensional space with n coordinates [7].

- **Instance** is an observation of a specific object within the population that constitutes the dataset [7].
- **Feature** is a measurement of an attribute for a specific instance in the database [7].
- **Labeled data** is data that comes with tags that help identify some of its characteristics, properties, and classifications. It is typically used in supervised learning to make predictions [7].
- **Unlabeled data** is data that comes without tags. It is typically used in unsupervised learning to find patterns [7].
- **Loss function/Cost function** is a mathematical tool used to evaluate how accurate a machine learning model is in predicting certain outcomes. The goal of machine learning models typically is to minimize their loss (cost) function and to give the most accurate prediction possible based on the given dataset [7].

2.1.2. Unsupervised Learning

When data is unlabeled, unsupervised learning techniques are used to identify patterns in the data. Unsupervised learning techniques can be leveraged to explore the data without labeling it. It is not as complex and does not require as much effort to set up as supervised learning techniques since the labeling is usually done by humans which makes it more cost and time intensive. However, unsupervised learning is less effective at predicting future outcomes [3].

Clustering is done by grouping the different data points of a dataset into multiple cohorts to explore the data [3]. Data points are grouped into patterns of data based on their distance from each other (e.g Euclidean distance) in an N-dimensions space. These patterns are dubbed clusters [2]. There are multiple clustering techniques used in machine learning. K-mean, Hierarchical, and Density-based clustering are among the most used [3]. Figure 2 illustrates an exemplary set of clustered data points.

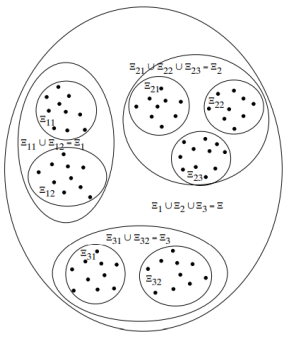


Figure 2.: Hierarchy of Clusters (source: [2]).

K-mean Clustering: The user selects a parameter K that presents the number of clusters. K points are randomly given as the clusters' centers. Then a two-phase algorithm starts:

1. Assignment: Data points are assigned to the cluster based on their Euclidean distance from the center of the clusters. Each data point is assigned to the closest center.
2. Center shift: The coordinates of the cluster centers are adjusted by calculating the mean of all the coordinates of the data points assigned to a given cluster. The result of that is the new cluster center.

The algorithm iterates through these two phases until the coordinates of all the cluster centers are unchanged [3].

Density-based Clustering: In this clustering technique, clusters are defined as the high-density areas of data points in a feature space. That means the areas with a high density of data points. Data points that do not belong to any high-density feature space are called noise and do not belong to any of the clusters. The benefits of this approach are that it is deterministic, it does not require any parameters from the outside, and it produces complex shapes of clusters [3].

Hierarchical Clustering: The objective of hierarchical clustering is to build a hierarchy of clusters. That could be done through multiple algorithms. One of which is neighbor-joining. In neighbor joining the user selects a threshold for the joining of clusters. Then a two-phase algorithm starts:

1. Distance calculation: The Euclidean distance between all points is calculated.
2. Data points Joining: The two closest points are joined together (clusters can also be joined).

This is repeated iteratively until the threshold set by the user is reached [3].

Dimensionality reduction techniques used to reduce the number of dimensions or features of a dataset while preserving the variability and the distance between the data points in that dataset. Datasets may have thousands of features and dimensions which can create a lot of noise and make the analysis and visualization of the data as well as the testing of the machine learning models challenging. Dimensionality reduction promises to solve exactly these two problems. Some of the dimensionality reduction techniques are Principal Component Analysis (PCA), t-distributed stochastic neighbor embedding (tSNE), and Uniform Manifold Approximation and Projection (UMAP) [3]. They all operate differently but they are all used to reduce the dimensionality of the data for it to be more easily interpreted at a later stage.

2.1.3. Supervised Learning

A machine learning process is called "supervised" if the model is optimized to make future predictions based on labeled data. There are multiple supervised learning techniques (e.g Linear regression, Naive Bayes, and decision trees). The algorithm and model choice depends mostly on the labeling of the data. For instance, for numerical data, regression techniques

like linear regression or logistic regression are well suited. For categorical data, classification techniques like Naive Bayes and decision trees are used [3].

Regression is a technique that is used to study the relationship between variables. The cause variable (independent variable) is denoted X and the effect variable (dependent variable) is denoted Y. The relationship is denoted as Y in the function of X or$f(X) = Y$. A function could also have multiple independent variables $(X1, X2 \ldots Xn)$. In this case, the relationship is denoted as Y in the function of $X1, X2, \ldots Xn$ as $Y = f(X1, X2 \ldots Xn)$. Examples of regression techniques are linear regression and logistic regression [4].

Linear regression is a type of regression that is used to analyze the relationship between the independent and dependent variables that have a linear correlation. It can be denoted as:

$$Y = \beta 0 + \beta 1X1 + \beta 2X2 + \beta 3X3 + ... + \xi x$$

$Y, X1, X2, X3 \ldots$ are continuous variables. $\beta 0, \beta 1, \beta 2, \beta 3, \xi x \ldots$ are constants. Y is the dependent variable, $X1, X2, X3 \ldots$ are the independent variables that influence Y. $\beta 0, \beta 1, \beta 2, \beta 3 \ldots$ are the coefficients of $X1, X2, X3 \ldots$, $\beta 0$ being the value of Y when all the independent variables are equal to 0. and ξ x is a set of random errors. The graphical representation of the linear regression is a straight line. It is used to predict and forecast the value of Y given $X1, X2, X3 \ldots$ [5]

Logistic regression: Unlike linear regression, logistic regression predicts the probability of an event occurring instead of a value. Instead of an independent variable, it has a predictor and instead of a dependent variable, it has a response variable [5]. The probability of an event occurring is denoted as follows:

$$Pr[Y|X] = p(x) = \frac{e^{(\beta 0 + \beta 1X)}}{1 + e^{(\beta 0 + \beta 1X)}}$$

$Pr[Y|X] \in [0, 1]$ is the probability of Y occurring given X. $\beta 1$ and $\beta 2$ are constants, with $\beta 1$ being the coefficient of X [5]. The odds of an event occurring are the following [5]:

$$e^{(\beta 0 + \beta 1X)} = \frac{p(x)}{1 - p(x)}$$

Figure 3 illustrates the graphical representation of a linear regression model (a) and a logistic regression model (b).

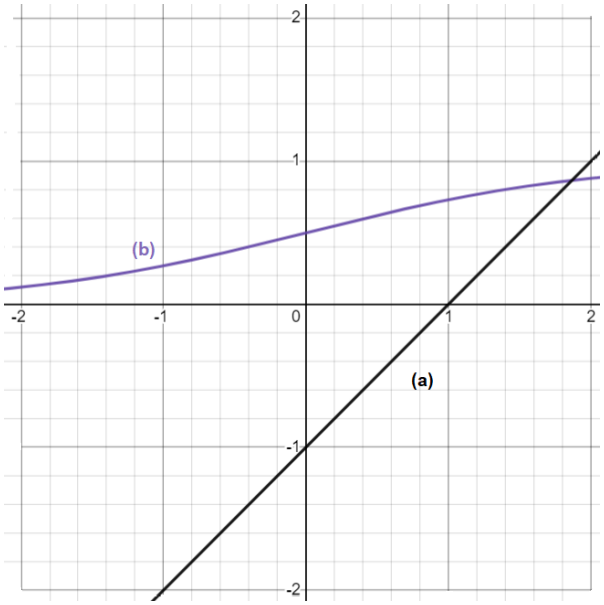


Figure 3.: Graphical representation of a linear model (a) and a logistic model (b) (source: own work).

Classification is a technique that is used to determine the membership of a certain instance in a group based on its features. Some of the classification methods are decision trees, Naive Bayes, K-nearest neighbors, and support vector machines [6]. Figure 4 illustrates the graphical representation of the different classification models. Namely, the decision tree, Naive Bayes, support vector machine, and K-nearest neighbor.

Decision Tree is a classification tool that organizes features and the decisions associated with them in a tree-like structure. It has a root node that presents the entire dataset and then it splits it based on the features into sub-nodes. Then, the sub-nodes are further split into other sub-nodes based on the other features until, in the end, each leaf (terminal node) results in a decision. The process of dividing the nodes is called splitting and the process of merging the nodes is called pruning. Figure 4a illustrates a simplified example of a graphical representation of a decision tree that determines whether someone is healthy or not [8].

Naive Bayes is a Bayesian network. Bayesian networks are probabilistic graphic models that present knowledge about uncertain events. The events are presented as nodes and the probability of the occurrence of an event e2 knowing an event e1 are presented as directed edges from the node of e1 to the node of e2 [8]. Naive Bayes is a Bayesian network. It has a posterior event h and a precedent event e. The goal is to determine the probability of h knowing e. This can be done using the following formula:

$$Pr[h|e] = \frac{Pr[e|h] \cdot Pr[h]}{Pr[e]}$$

Pr[h | e] is the probability of h knowing e, Pr[e | h] the probability of e knowing h, Pr[h] the

probability of h, and Pr[e] the probability of e. It could also have multiple precedent events ei that influence h. In that case, the following formula applies [8]:

$$Pr[h|e1, e2...en] = \frac{Pr[e1, e2...en|h] \cdot Pr[h]}{Pr[e1, e2...en]}$$

Figure 4b illustrates an example of a graphical representation of a Naive Bayes.

SVM classifies the instances of a dataset in an n-dimensional space. Each dimension represents a feature. The features of the data point are represented as the coordinate of the data point in the n-dimensions. SVM runs an algorithm to find the best possible plain that separates the different elements into categories, by maximizing the distance between the points of each pair of categories (called support vectors). The distance between the support vectors is called the margin. Figure 4c illustrates an example of a graphical representation of an SVM in a two-dimensional space [8].

K-nearest neighbor is a classification algorithm that is used to assign a given data point to one of the given classes of data points in a dataset in an n-dimensional space. Each dimension represents a feature of the dataset and the values of features of the data point are represented as their coordinates in the n-dimensions. The data points are scattered around the space. The classes used to classify the data points are given. The KNN algorithm calculates the euclidian distance of the given data point to all the other data points in the space. Then, it sorts the distances between the data points of the dataset and that given data point in increasing order. A value K is determined beforehand, and the occurrence of the classes of the K first data points are counted. The data point is then assigned to the class with the most occurrences. Figure 4d illustrates an example of a graphical representation of a K-nearest neighbor in a two-dimensional space [8].

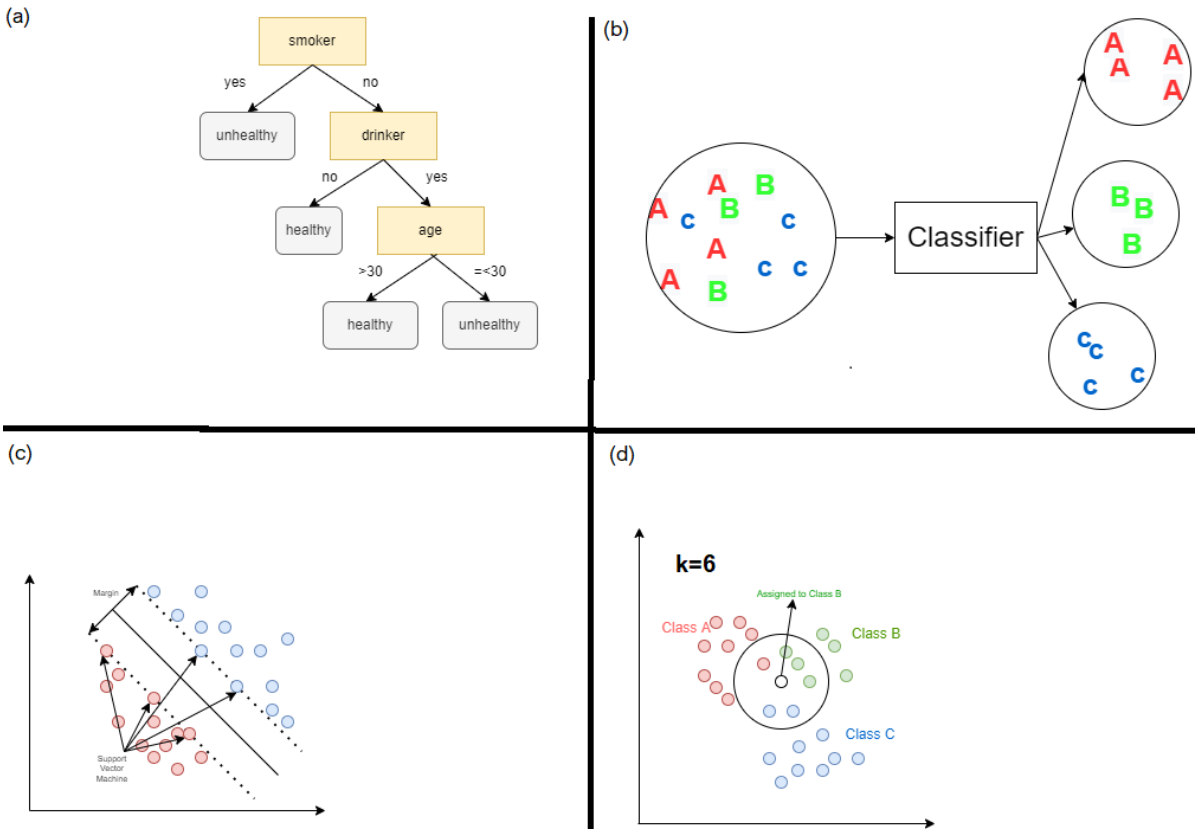


Figure 4.: Graphical representation of (a) Decision tree, (b) Naive Bayes, (c) SVM, and (d) K-nearest neighbor (source: own work).

Artificial neural networks are a subset of machine learning algorithms that are mostly used for classification. They are used to perform tasks inspired by the human brain like pattern recognition or making sense of something that requires a large amount of information. That is achieved by mimicking the neuron structure in the human brain and organizing it in a directed graph structure where the nodes take the role of neurons and the edges take the role of the connections between the neurons [9]. The input data is given to the first layer of nodes, called the input layer, and then forwarded to the next layer of nodes through the edges. Each node performs a set of calculations and then forwards them to the nodes in the next layer recursively until reaching the last layer. The last layer, called the output layer gives the final output of the calculations [9].

Each node has input data (xi), output data (y), a threshold/bias (b), and weights (wi). The input data is the data that a node receives either from the outside in the case of the input layer nodes or from the nodes of the previous layer in the case of other nodes. The output data can be the value that is output at the end in the case of the output layer node or the value that a node forwards to the next layer nodes in the case of other nodes. The weight determines by how much the signal that a node receives needs to be amplified. The

threshold determines the minimum value that a node needs to receive to be activated and do the required calculations [9]. Analogical to the human brain, the neural network also improves its accuracy with practice, the more an edge is used in the training of the neural network, the more its weight w will increase. Its importance in the network will grow as a consequence which will result in an increase in accuracy over time [9]. Figure 5 illustrates an example of a neural network.

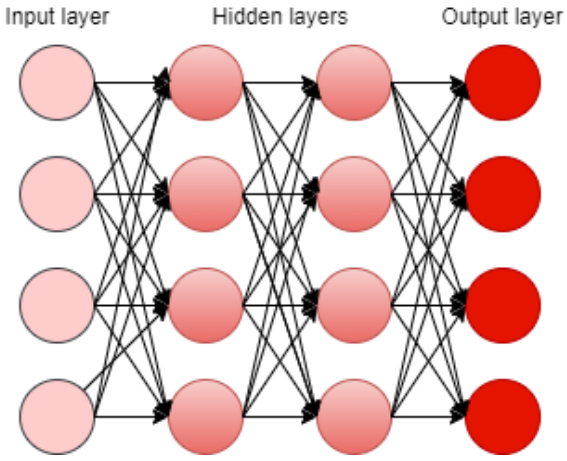


Figure 5.: Graphical representation of a neural network (source: own work)

The input data that get fed to a node typically has a form of a vector v = (x1,x2, ..., xn). with x1,x2, ..., xn being the values of the different inputs. Each of the input data variables has a weight. The weights w1,w2, ..., wn are assigned to the input variables x1,x2, ..., xn. The weights, change dynamically over time during the training of the model. Finally, the threshold b is a constant. Each node has its linear regression model to calculate its output data in the following form:

$$y = w1x1 + w2x2 + ... + wnxn + b$$

Each neural network model has a cost function associated with it and the goal of the training of the neural network is to minimize that cost function by constantly changing the weights and biases of each node. During the training, the loss function will converge to a local minimum. Gradient descent is the algorithm that is typically used to minimize the cost function [9]. There are multiple types of neural networks each with a different variation of how they function and their own set of characteristics with the main ones being Deep neural network (DNN), Feedforward neural network (FNN), Convolutional neural network (CNN), Recurrent neural network (RNN).

Deep neural network (DNN): When a neural network has one or multiple hidden layers in addition to its input and output layers, it is called a deep neural network. Deep neural networks are the foundation for deep learning, they are more suited for more complex tasks

like speech or image recognition. Figure 5 is an illustration of a deep neural network [9].

Feedforward neural network (FNN): When a neural network allows information to propagate only in a forward direction and contains no cycles or loops, it is called a Feedforward neural network. It is the most basic form of neural network. A feedforward neural network can be either a deep neural network or not depending on the number of layers it has. Occasionally, information is allowed to propagate in the opposite direction in a process known as backpropagation. Feedforward neural networks are typically used in speech and image recognition, as well as pattern recognition. Figure 5 is an illustration of a feedforward neural network [9].

Backpropagation is an algorithm that is used to calculate the derivative (the gradient) of the loss function between the different layers in the neural network. It is also used to adjust the weights of the different inputs of the nodes in that layer to minimize the loss function. This process is done iteratively until no further improvement can be done. Since this information (The value of the loss function after the input is passed to the next layer) is retrospective, the feedback needs to be sent in a backward manner, which gives the algorithm its name [9].

Convolutional neural networks (CNN) are feedforward neural networks that use linear algebra techniques to process imagery, mainly in the fields of computer vision and image recognition. CNN uses a mathematical operation called convolution to filter and map the data in an image. In addition to the classical layers in feedforward neural networks, which are called fully connected layers in the context of CNN. Convolutional neural networks have two additional layers. Namely, a convolution layer to recognize the image features in pixels, and a pooling layer to abstract these features in a way that a model could understand. The convolution layer typically comes first in CNN, then a pooling layer, and at the end, the fully connected layers [10]. Figure 6 illustrates how Convolutional neural networks work.

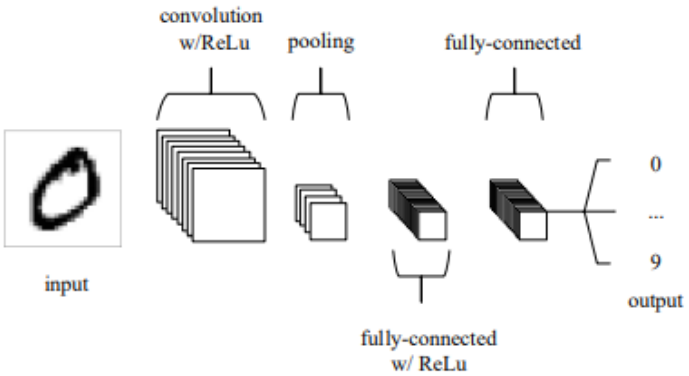


Figure 6.: A simple CNN architecture, comprised of just five layers (source: [10]))

Recurrent neural networks (RNN) are mostly used in speech recognition or natural language processing since they are well suited to processing time-series and sequential data. They

are different from feedforward neural networks and convolutional neural networks in their functioning mechanism. The nodes in RNN do not only take input from the nodes of the previous layers but also store the previous input given to them by these nodes in the previous iteration. Then, they combine these two inputs to make their prediction. This way, their prediction doesn't only depend on current inputs but also their past inputs. RNN has some key functionalities specific to them like using a variation of the backpropagation algorithm called "backpropagation through time" to adjust their weights. Also, all nodes across a particular layer share the same weight parameters [9].

2.2. Federated Learning

The lack of training data is often a persisting bottleneck of a successful machine learning application. However, in some use cases like medical research (e.g predicting cancer risk), or finance (e.g predicting the credit rating of someone), it can be challenging to get individuals to consent to their health or financial data is gathered and used, in a way, they do not understand. Individuals and companies are not willing to share their intellectual property or sensitive data if there is a risk of losing sovereignty over their data [11]. Especially with the introduction of laws such as GDPR. This inhibits a willingness for voluntary data sharing, which in turn can cause multiple bottlenecks when training the data like the availability of data itself, or a sampling bias with people who are willing to participate are not representative of the entire population. Federated Learning emerges as the solution to these systematic problems of machine learning. It is a practical solution to train machine learning models, all while preserving the privacy of the subjects participating. That is achieved by following a compute-to-data approach and training the model on the client side, therefore alleviating the need to share and centralize the data [11].

2.2.1. Federated Machine Learning Mechanism

In traditional machine learning, the clients produce the data and send it to the server to train the model. Figure 7 illustrates how traditional machine learning systems train their models. At first, the server collects the data from the clients, then it uses the data to train the model, and in the end, it sends the output to the clients [12].

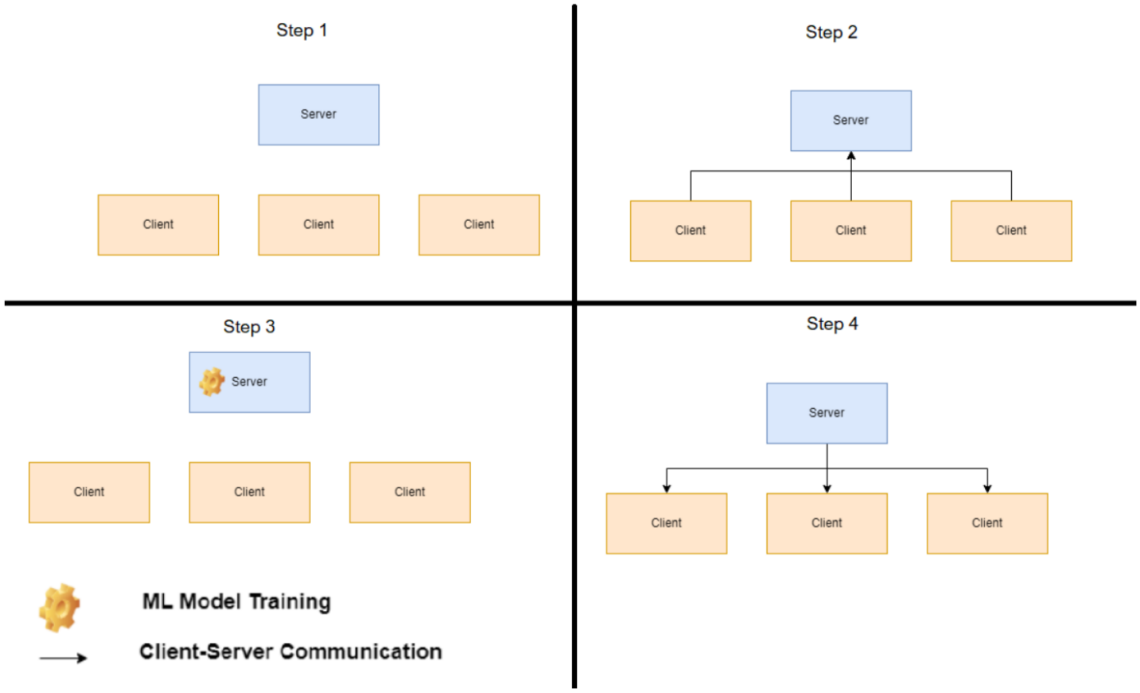


Figure 7.: Model Training in traditional Machine Learning (source: own work).

In federated machine learning, the server decides on a model and selects the clients. The clients will then receive the model and train it using their data on the client side without giving away any of the data to the server. Once the model is trained, the output of the training is then sent to the server. The server then aggregates the different updates received from the different clients using a federated averaging algorithm like FedAvg and produces a final combined gradient update, then the final gradient update gets sent to the different clients. The clients now have the updates of the global model without accessing each other's data [12]. Figure 8 illustrates the process of model training in a federated learning setting.

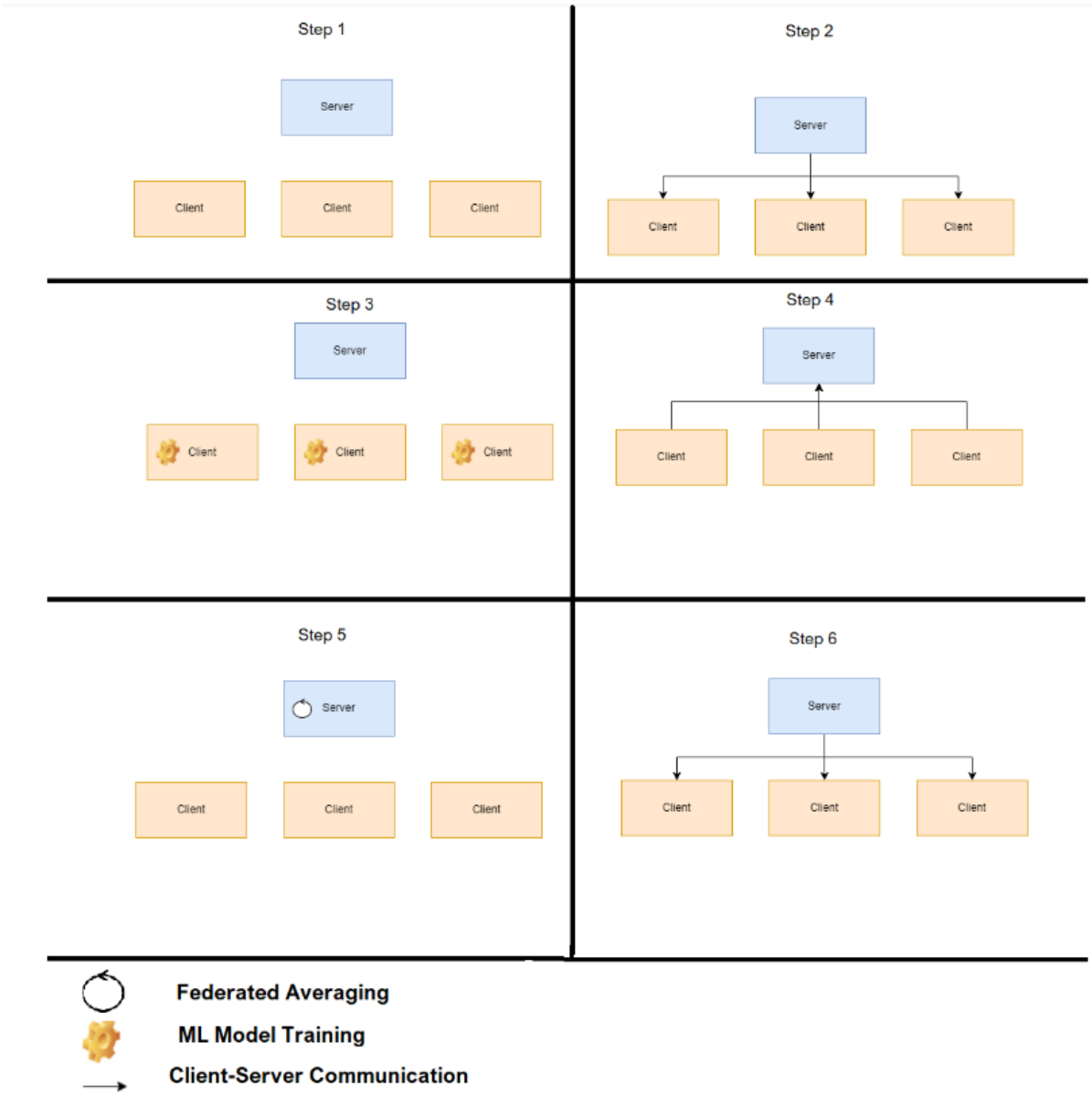


Figure 8.: Model Training in Federated Machine Learning (source: own work).

2.2.2. Types of Federated Learning Systems

Federated learning systems could be categorized across different dimensions. For instance, the distribution of the system, the partitioning of the data, the setting of the system, and the model deployment strategy. There are also several other particular types of federated learning systems like systems that use techniques like split learning or federated transfer learning (FTL).

Distribution of the system: Federated Learning systems can be categorized based on their distribution architecture. A federated learning system can be either centralized or decentralized.

Centralized Federated Learning systems: The central server orchestrates the implementation of the federated learning algorithm. It chooses the model and selects the clients for the training, then once the training is complete it aggregates the gradient updates from the clients and updates them again. The example illustrated in Figure 8 is a centralized federated learning system [14].

Decentralized Federated Learning systems: In decentralized federated learning systems, there is no central server to orchestrate the implementation of the federated learning algorithm. In a decentralized federated learning system, the clients are organized in P2P networks [14].

Partition of data Federated Learning systems can be also categorized based on the partition of the data in the system. The data can be either partitioned vertically or horizontally.

Horizontal Federated Learning systems: In horizontal federated learning systems, data is partitioned horizontally. All the stored data share the same scheme, which means all the partitions store the same attributes of the data instances but not the same set of instances. Each client trains the model with its instances of data [12].

Vertical Federated Learning systems: In vertical federated learning systems, data is partitioned vertically. Each client has its unique data scheme. Clients share different attributes of the same data instances [12].

Federated learning systems setting: Federated Learning systems can be categorized based on their setting. A federated learning system can be either cross-silo or cross-device.

Cross-silo Federated Learning systems: The clients in a cross-silo federated learning setting are deployed on the cloud. Each instance represents a different organization. The number of clients is low (typically 2-100), but they have high computational power as well as storage capability. Therefore, the complexity of the deployed models as well as the data volume of each client is typically high. Cross-silo federated learning systems could be either horizontal or vertical [12].

Cross-device Federated Learning systems: The clients in a cross-device federated learning setting are typically consumer electronic devices (mobile phones but could also be IoT devices or wearables). The number of clients is high (up to tens of billions), but their computational power, as well as their storage ability, is limited. Therefore, the complexity of the deployed models as well as the data volume of each client is typically low. Cross-device federated learning systems are always horizontal federated learning systems [12].

Model deployment: Federated Learning systems can be also categorized based on their model deployment. A federated learning system can be either model-centric or data-centric.

Model-centric Federated Learning systems: Clients in a Model-centric Federated Learning system share the same model. The model is typically broadcasted by the central server. Most Federated learning systems are model-centric systems [15].

Data-centric Federated Learning systems: A new approach for federated learning systems. In a data-centric Federated Learning system, each client typically chooses the model to train. The clients train different machine learning models [15].

Split Learning Systems: In split learning systems, the training of the model is done at each client until a particular point is reached. Then the output of that training (called smashed data) is transferred to the next client or the central server for further computations. The raw data of each client is not shared across entities and the training of the models can be parallelized [12].

Federated Transfer Learning Systems (FTL): The combination of federated learning and transfer learning is referred to as federated transfer learning. Transfer learning systems are systems that use the knowledge gained in a certain domain to solve problems in other domains. It could be the case that in federated learning settings, different clients have different data. That means, they share neither the data instances nor the data attributes. However, it could be the case that there is an overlap in some instances, the model to be trained or features. In these use cases, Federated Transfer Learning systems can be used to transfer the knowledge gained from training the model on a client to train the models on other clients. There are three types of federated transfer learning systems. Namely, instance-based, feature-based, and model-based FTL systems [12].

Instance-based FTL: The clients can use the knowledge gained from the data instances of other clients to minimize their loss function and optimize their model without accessing the data of the other clients [13].

Feature-based FTL: In feature-based FTL, the clients can select the features of the data instances required for training their models based on the knowledge transferred to them from the other clients. This is done by transferring knowledge about the features without sharing the features themselves [13].

Model-based FTL: In model-based FTL, the clients benefit from sharing pre-trained machine learning models without sharing the data [13].

2.2.3. Federated Learning Systems Properties

Federated Machine learning systems differ from traditional machine learning systems in multiple properties. Such as the lifecycle of the models, or features specific to federated learning like privacy mechanisms and Federated learning aggregation algorithms.

Federated Learning Models Lifecycle The lifecycle of federated learning applications resembles the lifecycle of traditional machine learning applications but there are some key differences. For instance, there are some extra steps in the Federated Machine learning model lifecycle that do not exist in the traditional machine learning lifecycle like the client instrumentation or simulations. Some steps are done differently as well, like the training and evaluation of the models. They are typically done on the clients rather than the server in a federated learning setting. The steps of the federated learning lifecycle are [12]:

1. Problem identification: Define the problem to be solved with federated learning.
2. Client instrumentation: Load the clients with the required datasets for the model training.
3. Simulation prototyping: Simulate different variations of the model on different datasets.
4. Federated model training: The model is trained just like traditional machine learning. The only key difference is that it is trained on the client and not the server.
5. Model evaluation: After being trained, the models are evaluated based on the same metrics as traditional machine learning applications. Namely, performance, accuracy, and other metrics. The best models are then taken to the next step of the process which is the deployment of the models.
6. Deployment: In the end, the model needs to run live. The owner of the model decides on the launch process but typically it is not different than the launch process of traditional machine learning processes with heavy manual and A/B testing before taking the model live.

Federated Learning Aggregate algorithms: The gradient updates of the model's training of the different clients need to be combined. That's the task of federated learning aggregate algorithms. They combine the gradient updates of the different clients using different techniques. There are multiple federated learning aggregate algorithms. The most prominent ones are FedAvg and FedSGD.

FedAVG is the most commonly used federated averaging algorithm. In FedAvg, once a client finishes training its model. It sends a gradient update to the server. The global model is then updated at once using the updates from all clients combined [12].

FedSGD is a variation of stochastic gradient descent used to optimize deep learning models iteratively. A random set of clients are periodically selected by the server to share their

gradient updates with the server. The updates are then used by the global model iteratively [20].

Privacy Mechanisms: Since federated learning systems require a lot of communication between the clients and the server. The communication needs to be secured. In a federated learning setting, this is done primarily through two distinct methods. Namely, differential privacy and cryptographic methods.

Differential privacy works by describing a dataset its features and patterns without disclosing any information about the instances of the dataset. That way the privacy of the individuals in the dataset is insured while the required information is shared [21].

Cryptographic methods: Federated learning systems use different encryption methods like HE, RSA, or security multi-party communication. These methods ensure that the communication between the different clients, as well as the server, is secured [12].

2.2.4. State of Federated Learning Research

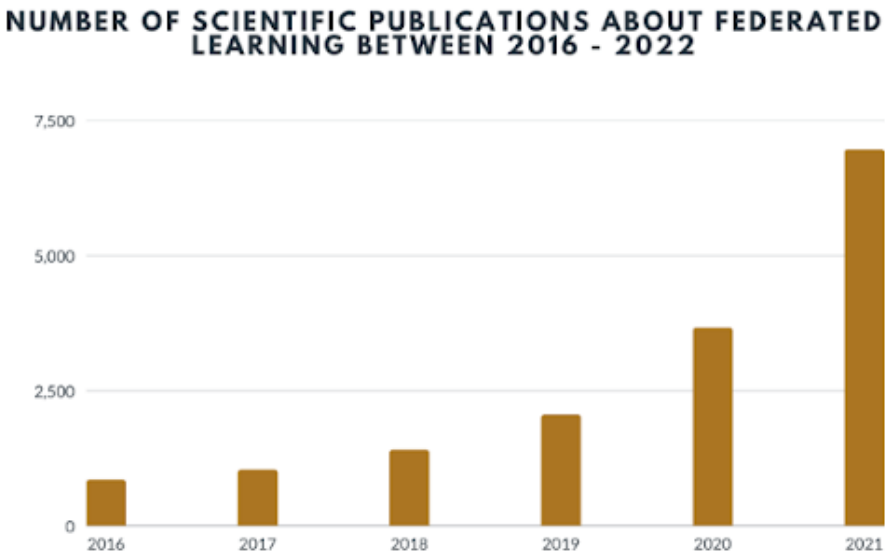


Figure 9.: Number of scientific publications with the term "federated learning" in their title (source: own work, data from [16])

In 2016, the term federated learning was introduced by Google [12]. Since then the interest in the topic grew considerably. According to the Scopus database [16], the number of scientific publications about federated learning has steadily grown from 840 publications in 2016

[16]Scopus: https://www.scopus.com/search/form.uri?display=advanced

to 6955 in 2021, as illustrated by figure 9 above [16]. Since the research field is still in its infancy, most of the research done in the field is exploratory. It is mostly focused on finding applications for federated learning, exploring the benefits and drawbacks of the technology, as well as prospecting the challenges posed by federated learning and trying to solve them.

Federated learning is a privacy-enhancing technology. Since the research field is still in a nascent stage, it found its first real-world applications in use-cases where privacy is critical like IoT, insurance, the medical field, and fintech [18]. Clients in federated learning systems never share their data with any other party, this enhances privacy. In a cross-device setting, the clients are heterogenous which could offer diverse data, and since the client has limited hardware capacities, federated learning models are designed for efficiency [19]. Federated learning systems require a lot of overhead investment compared to traditional machine learning systems (high bandwidth and a lot of storage). Performance is also a bottleneck since federated learning models tend to be slower than traditional machine learning models. While the system is designed to be decentralized, there is still a degree of centralization since a central server coordinates the different clients in the system [19]. For Federated learning to be widely adopted, it needs to tackle multiple systematic challenges like the handling of non-IID data, security and privacy concerns, communication cost, fairness, and systems heterogeneity [12]:

- Security: Federated learning systems are vulnerable to multiple types of attacks that can affect their performance and usefulness. Since federated learning systems tend to be larger than traditional machine learning systems, they come with an increased attack surface. Like machine learning systems, the model itself could be an attack vector (e.g data poisoning attack) but also since there is constant communication between the clients and the server, the information could be poisoned during the communication (e.g model update poisoning attack) [12].
- Privacy: Even though federated learning systems do not transfer the data of the clients to the server, they do transfer information about the model training. This can result in the leakage of sensitive data. Techniques like Secure multiparty computation and Differential privacy are used to enhance the privacy of federated learning systems [17].
- Communication cost: Federated learning systems could scale to incorporate tens of millions of devices. the global models in the servers need to constantly update and be updated by all these client devices. Thus, communication can constitute a major bottleneck for federated learning systems [17].
- Fairness: In federated learning settings, context plays a bigger role than in traditional machine learning settings. Contextual factors like geographic location, connection quality, and type of device all play a role in the selection of clients. This could result in systematic biases [12].
- Systems heterogeneity: Cross-device federated learning systems incorporate multiple different types of devices (different kinds of IoT devices, wearables, and mobile phones).

The different clients can for instance produce data with different schemes which can lead to decreased efficiency and biases in the system. The heterogeneity of the clients needs to be taken into account when designing the system [17].

2.3. Software Engineering Concepts

Software engineering is an engineering discipline that deals with the design and development of software applications. It uses a set of methods, tools, and techniques to help with the production of high-quality software systems [27]. Since the federated learning libraries are software systems and software benchmarking is a subfield of software engineering, some software engineering concepts like software libraries and frameworks, software requirements, and software benchmarking need to be introduced.

2.3.1. Software Libraries and Frameworks

Software libraries and frameworks are both pieces of software that are written to overcome a recurrent software engineering challenge so that developers do not have to resolve these problems over and over again, and instead focus on building the features of the application they are developing [26]. Despite the similarity, there are key differences between libraries and frameworks. Table 1 summarizes the differences between libraries and frameworks in the context of software engineering.

	Libraries	**Frameworks**
Call mechanism	The code calls the software libraries.	Frameworks work by inversion of call, which means that the framework calls the code.
Degree of Control	The developers have full control over the functions that they call from the libraries.	The framework calls the functions and methods for the developer. It also dictates the architecture of the project. Thus, the developer has a lesser degree of control over the methods and functions called in the project.
Integrability	Software libraries are easily integrated into existing software projects.	Frameworks can not be integrated into existing software projects. The project needs to be built from scratch with the framework instead.

Incorporation	A software library can not incorporate a framework. It only incorporates methods and functions.	A framework can incorporate many software libraries.
Examples	JQuery, Material UI, Bootstrap ...	Angular, React, VueJS ...

Table 1.: Key differences between software libraries and frameworks (source: [26])

2.3.2. Software Requirements

According to Wiegers, K., and Beatty, J. (2013) [22], software requirements are "a specification of what should be implemented. They are descriptions of how the system should behave, or a system property or attribute. They may be a constraint on the development process of the system." [22] That means requirements reflect the expectations of the different stakeholders of a software system by dictating the static structure as well as the dynamic behavior of that system. We distinguish two types of software requirements, namely functional requirements, and non-functional requirements.

Functional requirements describe what a software application offers in terms of features and what business rules the software adheres to. They describe the tasks that the software could accomplish under certain constraints (how the software functions) as well as the components that make up the software [22].

Non-functional requirements describe the characteristics of the system across different dimensions. They serve as a constraint on the system and thus need to be measurable [22]. Habibullah et al. (2021)[23] identified 36 non-functional requirements that are important for machine learning applications. Of these 36 non-functional requirements, 24 are important for the machine learning system operation. Figure 10 illustrates the NFR identified by Habibullah et al. (2021)[23].

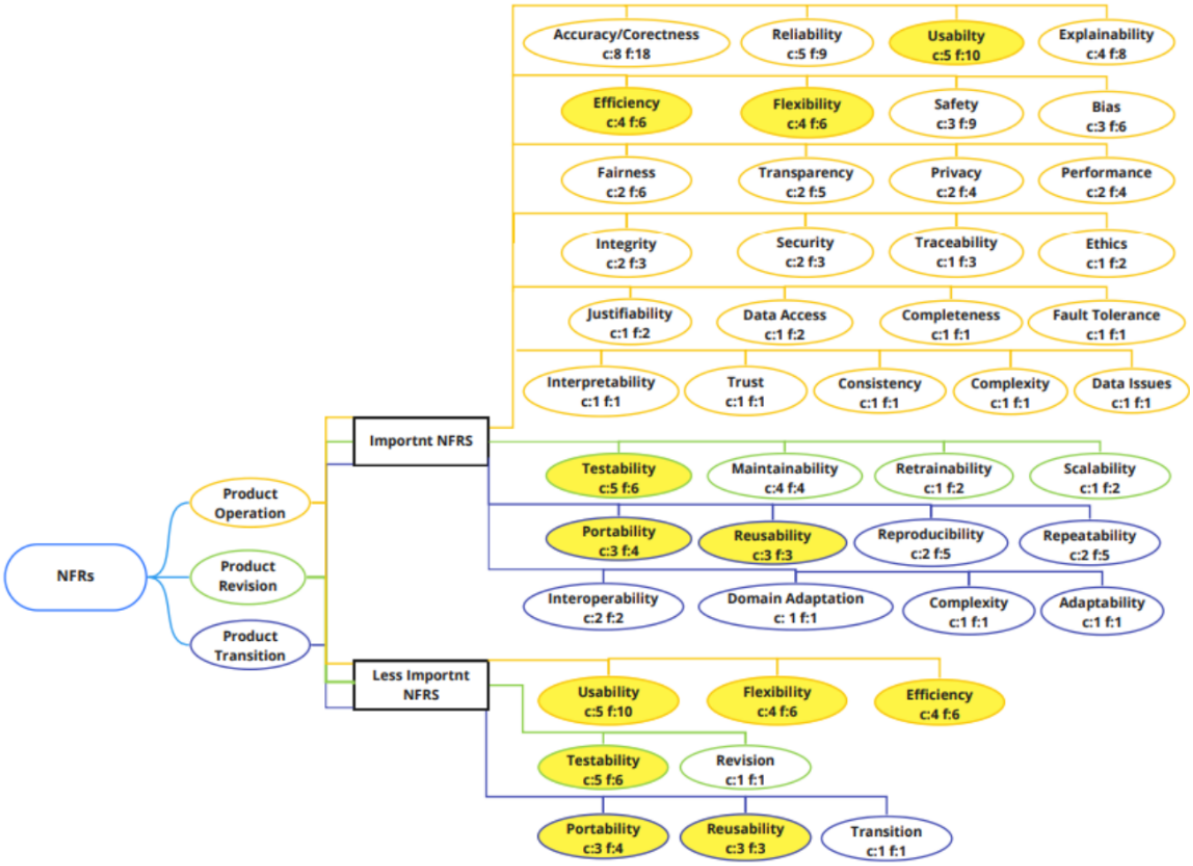


Figure 10.: Important and less Important NFRs for ML. (source: [23])

The study consisted of interviewing 10 different machine learning specialists. In Figure 10, c is the count of the number of interviewees whose interviews included the NFR. f symbolizes the count of occurrences of the code across all transcripts. the Yellow background refers to the NFRs being mentioned by some interviewees as important but identified as less important NFRs by other interviewees [23]. Typically, non-functional requirements act as a guideline for the overall quality of the product. Many quality models have been developed throughout the years to reflect the quality attributes of a software product and group them into non-functional requirements. McCall, FURPS, Dromy, ISO 9126, and ISO 2510 are among the most popular quality models but there are many others. All of these quality models group non-functional requirements into a category of NFR. For instance, in the ISO 9126 model, usability consists of understandability, learnability, operability, attractiveness, and usability compliance. In the FURPS model, the performance requirement consists of velocity, efficiency, availability, answer time, recovery time, and resource utilization. While the grouping of the different NFRs can differ between different models, the premise of all quality models is the same. It is to offer a taxonomy for the different NFRs [24].

Furthermore, non-functional requirements can be measured and benchmarked using multiple metrics. For instance, performance is measurable through the training time of a given machine learning model, or by measuring the time the server needs to communicate with different clients. Efficiency on the other hand can be measured by the amount of data being sent between the clients and the server, as well as, the number of communication rounds that the clients and server need to communicate [22]. Accordingly, the categories in a quality model can be divided into different NFRs. These NFRs can be measured using multiple metrics that reflect how well the software performs in that particular quality dimension.

2.3.3. Software Benchmark

According to Kistowski J.V. et al. (2015) [25] a software benchmark is a "Standard tool for the competitive evaluation and comparison of competing systems or components according to specific characteristics, such as performance, dependability, or security" [25]. Accordingly, In the context of software engineering, a benchmark is a software tool that uses certain quantifiable metrics to compare different competing pieces of software across different dimensions (e.g non-functional requirements). Kistowski J.V. et al. (2015) [25] provided 5 criteria that need to be met for a benchmark to be a high-quality benchmark and they are the following:

- Relevance: The importance of the metrics used by the benchmark to the end users of the benchmarked entities.
- Reproducibility: Results need to be reproducible within a similar configuration.
- Fairness: Not creating any artificial limitations for any of the competing entities.
- Verifiability: The results delivered by the tool are accurate.
- Usability: easy to use by any user that within the given benchmark environment.

3. Related Work

In the last years, there has been a growing interest in the comparison of federated machine learning systems. An increasing number of research papers comparing different federated learning libraries and different federated learning algorithms have been published. There has been also a growing number of benchmarks for federated learning systems that compare the different quality characteristics (e.g accuracy, communication cost, and performance ...) of federated learning systems. The following section aims to provide an overview over the state of research of Federated learning benchmarks

3.1. Federated learning Libraries Benchmarks

Some studies investigated the different FL libraries and created benchmarks for them. These benchmarks either use a specific library to compare the different federated learning systems developed with that library (e.g PyFed), or compare the libraries themselves (e.g UniFed).

3.1.1. PyFed

PyFed is a benchmark built on top of the Pysyft library. It is designed to evaluate the federated learning systems built using Pysyft when handling non-IID data. The benchmark offers five different datasets with three different data distributions (IID data, non-IID data with random distribution, and non-IID data with distribution by label). In addition to that, the benchmark offers an implementation of three different machine learning models using Pysyft. Namely, a convolutional neural network, a long short-term memory neural network, and a Gated recurrent unit neural network. It also offers four evaluation metrics (accuracy, micro average, macro average, and loss), as well as a notebook for the visualization of the results. Figure 11 illustrates the architecture of Pyfed [28].

Bouraqqadi, H., et al. (2021) [28] demonstrated the framework using three different scenarios, two Image classifications for the CNN, and text classification for the LSTM. The system performed better on IID data than on non-IID data. However, on non-IID data, it performed better on the randomly split dataset than on the split-by-label dataset. These results were consistent almost across all metrics.

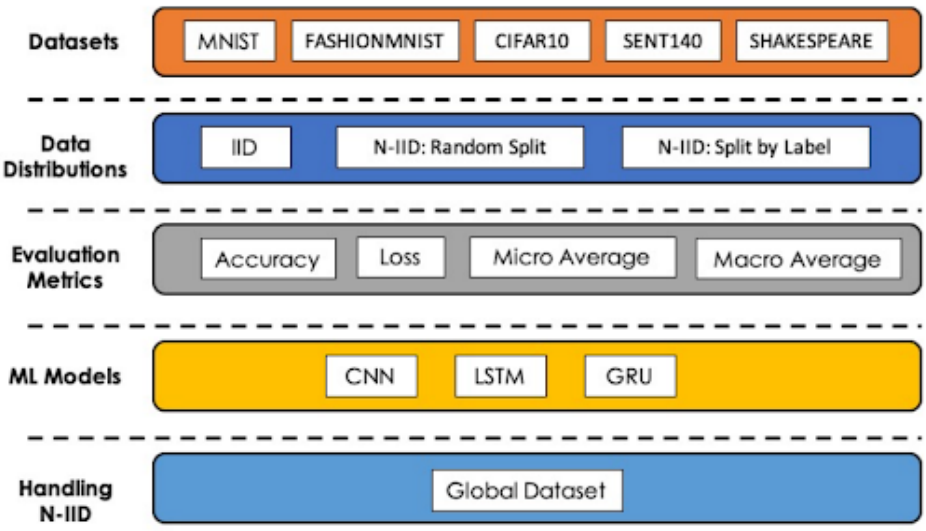


Figure 11.: Pyfed overview (source: [28])

3.1.2. UniFed

UniFed is a federated learning benchmarking tool that benchmarks the different federated learning libraries. It consists of 15 datasets, an environment launcher, a scenario loader, a test environment, and a log analyzer. The environment launcher acts as an interface between the user and the benchmarking environment. The scenario loader loads the dataset, as well as the different configurations. The testing environment has the implementations of the different ML models as well as FL strategies using the different FL libraries and frameworks. The log analyzer just collects the important logs from the different libraries [29]. Figure 12 illustrates the architecture of the benchmark.

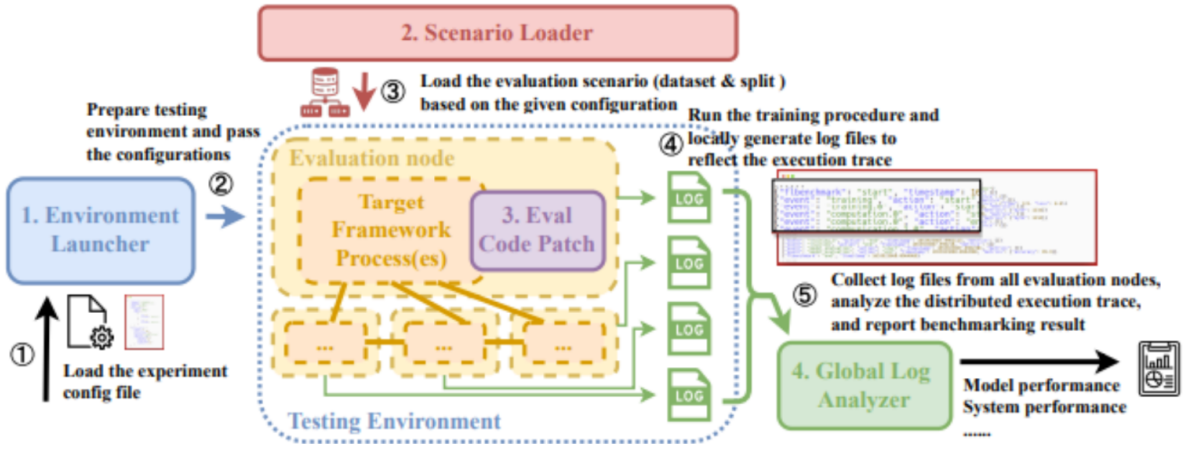


Figure 12.: Design of the (UniFed) benchmark workflow (source: [29]))

Nothing is mentioned about the ML models and FL strategies supported by the benchmark.

However, the experiment included logistic regression, neural networks, and binary tree ML models, as well as, the FedAvg, SecureBoost, and HistSecAgg FL strategies. It also used AUC, MSE, and accuracy as benchmarking metrics [29]. Liu, X., et al. (2022) [29] used the benchmark to compare nine different libraries. The compared libraries are FATE, FedML, PaddleFL, Fedlearner, TFF, Flower, FLUTE, CrypTen, and FedTree. The study first compared the functionalities of the different libraries in terms of privacy mechanisms, supported machine learning models, deployment support (e.g cross-silo, cross-device, simulation)...

The study also compared the overall support that exists for the different libraries in terms of documentation, tutorials, and code samples. In addition to that, the study compared the libraries and frameworks quantitatively. It studied the influence of the libraries as well as the FL strategy on the models' performance and tried to identify the best performant libraries. It used accuracy, MSE, and AUC as performance metrics. The choice of the library had a minimal influence on the models' performance since all the libraries implement the same mathematical procedures for the different machine learning models. However, there were some performance fluctuations. For instance, logistic regression using FATE had relatively low performance. The choice of FL strategy had little to no influence on the performance of the different ML models. Overall, in vertical settings, the tree-based model performed very well, and deep neural networks perform better than shallow ones. The study concluded that for different scenarios, different frameworks are suitable. For instance, Flower and FLUTE are the best when training time is an important factor, and Fedlearner has the lowest memory usage [29].

3.1.3. Federated Learning Libraries Comparison

Kholod, I. et al. (2020) [30] compare five different federated learning libraries in the context of IoT, namely Tensorflow Federated from Google, FATE from Webank, PFL from Baidu, Pysyft from Openminded, and FL and DP from Sherpa.AI. They analyze the ease of setup, capabilities, and maturity of each library, and compare their performance and accuracy. They conclude that the most mature and ready-to-use federated learning library is PFL since it had the best accuracy and performance in the comparison experiment. However, they mention the small community support and the lack of documentation as a shortcoming. Tensorflow Federated and FATE only offer deep learning models and do not offer any implementation of traditional machine learning models. Unlike other Federated learning libraries, FL and DP is easy to install and supports both traditional machine learning and deep learning models. The paper concludes that all the libraries except Pysyft require a lot of resources and are not suitable for IoT use. The comparison done by Kholod, I. et al. (2020) [30] does not offer a step-by-step explanation of their comparison and thus it is hard to reproduce their results. The validity of the results is also specific to the IoT use cases.

3.2. Federated Learning Metric-specific Benchmarks

Some benchmarks and evaluations only compare one quality dimension of federated learning systems. Wei, W., et al. (2020) [31] compare the privacy and security of different FL settings. While Nilsson, A., et al. (2018) [33] and Zhuang, W., et al (2020) [32]. both compare the performance of different federated learning aggregation algorithms

3.2.1. Privacy and security

Wei, W., et al. (2020) [31] offer a framework for evaluating and comparing different attacks on federated learning systems. They also perform a gradient leakage attack against multiple federated learning systems configurations and analyze the effectiveness of these attacks. Finally, they offer two mitigation strategies to limit the impact of gradient leakage attacks.

3.2.2. Performace

Both Nilsson, A., et al. (2018) [33] and Zhuang, W. et al. (2020) [32] compare the performance of federated learning aggregation algorithms. The first compares FedAvg to CO-OP and FSVRG, and the latter compares FedAvg to FedPav. They both use the model training time as a metric for performance.

Performance Benchmark of FedAvg, CO-OP, and FSVRG Nilsson, A., et al. (2018) [33] compare the performance of three widely used federated learning aggregation algorithms. Namely, FedAvg, FedSVRG, and Co-op. They conduct a performance benchmarking experiment of the three aggregation algorithms compared to a centralized machine learning system using the perceptron deep learning model on the MNIST dataset (both i.i.d and non-i.i.d data) for 10000 clients. Table 2 summarizes the findings of the experiment.

i.i.d data				
	FedAvg	**CO-OP**	**FSVRG**	**Centralized ML**
FedAvg	-	better	better	equivalent
CO-OP	worse	-	equivalent	worse
FSVRG	worse	equivalent	-	worse
non-i.i.d data				
	FedAvg	**CO-OP**	**FSVRG**	**Centralized ML**
FedAvg	-	better	better	equivalent
CO-OP	worse	-	equivalent	worse
FSVRG	worse	equivalent	-	worse

Table 2.: Summary of algorithm comparisons (source: [33])

Performance Benchmark of FedAvg, and FedPAV Zhuang, W., et al. (2020) [32] tackles the challenge of statistical heterogeneity by benchmarking the FedPav and FedAvg aggregation algorithms on six distinct datasets using two distinct scenarios for person reidentification (the federated-by-camera scenario and the federated-by-dataset scenario). It uses ID-discriminative embedding (IDE) as a reference model. The results of the experiment concluded that FedPav has a better performance than FedAvg in the person reidentification scenario. However, its performance can be further improved by adjusting the weights of the model, as well as using knowledge distillation, which is a technique that consists of transferring knowledge from a trained large deep learning model to a smaller untrained deep learning model.

3.3. Federated Learning Use-case-specific Benchmarks

Some benchmarks are specific to certain use cases (e.g deep learning, NLP, edge computing). They only measure the performance metrics of the different algorithms and models for a certain scenario. For instance, Basu, P., et al. (2021) [34] benchmarked the privacy of ML and FL models for the NLP use case, and Hao et al. (2018) [35] offers a benchmark for the edge computing scenario. He, C. et al. (2021) [36], Gao, Y., et al. (2020) [37], and Zhang, Z., et al. (2021) [38] all dealt with benchmarking models that dealt with scenarios involving Deep learning models.

3.3.1. NLP and Edge Computing

Basu, P., et al. (2021) [34] benchmarked the privacy of traditional machine learning systems and federated learning systems in the medical field. It used BERT-based models, for different levels of privacy and depression and sexual harassment-related Tweets as a dataset. The study concluded that utility degrades more in non-IID setups than in IID setups.

Edge AIBench [35] claims that it offers a benchmark for federated learning applications in four different edge computing scenarios (ICU, smart home, Autonomous Vehicles, and surveillance camera) across the three layers of edge computing systems (Client, Edge, Server). However, Hao et al. (2018) [35] offer no demonstration and no reference implementation of the benchmark.

3.3.2. Deep Learning

Some publications tackled the use of deep learning in a federated learning setting. Deep learning models are usually resource-intensive. Thus, performance and efficiency can both be bottlenecks for federated learning systems that use deep learning models.

FedGraphNN is a federated learning benchmark for Graph neural networks. Graph neural networks are neural networks that learn from graph-structured data like social network

connections, recommendation systems, or traffic flow models. The learning in Graph neural networks can be done on three distinct levels. Namely, at the graph level, sub-graph level, or node level. The benchmark contains 36 different private and public datasets from seven different domains. The domains are molecules, proteins, and social networks for the graph-level learning datasets. For the sub-graph-level learning datasets, the domains are recommender systems and knowledge graphs. the domains for the node-level learning datasets are social networks and publication networks. In addition to that, the benchmark includes an implementation of both the popular graph neural network models as well as federated learning algorithms. It also incorporates six different performance metrics relevant to graph neural networks. Namely, the training time, the receiver operating characteristic, the mean absolute error, the mean squared error, the root mean square error, and the micro-averaged F1 score. The benchmark is built on top of the FedMl framework. Figure 13 illustrates the architecture of the benchmark [36]. He, C. et al. (2021) [36] performed a comparison of the federated learning system using the FedAvg algorithm to a centralized machine learning system. the federated learning system performed worse in all performance metrics [36].

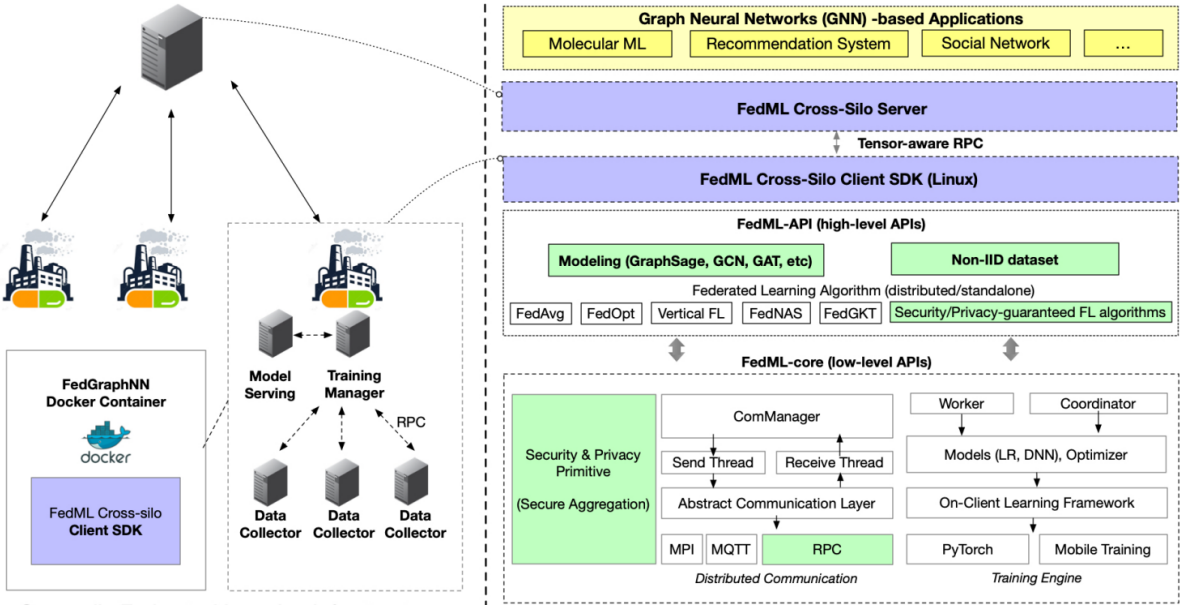


Figure 13.: Overview of FedGraphNN System Architecture Design (source: [36])

Comparison of Federated learning systems to SplitNN: Gao, Y., et al. (2020) [37] compare the performance of federated learning systems to the performance of another distributed machine learning paradigm called SplitNN in the context of IoT. SplitNN is a similar machine learning paradigm to federated learning, with the key difference being that the server distributes the data between the different clients, instead of the clients using their data to train their machine learning models. They use Raspberry Pi devices as a test environment and use model accuracy and model convergence speed in a non-i.i.d dataset as performance metrics. The results of experiments conclude that overall SplittNN performs better than

federated learning on non-i.i.d data. However, when the distribution of the data is extremely imbalanced, federated learning performs better. Furthermore, they conclude that federated learning is better suited for IoT use cases since it uses the hardware resources more efficiently.

Gradient diversity Comparison: Zhang, Z., et al. (2021) [38] studied the effect of gradient diversity on semi-supervised federated learning systems performance. Semi-supervised federated learning is defined in the study as when the clients have unlabeled data, and the server has labeled data. The accuracy is used as a metric. They investigate the different gradient descent techniques used in semi-supervised settings like consistency regularization loss (CRL) and Batch Normalization (BN) and present their technique named Group Normalization (GN). They also propose the aggregation algorithm that they called a grouping-based average. Their experiment demonstrates that the new averaging algorithm performs better in terms of accuracy compared to the FedAvg algorithm and that the combination of group normalization and consistency regularization loss performs better than the combination of batch normalization and consistency regularization loss.

3.4. Federated Learning Setting Specific Benchmarks

Multiple benchmarks have been developed to evaluate federated learning systems in a specific setting (e.g personalized FL, Hetero-task FL, and large-scale FL). While theoretically, general purpose benchmarks can be also used to evaluate such a system, they are not designed to do so. For instance, FedScale can scale to incorporate hundreds of thousands of clients, something that general-purpose benchmarks are generally unable to do.

3.4.1. Personalized Federated Machine Learning Setting

Personalized federated machine learning is when each client in a federated learning setting has its unique model deployed. That means the central server in this setting does not choose and distribute a model. Instead, the different clients choose their model to use. Motley [39] and pFL-Bench [40] are both benchmarks for personalized federated machine learning systems.

Motley tackles the issue of personalized federated machine learning. It keeps track of metrics like accuracy, fairness, and robustness. It also offers seven datasets for both cross-silo and cross-device setups. Motley is designed to study the effectiveness and the extent of personalization techniques on the performance of federated learning systems. Wu, S., et al. (2022) [39] uses Motley in three distinct experiments using different settings (two cross-device, and cross-silo settings) and comes to the following results:

- Different personalization methods perform differently across different performance metrics.
- Tuning the model's parameter in a cross-silo setting is highly effective at improving the performance of the system.

- Personalized federated machine learning comes with tradeoffs compared to traditional federated machine learning.

pFL-Bench offers eleven datasets for benchmarking personalized federated machine learning systems. pFL-Bench also offers multiple performance metrics (e.g memory cost, computational cost, and communication cost of different federated learning settings), as well as different fairness metrics (e.g the top and bottom deciles of performance across different clients and the value of a standard deviation). In addition to that, it offers three deep learning models. Namely, a two-layers CNN, a graph isomorphism neural network, and a matrix factorization model. pFL-Bench also allows the selection of any proportion of clients for benchmarking. That way, not only the effect of the data distribution on the performance of the system can be evaluated but also the effect of the client sampling [40].

3.4.2. Hetero-task Federated Machine Learning Setting

B-FHTL is a federated learning benchmark for hetero-task learning. Hetero-task learning is a federated learning technique where each client in the system does a different task (e.g object detection, image classification, prediction ...). The benchmark consists of three different clusters of non-IID datasets with increasing heterogeneity, the implementation of the tasks that are carried out on each of the clusters, and a set of evaluation metrics. Figure 14 below illustrates the architecture of the benchmark [41].

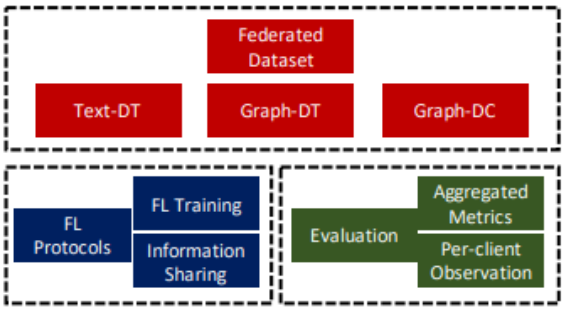


Figure 14.: Overview of B-FHTL (source: [41])

The first cluster of datasets comes with only the task of binary classification and it has a total of 13 datasets. The second cluster comes with two tasks. Namely, binary classification and regression, and has a total of 16 datasets. The third cluster has three datasets and comes with three tasks. Namely, Sentence pair similarity prediction, Sentiment classification, and Reading compression. In addition to that, the benchmark comes with a set of metrics. per-client improvement ratio is a metric used for all of the datasets. For the first cluster of datasets, accuracy is used as a metric. For the second and third clusters of datasets, the overall performance is used as a metric. Overall performance is defined as the following:

$$Overall = Ii \cdot \frac{1}{n} \cdot \sum_{i=0}^{n} \left(\frac{mi - bi}{bi} \cdot 100\% \right)$$

With mi the performance of the client i and bi the baseline on the client. n is the number of clients. Ii is the comparison indicator of client i. Yao, L., et al. (2022) [41] conducted an experiment to demonstrate the capabilities of the benchmark. They compared seven different federated learning strategies and used accuracy as a metric for the first cluster of datasets and the per-client improvement ratio as a metric for the benchmarking of the two other clusters. The used federated learning strategies are FedAvg, FedAvg+FT, FedProx, FedBN, FedBN+FT, Ditto, and FedMAML. For the first cluster of datasets, FedBN+FT had the best accuracy. For the second and third clusters, FedBN+FT and FedMAML had the best per-client improvement ratio respectively.

3.4.3. Large-scale Federated Machine Learning

There are evaluation frameworks and benchmarks specific to large-scale federated learning systems. In cross-device federated learning settings, the systems can scale to incorporate tens of millions of devices. Communication cost, accuracy, and performance can all be influenced by a large number of devices in a federated learning system. Lui, L., et al. (2020) [42] and Lai, F. et al. (2021) [43] both offer benchmarks for such scenarios. These benchmarks are designed to evaluate the performance of the federated learning systems across different metrics that are relevant when the scale of the system is large.

Evaluation framework for large-scale federated learning systems: Lui L. et al. (2020) [42] offers a benchmark for evaluating large-scale federated learning systems. It uses covariate shift, prior probability shift, and concept shift to transform datasets into non-i.i.d datasets. After that, it uses benchmarks like the number of communication rounds, data nodes number, the weight of data nodes, as well as the quality of data nodes to benchmark the federated learning system. It also offers a profile module to customize the different parameters of the datasets. Figure 15 illustrates the architecture of the evaluation framework.

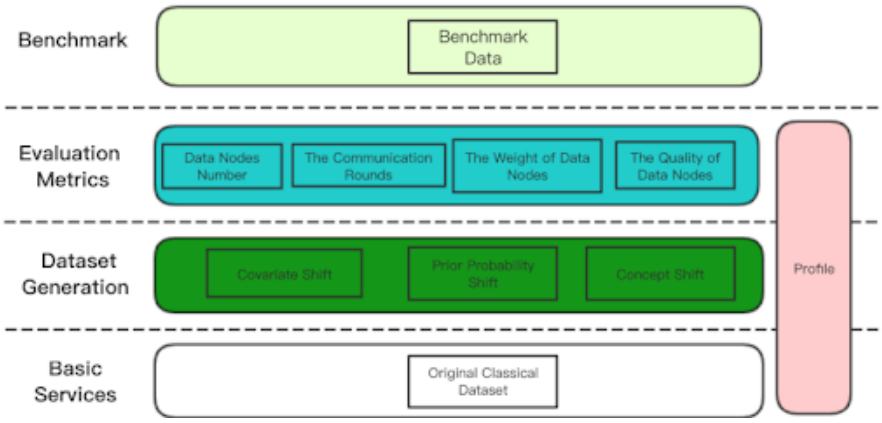


Figure 15.: Evaluation Framework Overview (source: [42])

FedScale offers a benchmark for big-scale cross-device federated learning systems consti-

tuted of twenty different datasets with different tasks ranging from object classification to speech recognition. They also offer emulation of real mobile client behavior like the speed heterogeneity of the different clients, as well as the different availability of the different clients. The system can scale up to 136k clients. The benchmark keeps track of metrics like the number of communication rounds and accuracy to benchmark the statistical and systems efficiency. It also offers configurable security and privacy settings to compare the performance of the system with different settings [43].

3.5. Federated Learning General Purpose Benchmarks

Some benchmarks are neither specific to a machine learning model, to a specific dataset, nor specific metrics. These benchmarks typically offer high customizability. In this thesis, these benchmarks were divided into datasets and complete benchmarks. The dataset benchmarks offered a collection of datasets that the users can use to benchmark their models and algorithms. While, the complete benchmarks offered parameter tuning, different ML models implementations, different federated learning data aggregation algorithms, and different metrics, in addition to the datasets.

3.5.1. Datasets

FLBench [44] and LEAF [45] are both general-purpose benchmarks. However, they both do not come with any reference implementations of any machine learning model. They both however offer datasets from various domains, suited for different federated learning scenarios.

FLBench is a benchmarking suite that offers real-life datasets from various domains (medical, finance, IoT) for real-life configurable scenarios keeping track of different metrics for communication, privacy, data heterogeneity, and cooperation strategy. Liang et al. (2020) however do not provide any insight into the benchmarked metrics, or any demonstration of the benchmark [44].

LEAF is a modular open-source federated learning benchmark. It offers six open-source machine learning training datasets, as well as efficiency, accuracy, and performance metrics. It also offers reference implementations of minibatch SGD, FedAvg, and Mocha federated learning algorithms to make the results of Caldas, S., et al. (2018) easily producible. Caldas, S., et al. (2018) [45] do not present any benchmarking experiment, they just present the functionality of their tool.

3.5.2. Complete Benchmarks

FedML [46], OARF [47], and FedEval [48] are all general-purpose benchmarks. Like FLBench [44] and LEAF [45], they are agnostic to the federated learning scenarios. However, in addition to, the datasets, they come with different metrics, reference implementations of a set of supported federated learning strategies as well a machine learning model. FedML can even

be used as a federated learning framework.

FedML is a multi-functional federated learning framework and benchmark. It offers multiple functionalities like the support of different computing paradigms (e.g standalone simulation, distributed computing, on-device training), flexible API design for message flows, message customization, and topology customization. It also supports Split learning (splitNN) as well as different federated learning paradigms (e.g FedAvg, decentralized FL, FedNAV, vertical FL). For benchmarking, it offers eleven different datasets for different federated machine learning use cases and scenarios (e.g linear, shallow neural networks, deep neural networks). He, C., et al. (2020) provide a demonstration of the benchmark. However, it is only for demonstration purposes and offers no actual results [46].

Open Application Repository for Federated Learning (OARF) is a general purpose highly customizable federated learning benchmark. It supports the customization of federated learning systems across multiple dimensions like the reference machine learning model, dataset, or aggregation algorithm. Figure 16 illustrates the architecture of OARF and the supported federated learning paradigms. The user can theoretically create any supported combination to benchmark any supported metric [47].

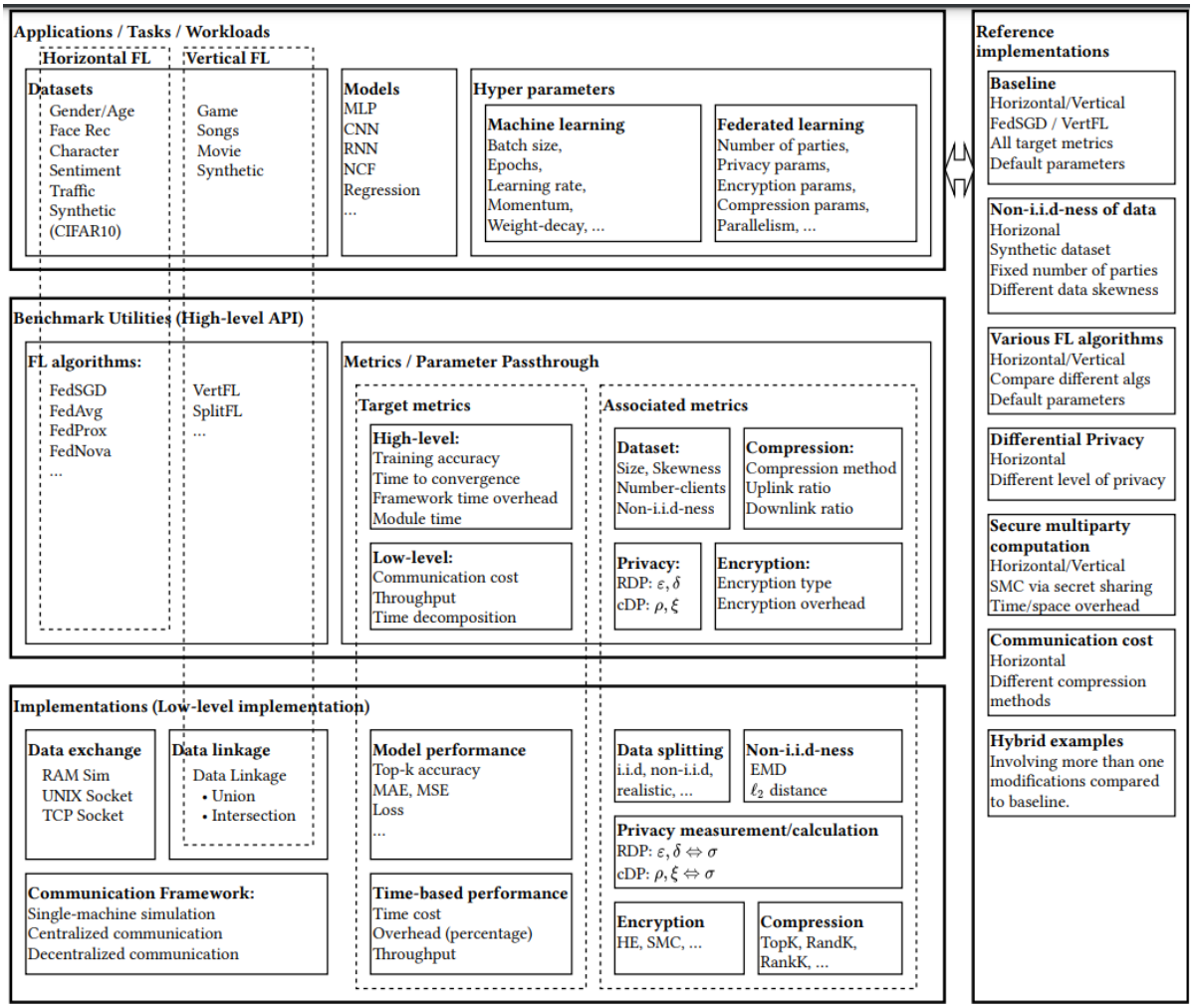


Figure 16.: The architecture of the OARF framework (source:[47])

Furthermore, Hu, S., et al. (2022) [47] offer a demonstration of the tool by benchmarking the accuracy of the FedSGD algorithm to a reference implementation using a sentiment analysis dataset as well as an object classification dataset. In both setups, the reference implementation outperformed the FedSGD algorithm.

FedEval is a benchmark for federated learning systems. Unlike other benchmarks, FedEval does not offer any dataset, machine learning model implementations, or federated strategy implementation. Instead, it takes them as input from the user and emulates a client-server setting. Then, it measures five different metrics. Namely, accuracy, communication cost, time consumption, privacy, and robustness. In the end, it displays the final results of the benchmarking on a dashboard. Figure 17 illustrates the architecture of FedEval [48]. Chai, D., et al. (2020) [48] demonstrated the benchmark by comparing the FedAvg and FedSGD federated strategies. It used five different datasets as well as two different machine-

learning models. The results were that FedAvg outperformed FedSGD concerning accuracy, communication cost, time consumption, and privacy. While FedSGD outperformed FedAvg concerning robustness.

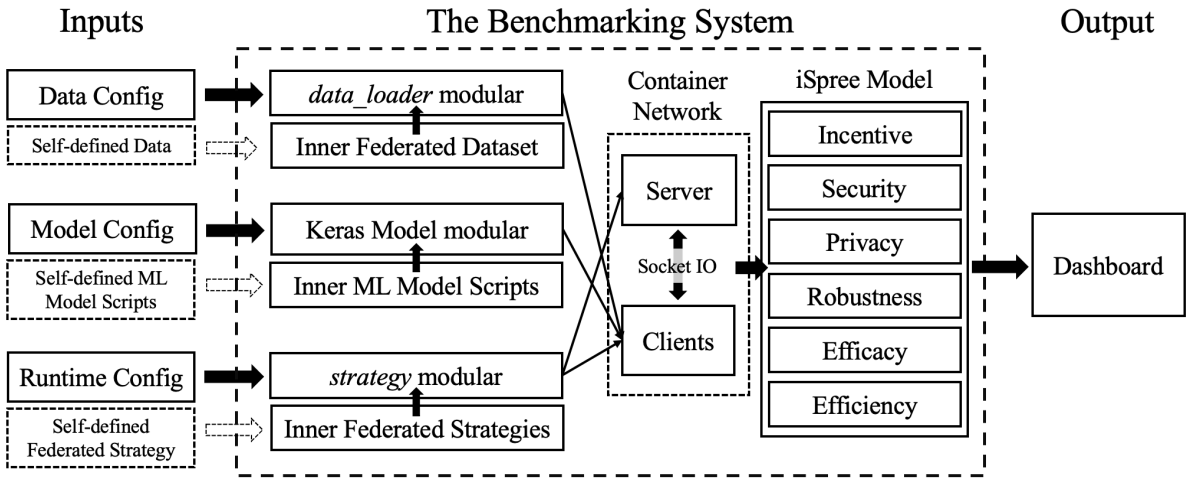


Figure 17.: The architecture of FedEval (source: [48]))

3.6. Summary

There are multiple publications that offer different benchmarks for federated learning systems. Each presented its datasets, metrics, machine learning models, and FL strategy, and tackle a different use case. There are a few that are general purpose and customizable but these can not be used for benchmarking the libraries since they already come with their implementation of the different scenarios. Table 3 illustrates the findings of the literature review of the different FL benchmarks.

	Number of datasets	**Metrics**	**ML models**	**FL strategies**	**Use-case**
FedEval	0	accuracy, communication, time, privacy, robustness	none	none	General
FedMl	11	accuracy	none	FedAvg	General
OARF	10	multiple metrics	multiple models	FedSGD, FedAvg, FedNOVA, and FedProx	General

pFL-Bench	11	memory cost, computational cost, communication cost, top and bottom deciles of performance , and the value of a standard deviation	CNN, a graph isomorphism NN, and a matrix factorization model	multiple	personalized FL
PyFed	5	accuracy, loss, macro, and micro averages	CNN, LSTM, GRU	FedAvg	Libraries Benchmark
Edge AIBench	none	none	none	none	Edge computing
FedGraph NN	5	ROC, MAE, MSE, RMSE, micro F1, and Training time	Graph NN	FedAvg	Graph NN
B-FHTL	32	per-client improvement ratio, accuracy, and overall performance	Binary classification, regression, and NLP	FedAvg, FedAvg+FT, FedProx, FedBN, FedBN+FT, Ditto, and FedMAML	Hetero-tasks
FedScale	13	number of communication rounds and accuracy	None	None	Large scale FL
UniFed	15	Accuracy, AUC, and MSE	Binary tree, NN, Regression	FedAvg	Libraries Benchmark

LEAF	6	efficiency, accuracy, and performance	None	minibatch SGD, FedAvg and Mocha	General
FLBench	multiple	communication, privacy, and data heterogeneity	None	None	General
Lui L. et al.	2	number of communication rounds, data nodes number, the weight of data nodes, and the quality of data nodes	DNN	FedAvg	Large scale FL
Motley	4	accuracy, fairness, and robustness	DNN	FedAvg	personalized FL

Table 3.: The different FL benchmarks (source: own work)

There are two benchmarks for libraries. Namely, PyFed [28] and UniFed [29]. However, Pyfed is specific to Pysyft, and can not be used to compare the different libraries. UniFed is highly customizable and can benchmark the different libraries. However, it doesn't offer a lot of non-statistical metrics and it doesn't support many ML models. In addition to that, the users of UniFed are expected to be able to program their scenarios and use the CLI of UniFed. These two limitations hurt the usability and the relevance of the benchmark.

4. Research Methodology

This thesis leveraged a manifold research approach compromised of four methodologies. Namely, a semi-structured expert interview to answer the first research question, a literature review as well as a document analysis to answer the second research question, and the design science research methodology and an experiment to answer the third research question.

4.1. Expert Interview

To collect the FR and NFR for federated learning systems a semi-structured expert interview was selected as a research method. Unlike a structured interview where the interviewees should not divert from the interview questions, in a semi-structured interview, the interviewees are allowed to divert from the questions, to share their personal experiences with the interviewers [76]. The semi-structured interview was chosen because of its qualitative nature and the scarcity of literature on federated learning systems requirements. Before starting the interviews, more than 20 people were contacted through LinkedIn and other mediums. Five of them accepted the interview requests but only four showed up for the interviews. The sample size of the interviewees was four. All of them are industry experts with experience building or researching federated learning systems. The interviewer took notes during the interviews and recorded the answers of the interviewees to the different questions as well as the additional information they provided. A set of 20 interview questions was prepared to ask the interviewees about themselves, their experience with federated learning, the functional and non-functional requirements they deem important for federated learning libraries, and the metrics to measure the performance of their federated learning systems across these dimensions. The data collected during the interviews were qualitative. The method used for the data analysis was the thematic analysis method. The thematic analysis method is a data analysis method that emphasizes searching for and analyzing patterns in qualitative data [77].

4.2. Literature Review and Document Analysis

To extract the data about the different federated learning libraries and their capabilities and limitations a qualitative methodology is needed. Thus, a mixture of literature review and document analysis was chosen as a research method to answer the second research question. These methodologies were chosen because of their qualitative nature.

4.2.1. Literature Review

A literature review [74] is a research methodology that is used to give an overview of a specific research topic and the current state of research around that topic. It can be either systematic or non-systematic. Systematic literature reviews focus more on the collection of data from the scientific literature. While a non-systematic literature review focuses more on the current state of the scientific literature. For this thesis, a non-systematic literature review was chosen as the research methodology. The goals were mainly to get an overview of the scientific literature around federated learning, identify gaps in the research around the subject with regards to federated learning libraries comparisons, and identify the different FL libraries developed or used by the researchers. A systematic literature review will not be as suited as a non-systematic one because there are a large amount of key words that can be useful in the context of this thesis but hard to identify. For instance, key words that are specific to NFRs or to specific libraries. The following process was followed for the literature review:

Order	Step	Description
1	Defining the topic of research	1. Get an overview of the federated learning research. 2. Identify gaps in the research concerning library comparisons. 3. Identify the FL libraries in the literature. 4. Identify the FL libraries features used in the literature.
2	Search and select articles	1. A search was conducted on the google scholar search engine with FL-related keywords. 2. The articles with relevant titles were identified. 3. The abstracts of the candidate papers were read. 4. The papers with the relevant abstracts were read.
3	Analyse the literature	Extract the information relevant to the research topic.
4	Summarize the findings	Write a summary of the findings.

Table 4.: Literature review process (source: own work based on [74])

4.2.2. Document Analysis

Document analysis is a qualitative research method in which documents are analyzed and relevant information is extracted from them [75]. In this thesis, after identifying the different federated learning libraries, a document analysis of their official documentation, GitHub

repositories, and official websites was carried out. The goal was to identify the functionalities and features of each of the libraries as well as their capabilities and limitations. Some additional meta-data (e.g library version, library developer ...) was also extracted.

4.3. Design and Development of the Tool

To compare the difference between the FL libraries across different quality dimensions, an artifact needs to be built. The design science research methodology was used as the method to design and develop the artifact. The design science research methodology (DSRM) is an approach to designing and building artifacts for research purposes. A researcher using the DSRM needs to follow these five steps [78]:

1. Problem identification and motivation: This step deals with the motivation behind the artifact as well as the identification of the problem that the artifact needs to solve. The following question needs to be answered. What is the problem that the artifact is solving?

2. Definition of solution objectives: This step deals with the definition of the capabilities of the artifact. The following question needs to be answered. How is the artifact going to solve that problem?

3. Design and development: This step deals with the building of the artifact. The following question needs to be answered. How are the solutions going to be implemented?

4. Demonstration: This step deals with the demonstration of the capabilities of the artifact. The following question needs to be answered. What is the efficacy of the solution?

5. Contribution: This step deals with the contribution of the tool to the advancement of the current research. The following question needs to be answered. What is the contribution of the solution to the current research?

Table 5 illustrates how each of these steps is defined in the context of this thesis. First, the problem needs to be identified. Then, an artifact solution for it needs to be defined. Afterward, the solution needs to be designed and developed. The solution needs to be demonstrated after that. Finally, the contribution of that solution to the current research needs to be highlighted.

Step	**Question**	**Description**
Problem identification and motivation	What is the problem that the artifact is solving?	Qualitatively compare different quality dimensions of the FL libraries using different metrics. The quality dimensions are scalability, performance, efficiency, and accuracy.

Definition of solution objectives	How is the artifact going to solve that problem?	A benchmarking suite that allows multiple experiments to be conducted using different ML models implemented with the different federated learning libraries. It will collect the logs for the different metrics from the libraries and display them on an admin dashboard.
Design and development	How are the solutions going to be implemented?	The benchmarking suite is constituted of multiple modules. Namely, a module for each FL library that has the implementation of the different ML models and FL strategies in it, a module for a web application that communicates with the different libraries modules and acts as an admin panel to configure and conduct the different experiments using the tool, and a module for the different datasets.
Demonstration	What is the efficacy of the solution?	The benchmarking suite needs to present fair, verifiable, and reproducible results.
Contribution	What is the contribution of the solution to the current research?	An easy-to-use benchmarking suite that is modular and extensible.

Table 5.: DRSM for the benchmarking suite (source: own work based on [78])

4.4. Experiment

To test the quality of the library across the following quality dimensions: scalability, performance, efficiency, and accuracy, 13 different experiments on the MNIST dataset were conducted. The first one was a traditional ML experiment to train a CNN using PyTorch. It was used for reference. The other 12 experiments were conducted with four different FL libraries (FedML, Flower, Pysyft, and FATE). Each of the libraries was used for three experiments. The first experiment had 2 FL clients, the second experiment had 16 FL clients, and the third experiment had 100 FL clients. The used federated aggregation algorithm is FedAvg. Twelve different metrics relevant to the quality dimensions were monitored and logged during the different experiments. The hardware specs used in the experiments are illustrated in Table 6.

Characteristics	**Specifications**
Model	Dell DE-L081906
OS	Windows 10 Enterprise
CPU	Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz 2.59 GHz
GPU	Nvidia Quadro P1000 (was not used)
RAM	32,0 GB (31.8 GB usable)
Hard Drive	SSD 470GB

Table 6.: Hardware specifications for the experimentation computer (source: own work)

5. Libraries Comparison

Since 2016, and the introduction of federated learning. Multiple libraries and frameworks have implemented the different functionalities of federated learning to make the development of FL systems easier. In this thesis, a series of semi-structured interviews will be conducted to identify the different features and criteria deemed important by the FL community. Then, the libraries as well as their functionalities will be researched, and the important metrics to compare them will be identified through a combination of document analysis as well as a literature review. Then the libraries will be benchmarked following these metrics using the FMLB benchmark suite that was developed as part of the thesis to conduct the benchmarking experiments.

5.1. Important Functional, Non-functional Requirements and their Metrics for Federated Machine Learning Libraries

To gather the functional and non-functional requirements relevant to federated machine learning libraries as well as the metrics to measure them, a series of semi-structured interviews [76] were conducted and thematic analysis [77] as the data extraction method. This section presents the results of the interviews.

5.1.1. Pre-Interviews

Before the interviews, people that had experience with federated learning were contacted. Around 20% of the contacted people accepted the interview request, an invite to a one-hour-long interview was sent to them. All of the interviews took between 50 minutes, and 1 hour and 5 minutes.

Data collection: The first two interviews were transcribed using the Microsoft Teams transcript functionality. The rest was documented in the form of notes taken by the interviewer whenever the interviewee's answer was relevant to the question being asked. Since the interview was semi-structured, the interviewee gave long sentences as answers and discussed every question in detail. The interviewer took notes, shortening the long answers only to the part relevant to the questions being asked. Table 7 illustrates the profiles of the different interviewees. All the interviewees are researchers or come from a research background. They come from different backgrounds (IT security and cloud computing, machine learning, robotics, aerospace ...). They all hold a masters degree, and all of them have one year of experience researching, designing and implementing federated learning systems. The overall

experience of the interviewees differs. Three have 3-4 years of industry experience and one has 18 years.

Participant	Field	Role	Experience	Experience with FL	Responsibility	FL Projects
P1	Industry research	Researcher	3 years	1 year	Research, design, and implement federated learning sector projects	One public sector project
P2	Industry research	Senior researcher	18 years	1 year	Research, design, and implement privacy enhancing technologies	One Edge computing project
P3	FL	AI software developer	3 years	1 year	Develop an FL library	Building an FL library
P4	Industry research	Research assistant (Ph.D.)	4 years	1 year	Research about security and privacy in the aerospace industry	two projects (in robotics and automotive)

Table 7.: Interviewees profile and experience (source: own work)

Interview questions: There was a total of twenty interview questions, divided into four categories. The first category was the background questions. These questions dealt with the experience of the interviewee in general (questions 1 - 4) and with federated machine learning in particular (questions 5 - 7). After that, there were the functional requirement questions (questions 8 - 13). These questions dealt with what the interviewee deemed important for a federated learning library to have in terms of features and functionality. The third part was the non-functional requirements questions (questions 14 - 17). These questions dealt with which non-functional requirements the interviewees deemed important for a federated learning system to have, and thus to be built-in in the library. The last part was the non-functional requirements metrics part (questions 18 - 19). Which dealt with the relevant metrics used to measure the different non-functional requirements in federated learning systems. In the end,

the interviewees were asked an open-ended question (If there is anything they would like to add). Table 8 represents the list of questions asked by the interviewers to guide the interview.

Interview Question	**RQ**
Background of the Interviewee	
1. Please introduce yourself and your role in this company/organization.	N/A
2. Do you consider yourself more of an academic person or an industry-related person?	N/A
3. Total years of experience in the industry and how long have you been in your current position?	N/A
4. Please describe your responsibilities in your organization (e.g., Product owner, developer, Software Architect).	N/A
5. Please describe your experience working with FL.	N/A
6. For what use cases do you use FL?	N/A
7. Which FL libraries do you know?	RQ 1
FR-related questions	
8. What features are the most important for FL libraries?	RQ 1
9. Which aggregation algorithms do you usually use?	RQ 1
10. Which ML models do you usually use?	RQ 1
11. Do you use security mechanisms? if yes, do you prefer encryption-based security or Anonymisation-based security?	RQ 1
12. How often do you work with vertically partitioned data?	RQ 1
13. How often do you work with non-IID data (heterogeneous data)?	RQ 1
NFR-related questions	
14. Do you think NFRs play an important role in the success of FL systems? If yes, how?	RQ 1
15. What non-functional requirements do you think are important for FL systems?	RQ 1
16. Do you measure NFRs over FL-enabled software?	RQ 1
NFR Measurement questions	
17. What are the most important metrics for NFRs in an FL context?	RQ 1
18. How do you capture NFRs and their measurement for FL?	RQ 1
19. What are the challenges you face measuring NFRs for FL?	RQ 1
20. Do you have anything else you would like to add?	N/A

Table 8.: The list of the asked interview questions (source: own work)

5.1.2. Interviews

The answers for the four interviews were summarized and reported to make the analysis of the results easier and more reproducible. The summary contains what was said by the interviewees that were relevant to the interview questions. Everything that is not relevant has been discarded from the summary. Also, each interviewee structured their answers differently, the summary abstracts away all of this and reports the answers in a uniform structure.

First Interview: The first interviewee is a researcher at a bavarian software research institute. During his studies, he focused mostly on AI and worked for three years in total in AI research one of which he worked full-time. His main tasks are the research, design, and implementation of a PoC for the solutions for the industry clients of his institute. He considers himself both an industry and an academic person. He has been working on federated learning for one year. Namely, on a project for the public sector to enhance civic participation using NLP models to classify the citizen's ideas. In his work, he used the IBM federated learning library but he also knows of TensorFlow federated. In addition to that, he has been researching the topic of accountability in federated learning systems and trying to come up with more FL use cases for the public sector. Mainly in agriculture and finance. He has both practical and research experience with federated machine learning.

He worked with FL only using deep learning algorithms (mainly in NLP) and only used fedAvg as a federated data aggregation algorithm. He thinks that a federated learning library should implement both encryption and differential privacy as security mechanisms because there are multiple stakeholders in an FL project: Encryption is more important for the technical stakeholders while differential privacy is more important for the non-technical ones. He has never worked with vertically partitioned data in an FL setting and has often worked with non-I.I.D data. He thinks that in terms of functionality a federated learning library should offer features to easily implement and deploy FL systems in a cross-silo setting as he deems it the most realistic use case for FL. The support of decentralized paradigms like the support of P2P network topologies is an important feature for him too. Since FL in a cross-silo environment requires some level of consensus.

He thinks that the non-functional requirements are important for FL projects especially fairness, scalability, and accountability since these systems are decentralized by nature. He thinks that a library should implement metrics to measure the bias in a system (fairness), it should be able to scale both in terms of the number of clients and volume of data by the clients (scalability), and it should provide a logging system to track the actions of the different participants (accountability). He thinks that accuracy, efficiency, and performance are not important in federated learning settings since these are issues that need to be dealt with on the level of the used ML framework. He also mentions that robustness is an extremely important non-functional requirement in the context of federated learning since these systems need to be working 99.99% of the time.

The only non-functional requirements he tried to measure were scalability (maximum number of supported clients and number of communication rounds), accuracy (F1, Hit rate, Precision, Specificity, False alarm rate), and accountability (by setting different accountability levels). Accountability was the most challenging NFR to measure for him since there isn't a unified system to measure accountability in software engineering. Overall he thinks that federated learning is a technology with a lot of potential and that it will open the door for many applications that we don't know of yet. He also thinks that legal ambiguity is a challenge that needs to be solved to be able to design better privacy-enhancing technology for the future.

Second Interview: The second interviewee is an applied researcher at a german cloud computing company. He comes from an IT security background and has 18 years of industry experience overall. Mainly in research. He has been in his current position for two years and has one year of experience with federated learning. He mainly researches, designs, and implements IT security systems for companies but lately, he has been focusing on privacy-enhancing technologies, and that's how he got introduced to federated machine learning. He worked for a year on the implementation of federated learning in edge computing scenarios for companies that are willing to share their ML models. He knows flower, pysyft, and the IBM federated learning library. He has both practical and research experience with federated machine learning.

He thinks that a federated learning library should abstract away the communication layer of the system and automate the orchestration of the different clients in the system. He also thinks that it is not important for a federated learning library to support the different distributed computing paradigms and sees cross-silo as the most realistic scenario for the implementation of federated learning systems. In addition to that, he thinks that federated machine learning should be easily integrable with as many machine learning frameworks as possible. He worked with both deep learning and traditional machine learning models in a federated setting and he doesn't think that the federated learning library should have the function of implementing the different machine learning models but it is the function of the machine learning frameworks. He only used both of fedAvg and SecBoost for federated data aggregation and differential privacy as a security mechanism in his project. He deems both differential privacy and encryption as important security mechanisms that an FL library should support. He has worked with neither vertically partitioned data nor non-i.i.d data in a federated setting.

The interviewee thinks that non-functional requirements are very important for federated learning projects. Especially accuracy and scalability. He sees resource efficiency as important, and fairness as somewhat important but sees performance (in terms of speed) as not important at all. For a library, he thinks that a library should be easily useable and should offer consistent APIs. The interviewee doesn't measure the non-functional quality of his FL project other than accuracy and efficiency, and he uses metrics like the per-client improvement rate and loss function to measure the first, and communication cost (in terms of network usage) for the latter. He thinks that a normal benchmarking experiment suffices to compare the

efficiency, scalability, and accuracy of the different libraries and that there should be no issue benchmarking these NFRs. As a closing note, he thinks that federated learning has taught us about many new use cases that we didn't know about.

Third Interview: The third interviewee was a researcher in the field of robotic and artificial intelligence and also holds a master's degree in that field. He had two years of experience working as a researcher, and for a year he has been working on developing his federated learning library and launching a company in that field. He considers himself both an academic and an industry person. He is mainly working on the research and development of the FL library. He aims to launch the first release in two months. He did a market research about the federated learning libraries currently available, and he knows Tensorflow federated, Pysyft, and Flower. He deployed some mini-projects using the different libraries to understand how federated learning works but most of his experience comes from the research he has done, and the federated learning library he is currently developing. He thinks that the most important features an FL library should have are the following:

- Easy integration with the existing ML framework
- C++ support
- An encryption mechanism
- State management
- Support of vertically partitioned data
- Federated data aggregation (especially fedAvg)
- Easy benchmarks
- Easy way to handle biases and non-i.i.d data
- Offers a simulation mode
- GPU support

In addition to that, he thinks that the support of different computing and machine learning paradigms (like P2P network topologies, personalized learning, hetero-task federated learning ...) should not be the concern of the federated learning library. According to the research he has done the use of the following features (fedAvg, horizontally partitioned data, cross-silo, encryption) covers the majority of the federated learning use cases that we currently have. He thinks that the machine learning models implementation should not be the task of the federated learning library but the machine learning framework that the library uses.

The interviewee thinks that non-functional requirements are important for federated learning systems especially accuracy, fairness, hardware efficiency, and scalability. He thinks that performance is not that important. He usually measures the accuracy using logs of the

different native metrics of the ML frameworks but other than that he doesn't measure the NF quality of the systems he builds. He thinks even though that accuracy is important for any machine learning model, it is almost certain that the federated learning model will mostly outperform the non-federated learning model given enough data.

For measuring the accuracy, he thinks that the native metrics of the different machine learning frameworks are enough, the native OS functionalities are enough to measure the efficiency, and he doesn't see a suitable metric for measuring the fairness of the system. For scalability, he thinks the best way to measure it is to put some scalability thresholds (e.g 10 clients, 100 clients, 1000 clients) and try to reach them through the simulation of a real federated learning system. He also thinks that measuring performance is not important and should not be challenging since it is just about the implementation of a timer. Overall, he thinks that the measurement of the most important federated learning quality metrics other than fairness and scalability should not be challenging, and the use of basic tools for that should be enough.

As a closing note, he thinks that federated learning has a lot of potential as a technology. Especially in the field of autonomous driving, but he thinks that the main issue in the field is that most of the research and experimentations are done on a small scale in simulation or cross-silo environments. Thus, more research needs to be done on large-scale cross-device settings to unveil the real bottlenecks of federated learning. He also sees federated learning being intensively used on the cloud to share data between corporations.

Fourth Interview: The fourth interviewee was a research assistant and a Ph.D. candidate in the field of automotive and aerospace. His main focus is safety, security, data sovereignty, and privacy in AI. He has four years of research experience in the field of ML and one year of experience in the field of federated learning. He works on a project that will enable german corporations to have more data sovereignty and safely share their data. He describes his tasks as a combination of scientific research, management, organization, and software development. He considers himself both an academic and an industry person. He worked on the research of federated learning. The main topics were the different use cases of FL and the mechanisms that enable the technology. In addition to that, he implemented two FL projects with pysyft. The first one was an automotive ML model that analyses time-series data and predicts when a car component would fail. The second project was the implementation of a federated learning system that trains the data locally across multiple robots that generate data. He only used pysyft but also knows flowers.

He thinks that a federated learning library should be easily compatible with as many machine learning frameworks as possible. The support of vertically partitioned data and mechanisms to handle non-i.i.d data are crucial features for him. He also thinks that a federated learning library should offer GPU support, and should be able to run in a simulation mode, cross-silo, and cross-device environments. He thinks that having native benchmarks is a good feature to have but not important and that the support of the different computing

paradigms and the customization of the networks are not important features to have. He thinks that the support of neural networks is more important than the support of traditional machine learning models for federated learning systems. He thinks that encryption is more important than differential privacy as a security mechanism since he is not convinced of the effectiveness of the latter, and he thinks that fedAvg would be enough to have as a data aggregation algorithm since it covers most of the use cases for federated learning. He also thinks that non-functional requirements are important for the success of federated learning projects and that a good federated learning library should accommodate them. He thinks that efficiency and accuracy are the most important NFR to optimize for in a federated setting. While performance and fairness are important, scalability is somewhat important. However, he thinks that accuracy should not be a concern despite its importance since federated machine learning models tend to have better accuracy than centralized machine learning models but bias should be a concern despite its lower importance because federated learning systems tend to have a lot of device biases in terms of availability and data homogeneity.

The federated learning projects he worked on, measured mainly the efficiency and accuracy of the federated learning models that they developed and deployed. For accuracy, they used the native ML framework metrics (like hit rate, false alarm rate, and precision ...), and for efficiency, they measured the RAM, GPU, and network usage using an internal Linux monitoring package that they developed called LAR. He thinks that these are the most important metrics to measure in a federated learning system, and he thinks that there could be some obstacles trying to measure them but overall they should not be challenging to measure. As a final note, he thinks that federated learning is a powerful technology and that many new use cases will get unveiled as the technology matures. He thinks that the industry is still not aware of the potential of the technology, and that is partly due to the complexity of the technology. He thinks that as the technology will become easier to use, there will be a spike in its adoption.

5.1.3. Post-Interviews

After the interview, the data was extracted, encoded, and analyzed. Finally, the results of the analysis were summarized.

Thematic code: The data was encoded thematically [77] as follows:

1. A profile of the average interviewee persona was created. The profile reflects the role, responsibility, as well as experience of the average interviewee.

2. The use cases of federated learning as well as the libraries known by the interviewees were encoded with an attribute (c = the count of occurrences).

3. The most important features were also enumerated with the attribute (c = the count of occurrences). The features are categorized functionally (e.g differential privacy and encryption are categorized as security mechanisms).

4. Non-functional requirements are also enumerated and have two distinct attributes (c = the count of occurrences, and a = the average importance according to the interviewees. a=1 is the least important, and a=5 is the most important). The non-functional requirements are also further divided into the metrics that measure them. The metrics only have one attribute (c = the count of occurrences). Some of the NFRs didn't have an average because they were not mentioned by all of the interviewees.

5. Another theme was added to encode the relevant additional information that doesn't fall into any of the above-mentioned themes.

Figure 18 illustrates the average profile of the interviewee, as well as the mentioned use cases of federated machine learning, and the libraries known by the interviewees. It was color-coded as follows: grey for the categories, lighter grey for the subcategories, red for the use cases and libraries that are mentioned once during the interviews, orange for the ones mentioned twice, and yellow for the ones mentioned three times.

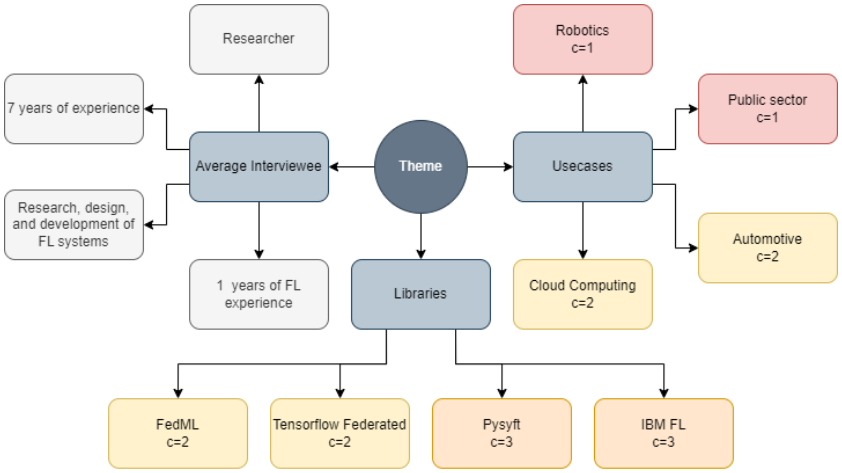


Figure 18.: Thematic encoding of the average interviewee, and the occurrences of the federated learning libraries as well as the use cases of FL in the interviews (source: own work)

Figure 19 illustrates the important features of a federated learning library (categorized by functionality) as well as the mentioned non-functional requirements (sorted by average importance), and the metrics relevant to their measurement. It was color-coded as follows: grey for the categories, lighter grey for the subcategories, red for the FR and NFR that are deemed not important according to their count (for metrics and FR) or relevance factor (for NFR), orange for the ones deemed somewhat important, and yellow for the ones deemed important, and green for the ones deemed so important.

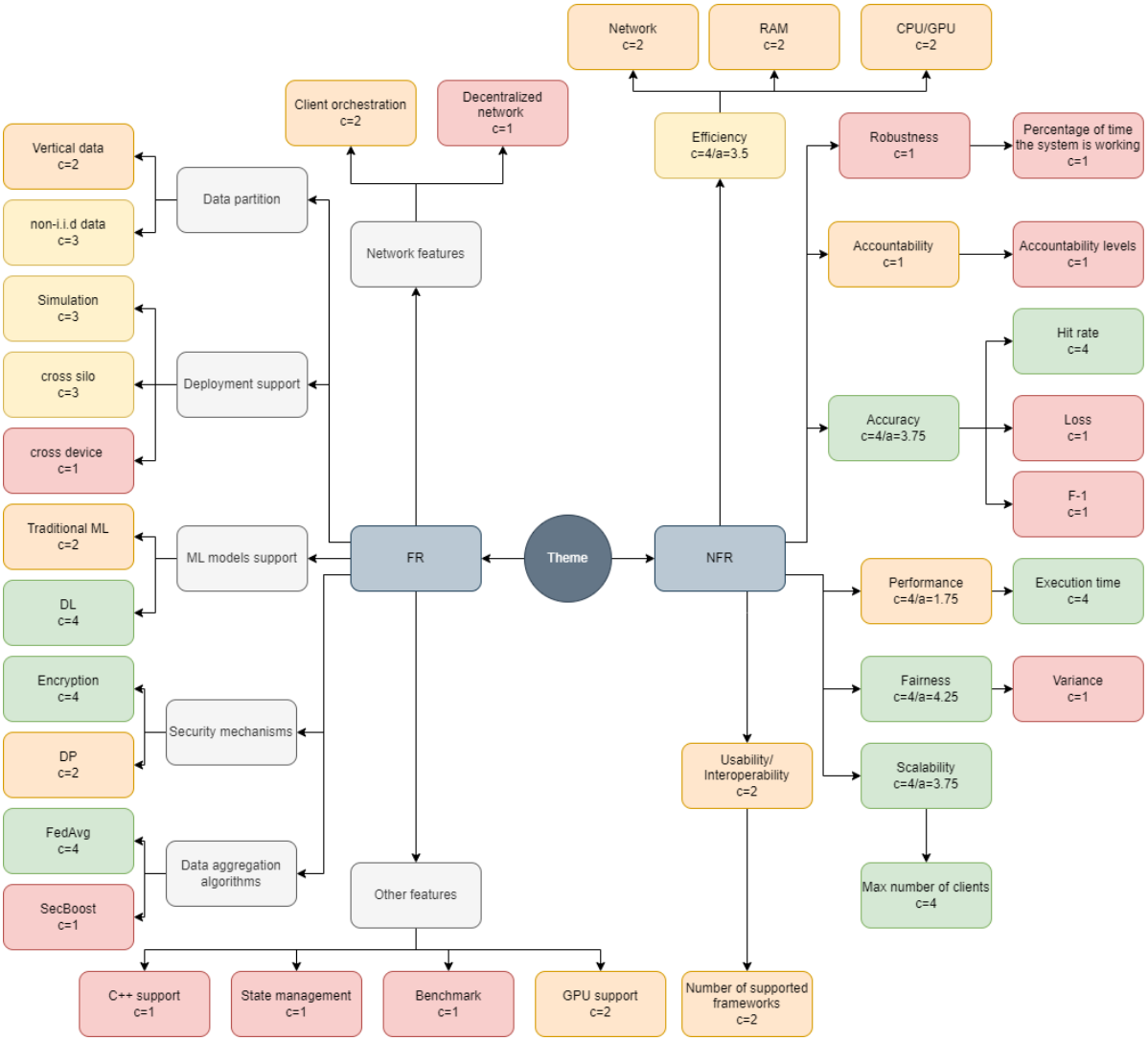


Figure 19.: Thematic encoding of the FR and NFR for FL systems (source: own work)

Figure 20 illustrates the bottlenecks for the adoption of federated learning as a technology. It was color-coded as follows: grey for the category, orange for the bottlenecks with a count of two, and yellow for the bottlenecks with a count of one.

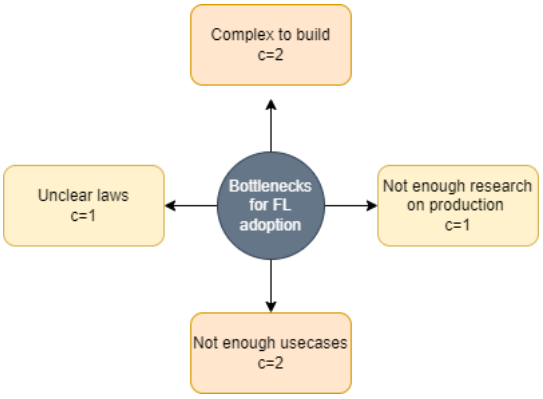


Figure 20.: Thematic encoding of the bottlenecks for FL adoption (source: own work)

Data analysis: The average interviewee is a researcher with 7 years of industry experience, and one year of experience researching, designing, and developing federated learning systems.

Table 9 summarizes the importance of functional requirements according to the interviews. It was encoded as follows:

- If c = 4 → The FR is very important (Green)
- If c = 3 → The FR is important (Yellow)
- If c = 2 → The FR is somewhat important (Orange)
- If c = 1 → The FR is not so important (Red)

Functionality	Feature	Importance
Network topology	Decentralized federated learning	Not so important
	Automatic clients orchestration	Somewhat important
Data partition support	Vertical data	Somewhat important
	Non-i.i.d data	important
Deployment support	Simulation	important
	Cross-silo	important
	Cross-device	Not so important
ML models	Traditional models	Somewhat important
	Deep learning models	Very important
Security mechanism	Encryption	Very important
	Differential privacy	Somewhat important
Data aggregation algorithm	FedAvg	Very important
	SecBoost	Not so important
Other features	C++ support	Not so important
	GPU support	Somewhat important
	State management	Not so important
	Native tests and benchmarks	Not so important

Table 9.: Importance of the different features for federated learning libraries according to the interviews (source: own work)

To encode the relevance of a non-functional requirement. a relevance-factor (R-factor) was introduced. The formula for the relevance factor is the following:

The R-factors are categorized as following:

$$R - factor = CountsOfMention \cdot AverageImportance$$

- 14 < r-factor<20 $\rightarrow$ The NFR is very important
- 9 < r-factor<15 $\rightarrow$ The NFR is important
- 4 < r-factor<10 $\rightarrow$ The NFR is somewhat important
- 0 < r-factor< 5 $\rightarrow$ The NFR is somewhat important

If two NFRs had the same relevance factor, they are sorted by the number of interviews in which they were mentioned as very important. The sorted list of the most important NFRs for FL systems and the metrics to measure them is the following:

1. Fairness (R-factor=17): The most important of all the NFRs, and also the most challenging to measure according to the interviews. Especially for classification tasks. Variance is the only mentioned metric that could somehow capture fairness in a federated learning system.

2. Accuracy (R-factor=15): A very important NFR, extremely easy to measure using the native metrics of ML frameworks (e.g loss, precision, hit-rate, false alarm rate, F1-score ...).

3. Scalability (R-factor=15): A very important NFR, It can be also tricky to measure, however, it could be benchmarked quite easily by setting a threshold for the number of supported clients in a federated learning system (e.g 1000 clients). It is classified as less important than accuracy despite the equal R-factor for the two NFRs because it was only mentioned 2 times as important, while accuracy was cited 3 times as important.

4. Efficiency (R-factor=14): An Important NFR, relatively easy to measure using the native OS monitoring functionality. RAM, network, CPU, and GPU usage could be all used as metrics for efficiency.

5. Performance (R-factor=7): A somehow important NFR, it could be measured easily by measuring the training time of an FL model.

6. Interoperability/Usability (R-factor=6): A somehow important NFR, It could be measured by the number of the ML frameworks the FL library could easily interoperate and integrate with. However, defining "easily interoperable" could be challenging.

7. Accountability (R-factor=5): A somehow important NFR, it reflects the level to which a system monitors and logs who is doing what, when, and where. However, there isn't any clear metric to measure accountability in software engineering. The solution put forward by one of the interviewees is to set different accountability levels and categorize FL systems in them.

8. Robustness (R-factor=4): Not so important. It could be measured by measuring the percentage of time the FL system works.

According to the interviewees, the main bottlenecks for federated learning adoption are the complexity of the technology, the compliance with data privacy laws, the lack of research and experimentations done in production, and the lack of use cases in the industry where federated learning could be effective to use. However, the sample size for the interviews was small due to the scarce number of people that work in the field of federated machine learning. This means that the results presented by this section is only primary and conclusions cannot be drawn from it about the preferences of the entire FL community.

5.2. Federated Learning Libraries and their Characteristics

The field of federated learning is a nascent research field. Most of the available libraries are still under active development and are still not production-ready. Some of the libraries have already died out. For instance, FLDL by Shepra.ai was mentioned in the scientific literature, but it was hard to find traces of it online. This part of the thesis deals with the presentation of the current state of libraries. The libraries are divided into prominent libraries

and non-prominent libraries. The prominent libraries need to meet two of the following three criteria:

- It has more than 1000 Github stars and 350 forks.
- It was mentioned by the experts in the interview series
- It supports all OSs.

The Github stars and forks are proxies for the use and development of the libraries. While, the interview mention is a proxy for the community awareness about the library. The OS support is a proxy for the potential adoption of the library. Table 10 illustrates the Github popularity of the different federated learning libraries. The prominent libraries were color coded in green, while the non-prominent onces were color coded in red.

	The number of stars	**Number of watchers**	**Number of forks**	**License**
Pysyft	8300	210	1800	Apache-2.0
FATE	4400	139	1300	Apache-2.0
Tensorflow Federated	1900	67	482	Apache-2.0
FedML	1400	36	406	Apache-2.0
Flower	1200	20	316	Apache-2.0
FedLearner	799	27	170	Apache-2.0
PaddleFL	404	25	103	Apache-2.0
OpenFL	364	17	108	Apache-2.0
IBM Federated learning	339	24	106	IBM license
EasyFL	222	13	37	No license
Flute	109	6	10	Apache-2.0
FedTree	59	5	8	Apache-2.0

Table 10.: GitHub popularity of different FL libraries and frameworks (source: own work)

Five libraries meet the first criterion of prominence. Which are Pysyft, FATE, FedML, TFF, and Flower. Pysyft, IBM federated learning, TFF, and Flower meet the second prominence criterion. FedML, Pysyft, IBM federated learning, TFF, and Flower meet the third criterion of prominence. The comparison was carried out only with the prominent libraries since they are the most adaptable and the most actively developed and used ones. However, the other libraries were also researched on a high level.

5.2.1. Prominent Federated Libraries

In this thesis, the prominent libraries are defined as the libraries meeting two of the three prominent library criteria or more. Five libraries meet these criteria. These libraries are Pysyft, FedML, Flower, IBM FL, and TFF.

Pysyft[50] is an open-source federated learning library developed by OpenMinded. It is distributed under the Apache-2.0 license and uses python 3.7. It promises secure and private distributed machine learning and can use both TensorFlow and PyTorch machine learning frameworks. The library can be run on a Mac, Linux, or Windows machine. It can also run in a dockerized environment. However, it is still in version 0.6.0 and is not yet production ready [50]. Pysyft has a large developer community (over 250 developers), comprehensive documentation, and an extended ecosystem. For instance, the functionalities of the library could be further extended by using the different tools developed as part of its ecosystem [51]. These libraries include:

- PyGrid is a peer-to-peer platform for federated learning and data science. It can be used by data scientists to train their machine-learning models without ever accessing the data used for training. The platform can enable Pysyft to run in federated mode.
- PyVertical is an extension to the Pysyft library to enable it to handle vertically partitioned data.
- Syft.js is the library that extends Pysyft to support web clients.
- KotlinSyft is the library that extends Pysyft to support android clients.
- SwiftSyft is the library that extends IOS to support android clients.

Standalone Pysyft can not yet run in a real federated learning environment and only supports a simulation environment. However, when used with PyGrid, Pysyft can be run in a federated environment. As illustrated in figure 21, the architecture of the Pysyft library is constituted of three layers. The first one implements the low level-code for the clients, as well as the virtual worker and the different machine-learning frameworks that Pysyft is built on top of. The second layer implements the different security mechanisms used by Pysyft (HE, DP, MPC ...). The last layer implements the machine learning models and the federated learning strategies supported by Pysyft [51].

[50]Pysyft: https://github.com/OpenMined/PySyft

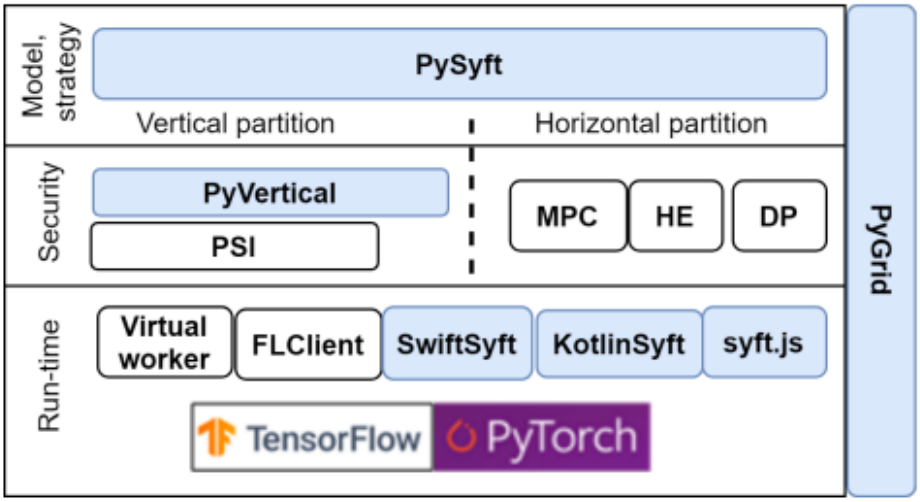


Figure 21.: The architecture of PySyft (source: [50])

Functionalities of Pysyft: Pysyft supports both vertical and horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time series). It uses gRPC as a communication mechanism and both encryption and differential privacy as security mechanisms. It uses FedAVG and FedSGD as Federated learning strategies and supports both traditional machine learning (regression, clustering, and Bayes networks . . .) and deep learning models (DNN, CNN, and RNN . . .). Pysyft also enables the customization of the network topology as well as the exchange of messages. it has native tests and benchmarks to analyze the applications built with it and supports the use of the GPU to train its deep learning models. Pysyft also supports multiple computing and federated learning paradigms like split learning, on-device training, distributed computing, edge computing, and simulations. Pysyft can also be used in both cross-device and cross-silo federated learning settings. In addition to that, pysyft applications can be deployed as a single simulation, multi-host (less than 16 clients), or cross-device (more than 100 clients) applications. In addition to its official documentation, code snippets as well as tutorials that demonstrate the functionalities and the capabilities of Pysyft can be found online [49].

Tensorflow Federated[52] is an open-source federated learning framework developed by Google initially to develop their mobile keyboard prediction, and ML models. It is distributed under the Apache-2.0 license and uses python 3. It promises an experimentation and simulation environment for FL algorithms and is built on top of the TensorFlow deep learning framework. The framework can be run on a Mac, Linux, or Windows machine. It can also run in a dockerized environment. However, it is still in version 0.34.0 and is not yet production ready [52]. Tensorflow Federated has a community of over 90 developers and comprehensive documentation. However, there is no ecosystem built around it [52]. Since it is not only built

[52]Tensorflow Federated: https://github.com/tensorflow/federated

to support ML engineers to build federated learning systems but also for data scientists, it also offers an additional functionality called federated analytics [53]. Federated analytics is used by ML engineers and data scientists to test their federated learning models on real-world data in a real data center. It tracks different metrics. Google uses this functionality for many different use cases like songs recognitions, and keyboard prediction. However, this functionality is not still not yet implemented in the open-source version of TFF [54]. Tensorflow federated offers two layers of APIs. These layers are:

- Federated Learning API is the API that enables developers to use the existing models and benchmarks as well as develop their own models. It is also called the tff.learning layer.
- Federated Core API is the foundation of the Federated Learning API. It implements the security and communication mechanisms used by TFF. The developers interact with this API to use the implemented federated learning strategies and develop their custom FL strategies.

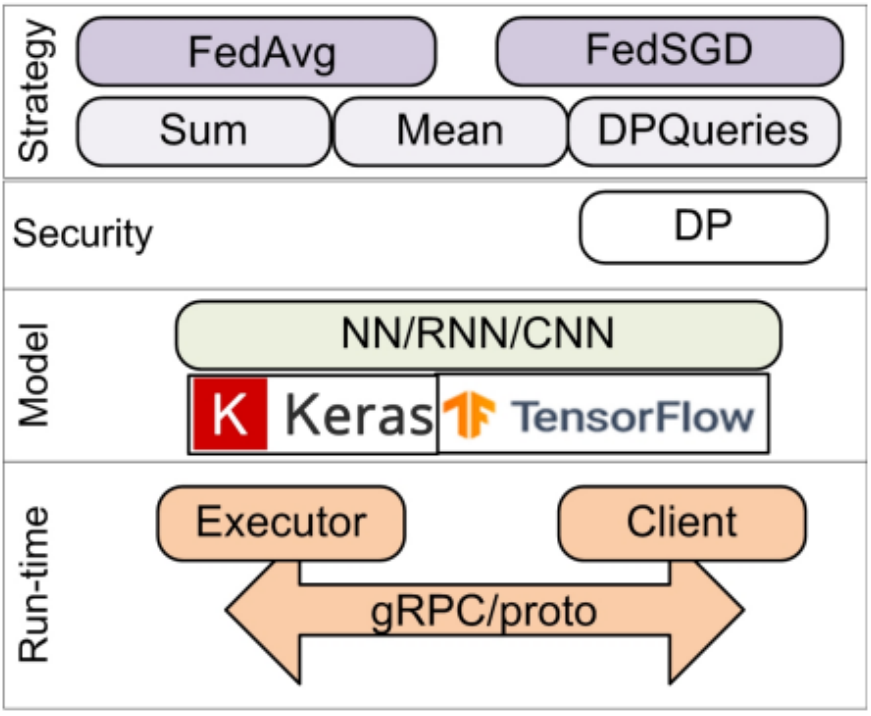


Figure 22.: The architecture of TFF (source: [52])

TFF has a multi-layered architecture. The first layer is the run-time layer. It implements the client, the communication protocols as well as the executor. The second layer implements the different supported deep learning models (DNN, RNN, CNN) using the Tensorflow framework. The third layer implements the differential privacy security mechanism. The

last layer implements the federated learning strategies (FedAvg and FedSGD) as well as the different differential privacy functionalities (Sum, Mean, DPQueries). TFF is the only FL library that offers DP functionalities. Figure 22 illustrates the architecture of Tensorflow Federated [52].

Functionalities of Tensorflow Federated: supports horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time-series). It uses both gRPC as well as its custom communication protocol as a communication mechanism and differential privacy as a security mechanism. It uses FedAVG and FedSGD as federated learning strategies, and only supports deep learning models (DNN, CNN, and RNN . . .). It also offers some relevant differential privacy functions like sum, mean, and DPQuerries.TFF doesn't offer any network or message customization. TFF has native tests and benchmarks to analyze the applications built with it and supports the use of the GPU to train its deep learning models. It also supports simulation as a federated learning paradigm and distributed computing as a computing paradigm. The framework is still in the development phase and many other paradigms are expected to be supported in the future. TFF can be only used in a cross-silo federated learning setting and does not offer any deployment model other than a single standalone simulation. In addition to its official documentation, code snippets as well as tutorials that demonstrate the functionalities and the capabilities of TFF can be found online [53].

FedML[59] is an open-source federated learning library developed by FedML Inc. It is distributed under the Apache-2.0 license and uses python 3.7. It promises a platform for building distributed and federated machine learning systems with high customizability. The library can be run on a Mac, Linux, or Windows machine. It can also run in a dockerized environment. However, it is still in version 0.7.98 and is not yet production ready [59]. FedML has a community of around 50 developers, and comprehensive documentation for both the framework and the applications built around it. it also has a large ecosystem to extend its functionalities. FedML has a two-layers architecture. The first layer consists of the low-level APIs (implementations of the workers, communication protocols, and security mechanisms . . .). The second layer contains the high-level APIs (implementations of FL strategies, ML models, tests and benchmarks, and mobile support . . .). The developer uses high-level APIs to build the system. Which in turn uses low-level APIs. Figure 23 illustrates the architecture of FedML [59]. Multiple tools are built around FedML as part of its ecosystems. These tools not only enhance the performance of FedML but also extends its functionalities. These tools are:

- Edge AI SDK is an SDK for developing ML models that train on edge devices. It can be used on the web, android devices, IoT devices (Raspberry Pi NVIDIA Jetson), as well as the cloud.
- MLOps Cloud is a user-friendly tool that deploys FL applications without the need for any code.

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

- FedML Parrot is a simulation tool that can be used to mimic the behavior of FL applications before deploying it in a real-world scenario.
- FedML Octopus is a cross-silo federated learning tool that enables collaboration between enterprises.
- FedML BeeHive is a tool that enables edge devices to collaboratively learn from each other.
- FedML cheetah is a tool that enhances the performance of FL systems built with FedML.
- FedML Enterprise is a full-package platform that offers a federated learning solution for enterprises. It is not part of the open-source project. The enterprise version of FedML includes all of the Edge AI SDK, MLOps, FedML Parrot, FedML Octopus, FedML BeeHive, and FedML cheetah. It also offers federated analytics functionalities for multiple use cases like statistics (e.g for clinical research), matrix operations (e.g for ads data science), and set operations (e.g for finance and retail). In addition to that, it offers a dashboard to visualize the workflow of MLOps and the federated analytics results.

In addition to the tools and applications developed by FedML Inc, multiple third-party benchmarks and frameworks have been developed as part of the FedML ecosystem [60]. These tools are:

- FedCV is a framework for computer vision tasks.
- FedIoT is a framework that extends the FedML IoT capabilities.
- FedGraphNN is a benchmark for graph neural networks.
- FedNLP is a benchmark for NLP tasks.

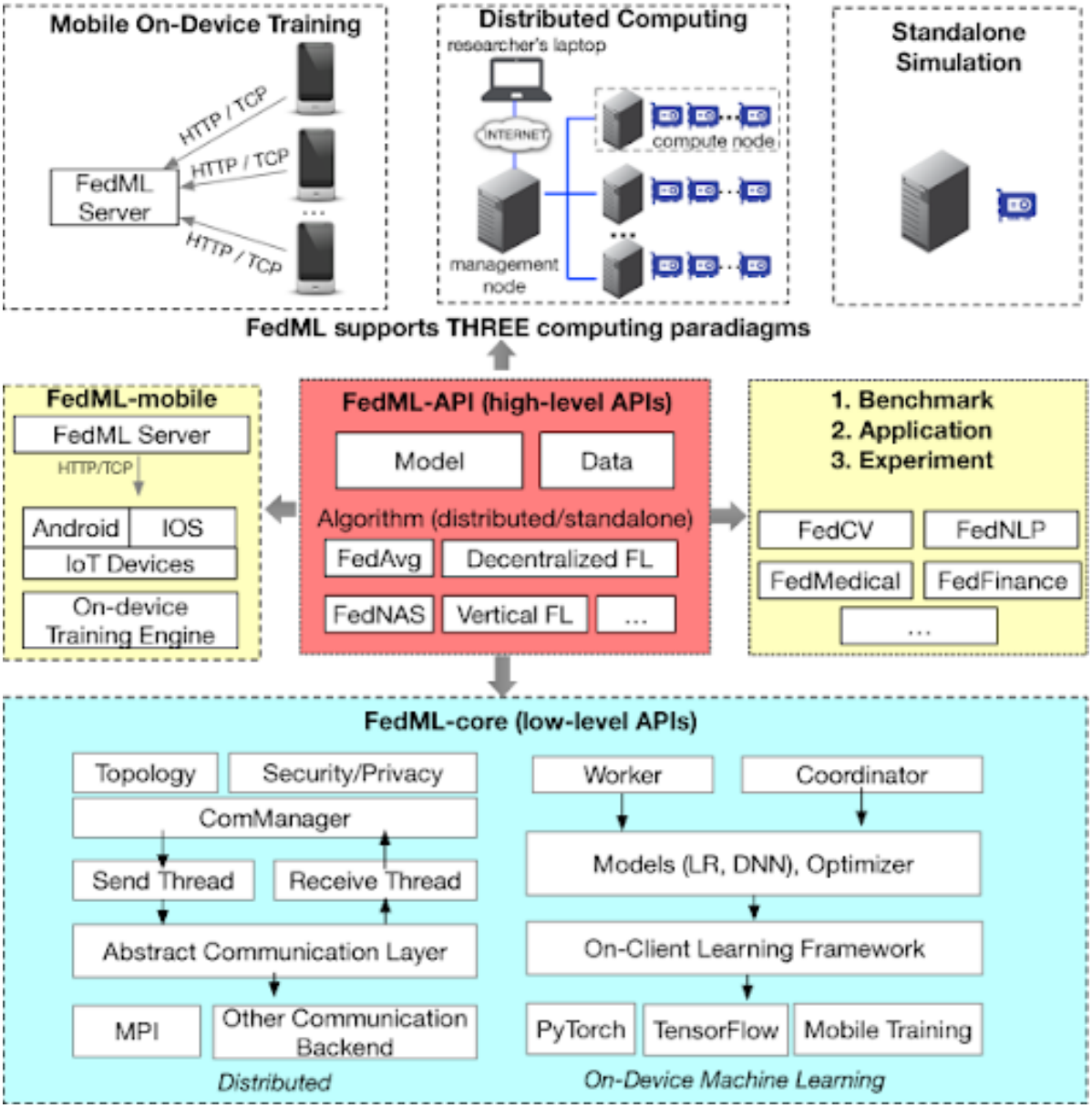


Figure 23.: The architecture of FedML (source: [59])

Functionalities of FedML: FedML supports both vertical and horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time series). It uses MPI, MQTT, and gRPC as communication mechanisms and encryption-based techniques as security mechanisms. It uses FedNOV, FedNAS, and FedAvg as well as other strategies as federated learning strategies, and supports both traditional machine learning (regression) and deep learning models (DNN, CNN, and RNN . . .). FedML offers network topology, exchange messages, and message flow customization. It has native tests and benchmarks to analyze the applications built with it and supports the use of the GPU to train its deep learning

models. FedML also supports multiple computing and federated learning paradigms like vertical FL, hetero-task learning, decentralized FL, on-device training, distributed computing, single-host simulation, edge computing, and split learning. FedML can be used in both cross-silo and cross-device settings. In addition to that, FedML applications can be deployed as a single simulation, cross-device (more than 100 clients), or multi-host (less than 16 clients) applications. In addition to its official documentation code snippets as well as tutorials that demonstrate the functionalities and the capabilities of FedML can be found online [59].

Flower[62] is an open-source federated learning library developed by Adap Gmbh. It is distributed under the Apache-2.0 license and uses python 3.7. It promises a customizable, understandable, extendable, and framework-agnostic source code. The library can be run on a Mac, Linux, or Windows machine. It can also run in a dockerized environment. Flower is in the version 1.1.0. So, the library has one stable release and is production ready. Flower has a community of around 51 developers, and comprehensive documentation. It doesn't have an ecosystem built around it. However, as it is one of the few production-ready libraries, it offers a Kubernetes cluster deployment in addition to the dockerized deployment. As a framework-agnostic library, It also supports the use of a wide range of ML frameworks like Tensorflow, Pytorch, Hugging face, JAX, Pytorch Lightning, MXNext, TFLITE (for android), and sci-kit-learn [62].

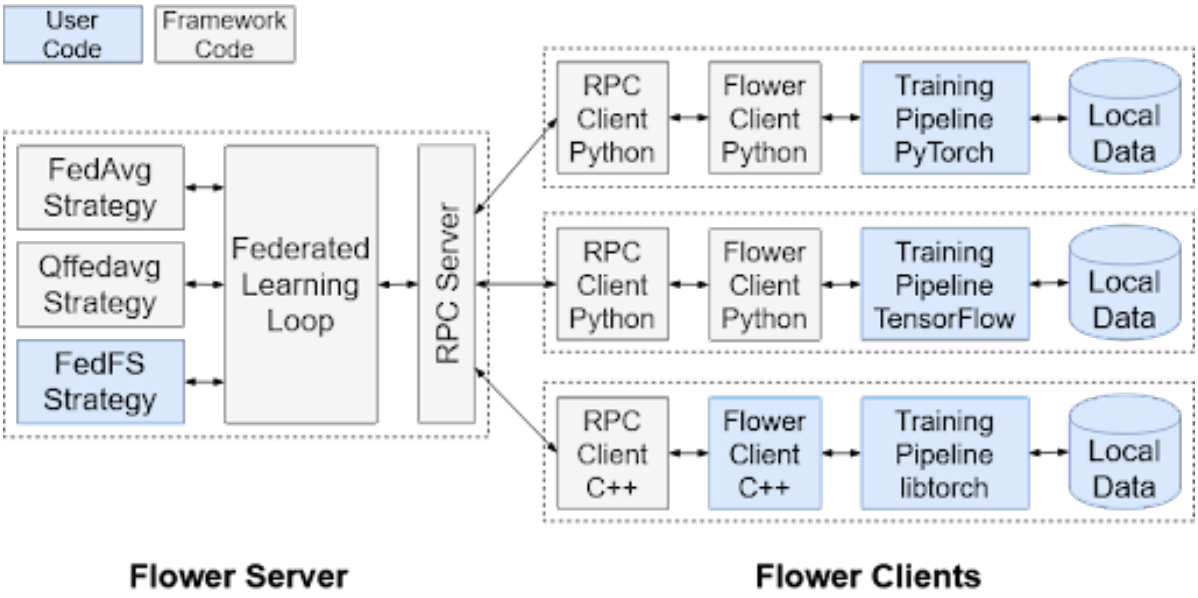


Figure 24.: The architecture of Flower (source: [62])

Flower has a client-server architecture. When building the FL applications some of the functionalities will be fully created by the developer and the rest of the functionalities will be implemented through the framework. For instance, the developer needs to use its dataset

[62]Flower: https://github.com/adap/flower

and build the entire ML training pipeline and the ML models on the client side but the library will implement the gRPC communication protocol to communicate with the server as well as the client interface to the server. On the server side, the library implements the gRPC communication protocol as well as the FL loops and some of the FL strategies. If the developer wants to use an FL strategy that is not supported by Flower, they have to develop it themselves. Figure 24 illustrates the architecture of Flower [63].

Functionalities of Flower: Flower supports both vertical and horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time series). It uses gRPC as a communication mechanism and SecAgg as a security mechanism. It uses FedAVG, FedFS, and qffedavg as well as other strategies as FL strategies, and supports both traditional machine learning (regression and clustering) and deep learning models (DNN, CNN, and RNN . . .). Flower offers exchange message customization. It has native tests and benchmarks to analyze the applications built with it and supports the use of the GPU to train its deep learning models. Flower also supports multiple computing and federated learning paradigms like vertical FL, personalized FL, on-device training, FL simulation, and edge computing. Flower can be used in both cross-silo and cross-device settings. In addition to that, Flower applications can be deployed as a single simulation, cross-device (more than 100 clients), or multi-host (less than 16 clients) applications. In addition to its official documentation code snippets as well as tutorials that demonstrate the functionalities and the capabilities of Flower can be found online [61].

IBM Federated learning[100] is an open-source federated learning library developed by IBM. It is distributed under a custom IBM open-source license and uses scikit-learn 0.23, Keras 2.2.4, and TensorFlow 1.15 as machine learning frameworks. It is designed for enterprises and can only be used in cross-silo FL settings. IBM Federated learning can work on any OS (Windows, Linux, and Mac), and can also use docker and Kubernetes as a setup environment. The library is in version 1.0.7 and it is production ready. IBM Federated learning supports many ML models like linear classifiers and regressions, logistic regression, linear SVM, ridge regression, Kmeans, Naïve Bayes, Decision Trees, Neural networks, and Deep reinforcement learning. It has only 10 contributors and comprehensive documentation. However, tutorials about libraries are scarce. The only official reference implementation is a demonstration of the library's capabilities on the MNIST dataset [65]. Despite being not as popular as the other prominent federated learning libraries. It was mentioned more than once in the interviews, and it is production ready, and can run in multiple environments. Thus, it was added to the list of the prominent libraries.

Functionalities of IBM Federated learning: IBM federated learning supports only horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time-series). It uses gRPC as a communication mechanism and encryption-based techniques as security mechanisms. It uses FedAVG and multiple other federated aggregation algorithms

[100]IBM: https://github.com/IBM/federated-learning-lib

as federated learning strategies and supports both traditional machine learning (regression, clustering, SVM, and trees ...) and deep learning models (DNN, CNN, and RNN ...). IBM federated learning does not offer any network or message customization. It does not have any native tests and benchmarks to analyze the applications built with it but it supports the use of the GPU to train its deep learning models. IBM federated learning also supports multiple computing and federated learning paradigms like, simulation, FL, personalized and hetero-task learning. IBM federated learning is a corporate federated learning framework and is designed to be only used in a cross-silo FL setting. In addition to that, IBM federated learning applications can be deployed as a single simulation or multi-host (less than 16 clients) applications. In addition to its official documentation, there isn't any material that demonstrates the functionalities and the capabilities of IBM federated learning [65].

5.2.2. Prominent Federated Libraries Feature Comparison

In addition to the non-functional quantitative quality characteristics, the federated learning frameworks and libraries also differ qualitatively in terms of functionalities. Table 11 summarizes the qualitative differences between them. It highlights the differences as well as the similarities in terms of features and functionalities between the different prominent federated learning frameworks and libraries. It also provides some general information like the main contributor of the library and the used ML frameworks to build it, as well as the environments that the library could run in.

Features/ Framework		**Pysyft**	**Flower**	**IBM FL**	**TFF**	**FedML**
General	**Contributor**	Open-Minded	Adap Gmbh	IBM	Google	FedML Inc.
	Environment	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker	Windows, Mac, Linux, Docker
General	**ML framework**	Pytorch	Pytorch, TF, Libtorch, JAX ...	SciLearn, Pytorch, TF, Keras	TF	Pytorch, TF
Architecture	**Data Partitioning**	Vertical Horizontal	Vertical Horizontal	Horizontal	Horizontal	Vertical Horizontal
	Datatypes	Numbers, Text, Image, Time-series	Numbers, Text, Image, Time-series	Numbers, Text, Image, Time-series	Numbers, Text, Image, Time-series	Numbers, Text, Image, Time-series

Architecture	**Communication scheme**	gRPC	gRPC	gRPC	gRPC, Custom Protocol	MPI, MQTT, gRPC...
	Privacy and Security	HE, MPC, DP	SecAgg	Multiple crypto-graphic methods	DP	Secret sharing key agree-ment
	FL Strategy	FedAVG, FedSGD	FedAVG, FedSGD , qffe-davg ...	FedAVG, FedProx, Fe-dAVG+...	FedAVG, FedSGD	FedAVG, Fed-NOV, Fed-NAS...
Paradigms	**Vertical FL**	yes	yes	no	no	yes
	FTL	no	yes	no	no	yes
	Cross device	yes	yes	no	no	yes
	Cross silo	yes	yes	yes	yes	yes
	Personalized	no	yes	yes	no	yes
	Hetero-task learning	no	yes	yes	no	yes
	Decentralized	no	no	no	no	yes
	Distributed computing	yes	no	no	no	yes
	Simulation	yes	yes	yes	yes	yes
	Edge comput-ing	yes	yes	no	yes	yes
	Split learning	yes	no	no	no	yes
	On-device training	yes	yes	no	no	yes
Engineering	**Customization**	topology, ex-change message	exchange message	none	none	topology, ex-change message, message flow
	GPU support	yes	yes	yes	yes	yes
	Native Bench-markS	yes	yes	no	yes	yes

Engineering	**Deployment**	single simulation, Multi-host (<16 clients), Cross-device (>100 clients)	single simulation, Multi-host (<16 clients), Cross-device (>100 clients)	single simulation, Multi-host (<16 clients)	single simulation	single simulation, Multi-host (<16 clients), Cross-device (>100 clients)
	Documentation	Detailed tutorial, Code Snippets, and API documentation	Detailed tutorial, Code Snippets, and API documentation	API documentation	Detailed tutorial, Code Snippets, and API documentation	Detailed tutorial, Code Snippets
ML Models	**Regression**	yes	yes	yes	no	yes
	Clustering	no	yes	yes	no	no
	Trees	no	no	yes	no	no
	SVM	no	no	yes	no	no
	Bayes networks	no	no	yes	no	no
	NN	yes	yes	yes	yes	yes
	DNN	yes	yes	yes	yes	yes
	CNN	yes	yes	yes	yes	yes
	RNN	yes	yes	yes	yes	yes

Table 11.: Differences between FL libraries (source: own work))

The functional requirements of the FL project heavily influences the choice of the library from a qualitative perspective. For instance, different libraries support different ML models. For NN all algorithms are suited. For traditional ML models, IBM federated learning offers the widest range of models. Pysyft and FedML support most computing paradigms and only FedML supports all of the FL paradigms researched in this thesis. TFF is not suitable for vertically partitioned data. FATE and TFF are not suited for a cross-device FL setting. FedML offers the highest level of network and exchange message customization. IBM federated learning and Flower are the only production-ready libraries.

5.2.3. Other Federated Libraries

In this thesis, the non-prominent libraries are defined as the libraries meeting less than two of the three prominent library criteria. Seven libraries meet these criteria. These libraries are FedLearner, FATE, EasyFL, PaddleFL, Flute, OpenFL, and FedTree.

FedLearner [64] is an open-source federated learning library developed by byteDance. It is distributed under the Apache-2.0 license and uses python 3. It uses TensorFlow as an ML framework. It has some implementations of some popular ML models and FL strategies, as well as some metrics and benchmarks. The library can work on Linux and Mac machines and needs docker and Kubernetes as a setup requirement. The library is currently in version 1.5 and it is production ready. It has a community of over 40 developers. However, It doesn't offer any documentation and no tutorials. There are some reference implementations in the project but they do not demonstrate the full capabilities of the library [64].

FATE[56] is an open-source federated learning framework developed by Webank. its full name is federated AI technology enabler. It is distributed under the Apache-2.0 license and uses python 3.6. It promises an industrial-grade federated learning open-source framework and can use both TensorFlow and PyTorch machine learning frameworks. The framework can run only on a CentOS 7+ Linux machine. It can also run in a dockerized environment. FATE is in version 1.8.0. So, the framework has nine stable releases and is production ready [56]. FATE has a community of around 70 developers, comprehensive documentation for both the framework and the applications built around it, and a large ecosystem to extend its functionalities [56]. It is one of the few production-ready federated learning frameworks and offers a Kubernetes cluster deployment in addition to the dockerized deployment. Multiple tools constitute the FATE ecosystem [55]. These tools are:

- KubeFATE is a tool that supports the deployment of FATE applications via docker-compose for the development environment and Kubernetes for the production environment.
- FATE-Flow is a task scheduling tool for building federated learning pipelines using FATE.
- FATE-Board is a tool for visualization that helps with the understanding of the different federated learning models by visualizing different metrics and logs.
- FATE-Serving is a system that helps federated learning servers to serve their models to different clients. It helps with the creation of robust, highly performant, and scalable systems.
- FATE-Cloud is a tool that enables the management of FATE applications on the cloud. It offers functionalities like the registration of federated sites, access control, cluster de-

[64]FedLearner: https://github.com/bytedance/fedlearner
[56]FATE: https://github.com/FederatedAI/FATE

ployment and monitoring, and multiple other functionalities. FATE-Cloud is constituted of two main components. Namely, Federated Cloud (Cloud Manager) and Federated Site (FATE Manager).

  - Cloud Manager manages the infrastructure of FATE applications on the cloud.
  - FATE Manager manages the FATE applications on the cloud.

- EggRoll is a framework to conduct high-performance machine learning computations using FATE.
- AnsibleFATE is a tool that enables FATE to interact with the Ansible software to automate its operations and configuration. Ansible is software that enables the automation of operations, configuration, and deployment of other software applications.
- FATE-Builder is a tool that helps with the packaging and building of the docker images of the software applications built with FATE.

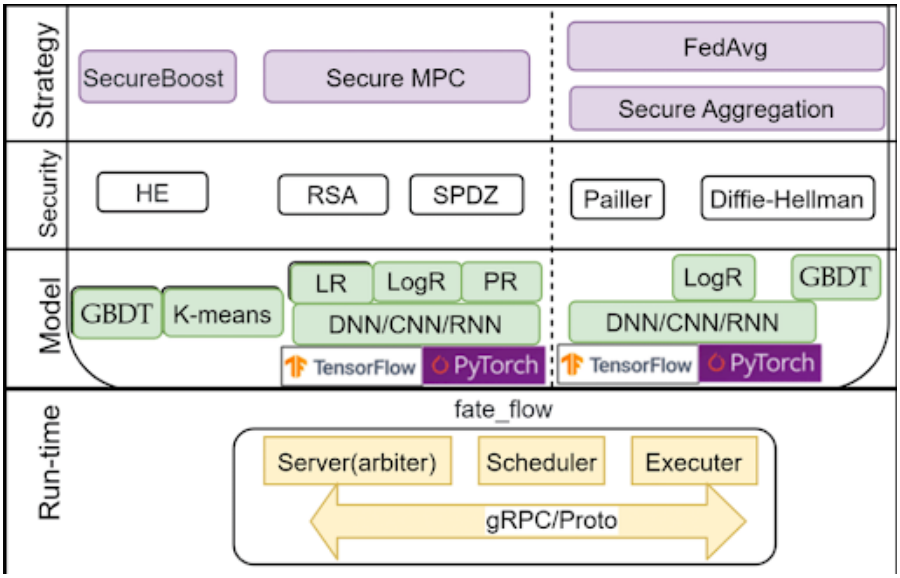


Figure 25.: The architecture of FATE (source: [56])

FATE is maintained by the TSC (Technical Steering Community). The TSC Board makes the decisions about the direction of the project, while the TSC Maintainers contribute to the further development of the project. FATE has a multi-layered architecture. It is constituted of four layers. The run-time layer is the runtime layer and it implements the low-level functionalities of the system. Namely, the gRPC communication protocol, the scheduler, and the executor. The second layer is the model layer, which implements all the machine learning models like regression, clustering, and neural network models. The third layer implements the encryption-based security mechanisms (RSA, HE, SPDZ . . .). The last layer implements federated learning strategies like SecureBoost and FedAvg. Figure 25 illustrates

the architecture of FATE [58].

Functionalities of FATE: FATE supports both vertical and horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time-series). It uses gRPC as a communication mechanism and encryption-based techniques as security mechanisms. It uses FedAVG as a Federated learning strategy and supports both traditional machine learning (regression, clustering, SVM, and trees ...) and deep learning models (DNN, CNN, and RNN ...). FATE does not offer any network or message customization. It has native tests and benchmarks to analyze the applications built with it and supports the use of the GPU to train its deep learning models. FATE also supports multiple computing and federated learning paradigms like distributed computing, simulation, edge computing, vertical FL, federated transfer learning, and hetero-task learning. FATE is an industrial-grade federated learning framework and is designed to be only used in a cross-silo FL setting. In addition to that, FATE applications can be deployed as a single simulation or multi-host (less than 16 clients) applications. In addition to its official documentation, code snippets as well as tutorials that demonstrate the functionalities and the capabilities of FATE can be found online [57].

EasyFL[66] is an open-source federated learning library developed by Śmietanka, M. et al. [66]. It is distributed under the Apache-2.0 license and uses python 3 and PyTorch as an ML framework. The library promises lightweight, easy-to-develop FL systems implementations. It is mainly used to experiment with federated learning rather than to develop actual FL systems. It can run on Linux, Mac, and Windows machines. It has only 3 contributors and offers official documentation that presents the architecture and the capabilities of the library. The library does not have any release yet even though the project can be found on GitHub [66].

Flute[67] is an open-source federated learning library developed by byteDance. It is distributed under the MIT license and uses python 3.8. Its full name is Federated Learning Utilities for Testing and Experimentation. The library is not made to develop actual FL systems but rather to experiment with federated learning. Flute supports the use of both the CPU and GPU for models' training and offers an implementation of the most popular NNs. It also can be extended with more models and functionalities and can scale to accommodate millions of clients. It has 6 contributors and a small documentation document on its GitHub repository. There are some reference implementations in the project that demonstrate how to use the library for the experimentation of FL. Flute does not have any release [67].

OpenFL[68] is an open-source federated learning library developed by Intel. It is distributed under the Apache-2.0 license and uses python 3.6. It uses TensorFlow and PyTorch as ML frameworks. It supports the use of multiple FL strategies like FedAvg, FedPro, FedOpt, and FedCurv... OpenFL is developed for the IoT use case and can only run on Ubuntu machines.

[66] EasyFL: https://github.com/WwZzz/easyFL
[67] Flute: https://github.com/microsoft/msrflute
[68] OpenFL: https://github.com/openfl/openfl

It has a community of over 35 developers and some documentation on its Github repository. However, It doesn't offer any tutorials [68].

FedTree[69] is an open-source federated learning library developed by Li, Q. et al. (2022) [69] . It is distributed under the Apache-2.0 license and uses python 3. The library is designed to develop tree-based federated learning applications both for vertical and horizontal datasets and does not offer any support for any ML model beyond that.The library can run on both Linux and Mac machines. It is currently in version 1.0.4. It has only 5 contributors and does not offer any documentation beyond the scientific publication that describes it and the description on its GitHub repository. There are no tutorials for it online as well [69].

PaddleFL[71] is an open-source federated learning library developed by Webank. It is distributed under the Apache-2.0 license and uses python 3.6. It also uses PaddlePaddle as an ML framework. It promises a solution for the development and deployment of large-scale enterprise FL systems. The library can be run on Mac and Linux machines. It can also run in a dockerized environment as well as in Kubernetes clusters. PaddleFL is in version 1.2. It has three stable releases and is ready to be used in a production environment [71]. It has a community of 17 contributors and comprehensive documentation. The tutorials available as well as part of the documentation are mainly written in Chinese. In addition to the FL strategy and the ML model. PaddleFL offers the customization of a third paradigm called training strategy. The training strategy customization allows the user to choose the model training paradigm easily (e.g Transfer learning, active learning, or Hetero-Task learning). It also allows the user to train the FL model in a distributed FL environment (without the need for a central server to coordinate) with the use of the Distributed-Config functionality. The FL-JOB-GENERATOR coordinates the worker, scheduler, and server to execute the program. PaddleFL does not use a traditional client-server architecture to build its FL applications but instead something similar. It has a server that is similar to the server in the client-server architecture and a worker and a scheduler that do the job of the client. The worker trains the ML models while the scheduler coordinates the different workers to communicate with the server [70]. PaddleFL has multi-layers architecture. It has four layers. The first layer is the run-time layer, it implements the communication mechanism as well as the worker and scheduler and the code that interacts with PaddlePaddle. The second layer is the models' layer. It implements the different machine learning models that PaddleFL uses like NN and regression. The third layer implements the security mechanisms (DP and encryption algorithms). The last layer implements the different FL strategies that PaddleFL uses (FedAvg, SecAgg, PSI, DPSGD ...). In addition to the library being vertically segregated, it uses different security mechanisms, FL strategies, and ML models for vertical and horizontal datasets. Figure 26 illustrates the architecture of PaddleFL [70].

[69]FedTree: https://github.com/Xtra-Computing/FedTree
[71]PaddleFL: https://github.com/PaddlePaddle/PaddleFL

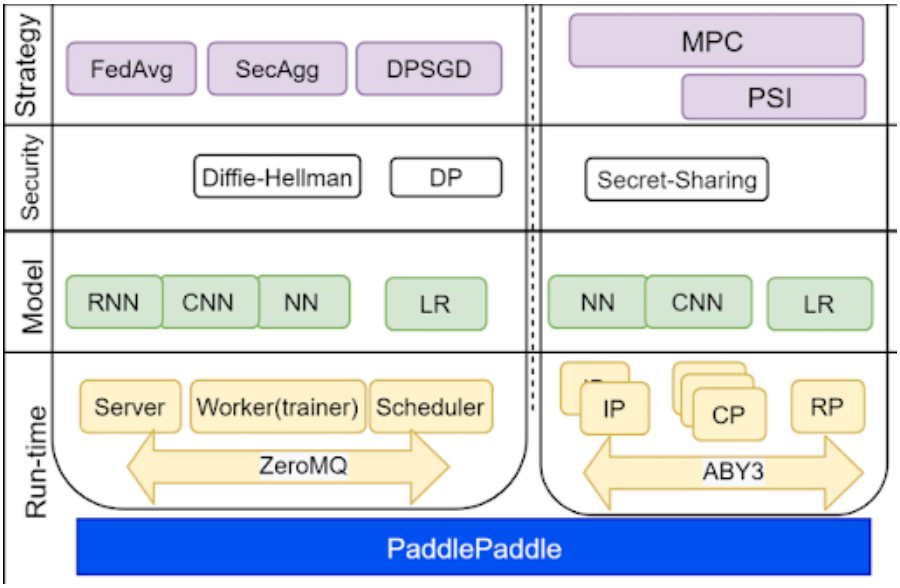


Figure 26.: The architecture of PaddleFL (source: [71])

Functionalities of PaddleFL: PaddleFL supports both vertical and horizontal data partitioning as well as different datatypes for the datasets (numbers, text, images, and time-series). It uses ZeroMQ as a communication mechanism and both encryption and DP as security mechanisms. It uses FedAVG, SecAgg, and MPC as well as other strategies such as Federated learning strategies, and supports both traditional machine learning (regression) and deep learning models (DNN, CNN, and RNN . . .). PaddleFL does not offer any customization over the network topology or the exchange of messages. It does not have native tests and benchmarks to analyze the applications built with it but it supports the use of the GPU to train its deep learning models [70]. PaddleFL also supports multiple computing and federated learning paradigms like vertical FL, Distributed FL, FTL, Split learning, FL simulation, and edge computing. PaddleFL can be used only in a cross-silo setting. In addition to that, PaddleFL applications can be deployed as a single simulation or multi-host (less than 16 clients) applications. It is resource intensive. It requires a minimum of 6 GB RAM and 100GB HDD. Even though it supports on-device training and cross-device deployment the resource requirements make it not suited for cross-device FL settings (more than 100 clients). Compared to other FL libraries, PaddleFL is mature and offers a wide range of capabilities. Even though PaddleFL is production ready and supports more functionalities than most of the prominent FL frameworks and libraries, it has one of the poorest community support and a small number of contributors. This could be attributed to three main reasons. Namely, it doesn't run on windows machines (the most used OS worldwide), the fact that documentation is mostly written in Chinese, and the fact that it uses PaddlePaddle as an ML framework not that popular outside of China [70].

5.3. The Federated Learning Benchmarking Suite

FMLB (the federated machine learning benchmark) [84] is an easy-to-use benchmark suite that can benchmark different federated learning settings using a web application. The suite contains a web application, different implementations of federated machine learning models using different federated learning libraries, as well as two datasets to train the federated learning models. FMLB is designed to be modular, easily extendable, and easy to use. It doesn't require any technical knowledge to benchmark a federated learning setting using the tool. Everything is handled through a web user interface.

5.3.1. Problem Identification and Solution Objectives

According to the DSR methodology, before designing and developing a tool. The tool needs to provide a clear solution for an identified problem and contribute to scientific research on that problem. FMLB does that by providing an easy-to-use FL benchmark to compare the FL libraries for the scientific community and practitioners. The need for FMLB was identified after a literature review of the scientific literature around federated learning that concluded the lack of such a benchmark.

Problem identification: The scientific literature around federated learning is focusing more on the bottlenecks of federated learning as well as the different use cases of federated learning. While some publications have dealt with the benchmarking of federated learning systems, they usually tend to benchmark the different ML algorithms or federated learning strategies. However, the literature on the benchmarking of the different federated learning libraries is scarce. Only one benchmark dealt with the comparison of the different libraries. Namely, Unifed [29]. While Unifed is rich in functionality and has a high configurability, it does not fulfill the usability criteria of what makes an adequate benchmarking suite [25]. It requires some technical knowledge to be used and can be only used through the CLI. FMLB is developed as a solution for this issue. It is designed for ease of use, modularity, and extensibility.

Solution objectives: Since the benchmark is designed for ease of use, it should fulfill the following functional and non-functional requirements that were aggregated based on the results of the expert interviews and literature review:

- Functionality
  - The user can create a federated learning setting in the benchmark.
  - The user can update a federated learning setting in the benchmark.
  - The user can delete a federated learning setting from the benchmark.
  - The user can see what are the available federated learning settings in the benchmark.
  - The user can create an ML model in the benchmark.
  - The user can see what are the available ML models to use in the benchmark.

  - The user can delete an ML model from the benchmark.
  - The user can create a dataset in the benchmark.
  - The user can see what are the available datasets to use in the benchmark.
  - The user can delete a dataset from the benchmark.
  - The user can train an existing federated learning setting using the benchmark.
  - The user can train ML models in both a federated and a non-federated setting.
  - The benchmark needs to have pre-configured federated learning settings as well as different datasets included in it.
  - The benchmark needs to provide persistent data storage for the settings so they don't have to be recreated every time a user wants to benchmark any of the existing settings.
  - In case of the execution of a successful operation or an exception, the user should be notified via a pop-up.
  - The user can easily configure the FL setting, they are willing to train and benchmark.

- Usability
  - The benchmark needs to have an easy-to-use UI.
  - The benchmark can also be used through the CLI.

- Modularity
  - The components of the benchmark need to follow the single responsibility principle where each component is only responsible for its function and doesn't interfere with other components.

- Extensibility
  - New datasets can easily be added to the benchmark.
  - New machine learning models, FL strategies, and FL libraries can be easily added to extend the settings that the benchmarks already support.
  - The benchmark can be easily extended with more features.
  - In case the benchmark doesn't have a dataset it should be easily downloadable.

- Reproducibility
  - The results delivered by the benchmark for the same setting on the same machine should be the same.

- Verifiability
  - The metrics delivered by the benchmark need to be reliable.

- Fairness
  - The benchmark should have the exact same implementation for the same setting for the different libraries.

FMLB is designed to fulfill all of the above-mentioned requirements. Through its easy-to-use web interface.

5.3.2. Design and Development of the Benchmarking Suite

After identifying the research gap and defining the objective solution. The solution needs to be designed and developed. FMLB is a software tool that fulfills all of the requirements of the solution objective. FMLB is a web application that comes with different implementations of different federated learning models using different libraries, and two datasets, as well as ten different metrics to benchmark these implementations. The suite can be easily extended with more federated machine learning models, implementations, and datasets.

Tech stack and High-level architecture: FMLB is a client-side rendered web application. The frontend was built with React [79], zustand [80] for state management, and Axios [81] for sending API requests and communicating with the backend. While the backend was built with nodejs[82]. The database is a MongoDB [83] document-based database. The backend also includes multiple federated and traditional machine learning models that are implemented in python. It also includes datasets to train the federated learning models. The backend communicates with the ML models through the terminal. Figure 27 illustrates a high-level architecture of the FMLB benchmarking suite.

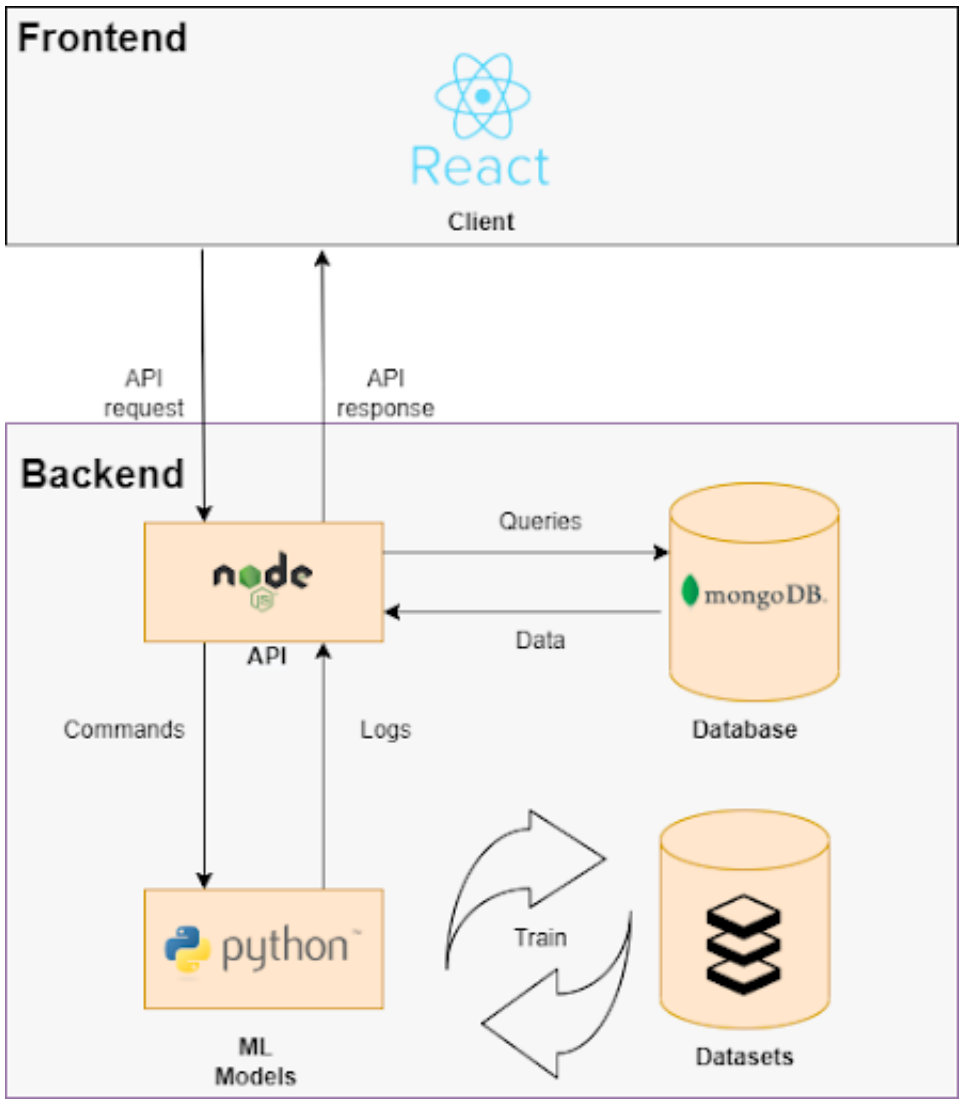


Figure 27.: The architecture of FMLB (source: own work)

The FMLB project structure: The FMLB benchmarking suite is all included in one project. The project is open source (distributed under the Apache-2.0 license) and can be found on GitHub [84]. The project has two directories, the first directory for the client, and the second directory for the server. The client directory contains four different directories. A directory for the pages, a directory for the components, a directory for the assets, and a directory for the services. While the server directory contains four different directories. A directory for the routers, a directory for the controllers, a directory for the data models, and a directory for the federated machine learning models implementations. The federated learning directory is further divided into three different directories. A directory for the datasets (It is also divided into other directories each for a dataset), a directory for the utilities like the declaration of the models, the configuration of the training, and the data loaders. The last

directory in the machine learning module is for the implementation of the training using the different libraries. The implementation directory is further divided into other directories each containing the implementation of the different training settings using different algorithms. FMLB was used to compare the non-functional quality metrics of the different libraries (e.g Accuracy, Precision, Recall, F1, RAM usage, Network usage, CPU usage, GPU usage, and Time of execution). A series of thirteen experiments were done, the first one was done with PyTorch without the involvement of any client, and twelve experiments were done with the four libraries in three scalability levels (2, 16, and 100 clients).

Client: The frontend is a react application. It uses zustand for state management, MUI as a UI library [85], and Axios for communication with the backend. The application has three assets, three pages, eleven components, and one service. They are the following:

- Assets: The assets are the static images that are included in the app. There are currently three images available under the directory. An open-source SVG image of a benchmark logo (used as the application's logo) [88], and two PNG images of MUI icons (used as images for the react components) [99].
- Pages: The pages are what the user sees and interacts with. The application has three different pages. The home page, the utility page, and the training page
  - The home page: The home page is the landing page that the user is first directed to. Úpon rendering, the home page sends an API request to the backend to get all of the existing FL settings and displays them one under the other in a table. It displays the different attributes of the settings under their respective title.
  - The utility page: Like the home page, the utility page sends an API request to the backend to get all of the available ML models and datasets. It then renders them into cards containing all of the required information about the datasets (e.g name of the dataset, the location of a dataset, the number of data points in a dataset, or the size of a dataset) or the model (e.g the model used, the library it is implemented with, and the name of the python script to train that model). In addition to that, the utility page has two "add cards". The first card can be used to add a new ML model, and the second card can be used to add a dataset to the application. The script for the model and the dataset, however, need to be added manually to the backend. The cards only act as an interface for the user.
  - The training page: Each federated learning setting has its own training page. Upon rendering the training page sends two API requests to the server. The first one is to get the different attributes of the setting (e.g library, version, environment name, script name, model, FL strategy, number of communication rounds, GPU, mode, epochs, batch size, learning rate, loss, and optimizer). The second one is to tell the server to train a specific federated machine learning model and get the metrics of the training as a response (e.g loss, precision, recall, F1, accuracy, time of execution, GPU usage, CPU usage, RAM usage, and Network usage).

- Components: The components are the building blocks for the pages. Each page is made of one or more components. In total, the application has eleven different components. Namely:
  - The header: The header has a clickable logo on the left that routes the user to the home page. In addition to that, it has a small blue plus button that opens the setting creation dialog, and also a small blue squared button to redirect to the utility page. The header is a sticky header that appears on all pages of the application.
  - The create dialogs: The application has three different creation dialogs. The first dialog is used to create a federated learning setting, the second dialog is used to create a dataset, and the third dialog is used to create an ML model. All creation dialogs contain required input fields that need to be filled in order to create their respective components. Some of the input fields do contain placeholders. The data entered in these input fields will be later used as attributes for the component. All create dialogs have two buttons at the bottom. The first button is used to create the component. The second button is used to cancel the operation.
  - The delete dialogs: The application has three different deletion dialogs. The first dialog is used to delete a federated learning setting, the second dialog is used to delete a dataset, and the third dialog is used to delete an ML model. The deletion dialogs take the ID of the to-be-deleted component through props. The dialogs display a warning and two buttons. The first button is the cancel button. It reverts the operation. The second button is the deletion button. When the deletion button is clicked, the frontend sends a delete request to the backend to delete the component.
  - The cards: There are three different cards. The “add card”, the model card, and the dataset card. The dataset card displays information about the datasets (dataset name, dataset size in MB, number of data points in a dataset, the URL to the dataset, and data format). The model card displays information about the models (model name, library name, library version, environment name, and the python script used to train that model). The “add card” can be used to add a dataset or a model. The cards are all displayed on the utility page. the datasets and models are each grouped together with an “add card” at the end to add another dataset or a model.
  - The setting table row: The setting table row is the component that displays the information about the settings (identifier, library, library version, ml model, federated learning strategy, clients number, communication rounds, dataset, GPU, mode, epochs, batch size, learning rate, loss function, optimizer . . .). The rows are displayed on the home page one under the other in the settings table.
- Services: The services implement application-wide functionalities that are not bound to a single component or page (e.g state management). The application only has one service:

  - useState: The state management service. It is built with zustand. It saves the states of the different components, as well as their setters, and operations to add and remove objects from these states

Each of the components and the pages has both a javascript file for the logic and the template structure (rendered with JSX) and a CSS file for the styles. In addition to that, the application has the App.js file which defines the routing logic within the application and acts as the root of the client side of the application.

Server: The server is a node server that runs on the node virtual machine. It communicates with the different FML implementations through the CLI using child processes, and with the MongoDB database through queries. The server has a multilayered architecture. The first layer is for the router, the second layer is for the controller, the third layer is for the model, and the last layer is for the machine learning implementation. The last layer is the only layer that is implemented in python with the help of multiple other packages, thus it is not counted as part of the server. The first three layers are the ones implemented in node.js and they serve the following functions:

- Routers: The routers route the API requests coming from the frontend to the controller function that handles them in the backend. In total, the backend has twelve functions to handle the API requests from the frontend, and thus it has twelve routers. five GET routers for getting a specific setting, getting all the settings, training a setting, getting all the datasets, and all the models. Three delete routers which are for deleting a setting, a dataset, or a model. Additionally, a PUT router that can be used to update a setting, and POST routers which can be used to create a setting, model, or dataset.
- Controllers: The controllers implement all the logic that handles the API requests from the backend. The controllers of the FMLB have twelve different main functions and one helper function divided between the different controllers as follows:
  - The settings controller: The controller is responsible for manipulating the settings in the backend. It has six different main functions and one helper function. Namely:
    * getSettings: This function queries the database for all of the available settings and sends them back in the form of an array of JSON objects.
    * getSetting: This function gets the id of the needed setting as a request parameter. It then queries the database to find it. Once found, it sends it back as a JSON object to the frontend.
    * trainSetting: This function takes the id of the to-be-trained setting as a request parameter and then queries the database to find it. Once it finds it, It checks for the library used, the name of the environment it is implemented in, its directory name, and the name of the script that is used to train the ML model of that setting. It uses this information to send a command as a child process through the CLI to start training the ML model in the respective setting. It also forwards other information like the batch size, the number of communication

rounds, and the number of clients to the python script as arguments. The logic for launching the commands differs from library to library. For instance, for the centralized model, and for pysyft only one command is required to launch the training. While for flower, each client has its own command plus one for the server. FedML, the commands include YAML files. After that, the backend waits for the response of the script, which will be a long log. It takes the log and sanitizes it with the help of the sanitize helper function. A JSON object containing all of the relevant metrics will then be created. This object will be then sent to the frontend. The IBM library could not be trained through the web application, despite it being part of the benchmark. It can be only used through the CLI because the user needs to interact with the library in a way that could not be automated.

  * deleteSetting: This function gets the id of the to-be-deleted setting as a request parameter. It then queries the database to delete it. Once deleted, it sends back a response to the frontend to notify it about the operation's success.
  * updateSetting: This function takes three attributes from the request body. Namely: the id of the setting, the attribute that needs to be changed (called a key), and its new value (called a value). It searches for the setting using its id. Once found, It checks which attribute to change, and gives it its new value. In the end, it sends back a response to the frontend to notify it about the operation's success.
  * createSetting: This function takes all of the 13 attributes of a setting and creates a new setting using them in the database (It also checks if the identifier is unique, and if each value is consistent with its schema). After that, it sends a response to the frontend to notify it about the operation's success.
  * sanitize (helper function): The sanitize function takes all the logs provided by the FL library during the training. It then deletes all of the irrelevant information from them, extracts the different metrics and tokenizes them into an array, casts their values, and shortens them up. Finally, it returns them as a JSON object.

- The model controller: The controller is responsible for manipulating the models in the backend. It has three different main functions. Namely:
  * getModels: This function queries the database for all of the available models and sends them back in the form of an array of JSON objects.
  * deleteModel: This function gets the id of the to-be-deleted model as a request parameter. It then queries the database to delete it. Once deleted, it sends back a response to the frontend to notify it about the operation's success.
  * createModel: This function takes all of the 5 attributes of a model and creates a new model using them in the database (It also checks if the identifier is unique, and if each value is consistent with its schema). After that, it sends a response

to the frontend to notify it about the operation's success.

- The dataset controller: The controller is responsible for manipulating the datasets in the backend. It has three different main functions. Namely:
  - getDatasets: This function queries the database for all of the available datasets and sends them back in the form of an array of JSON objects.
  - deleteDataset: This function gets the id of the to-be-deleted dataset as a request parameter. It then queries the database to delete it. Once deleted, it sends back a response to the frontend to notify it about the operation's success.
  - createDataset: This function takes all of the 5 attributes of a dataset and creates a new dataset using them in the database (It also checks if the identifier is unique, and if each value is consistent with its schema). After that, it sends a response to the frontend to notify it about the operation's success.
- Models: The models implement the mongoose schema of the different settings, models, and datasets. Each has its own model. The settings model has 13 different attributes and a unique _id. The ML model has 5 different attributes and a unique _id. The dataset model has 5 different attributes and a unique _id. The models tell the database how the data needs to be structured in the objects representing the FL settings, ML models, or datasets.

In addition to that, the server has an index.js file that starts it and defines the port to communicate with it as well as its routers. It serves as the main endpoint to the entire backend. It also includes the db.js file that is used to connect to the MongoDB database.

Machine learning: The machine learning layer gets activated either through the CLI or through the training function in the backend. The following command is used to tell the application which machine learning model to run in which conda environment: conda run -n environment name python path/to/the/model/training/script–args arguments. The first part of the command tells the application which environment to use, and the second part tells it which python script to run. The last part configures the training setting. Figure 28 illustrates the architecture of the machine learning layer of the application. The layer implements federated machine learning models as well as traditional machine learning models.

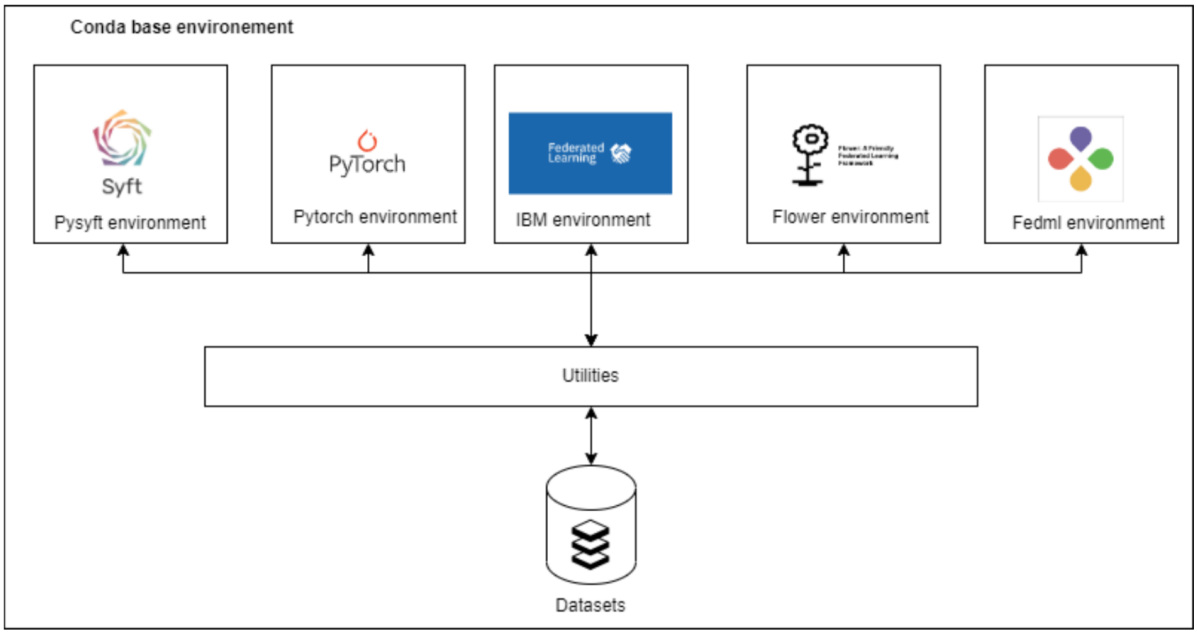


Figure 28.: The architecture of the machine learning layer (source: own work)

As illustrated in figure 28 above, the machine learning logic of the application is further divided into three distinct layers. Each of the layers has its distinct responsibility The layers and their responsibilities are the following:

- The implementation layer: The implementation layer has all the logic for training, testing, and benchmarking the different machine learning settings. The logic is implemented using the different libraries with the help of helper packages to track and retrieve the different metrics. The application currently has multiple implementations in four different federated learning libraries (IBM federated learning, pysyft, flower, and fedml), and one traditional machine learning implementation with PyTorch.
- The utility layer: The utility layer has all the utilities that the different libraries use to effectively train the models and retrieve the metrics. Namely, the implementation of the ML models, the data loaders for the different datasets, as well as a config file that takes the config arguments.
- The datasets layer: The datasets layer has all the datasets used during the training. Currently, only two datasets are available. Namely, the MNIST [86] and CIFAR-10 [87] datasets.

The Implementation layer: has all the logic to train, test, and benchmark the libraries. The layer has four different federated learning implementations (All of them use PyTorch as a machine learning framework), and one for a centralized machine learning setting for reference which is also implemented in PyTorch. The implementations work as follows:

- Pytorch (centralized model): The PyTorch centralized implementation first checks the config class to get the configuration and gets what model and dataset to use as well as get the configuration of the setting. Then the timer starts. During that time the data is loaded into the model and the model is trained and tested using it. During the training, the CPU, GPU, RAM, and network usage are monitored. After the training, the timer stops. During the testing, the precision, recall, accuracy, loss, and F1 are calculated and retrieved. All of the metrics are then logged. The centralized model can train a CNN with both the MNIST and the CIFAR-10 datasets. It can also train a logistic regression model using the MNIST dataset. This implementation serves more as a reference for the benchmarking of the other implementations.

- Flower: The flower implementation is divided into a server and a client implementation. The server starts a timer when it starts. Each of the clients gets the configuration of the setting, loads the dataset, and trains and tests its model. It also tracks the precision, recall, F1, accuracy, and loss of the model. After that, It sends the metrics to the server. The server averages them and saves them. It also tracks other metrics like CPU, GPU, RAM, and network usage. It stops the timer at the end. It logs them. The flower implementation can only train a CNN on the MNIST dataset. However, the implementation of the server can work with clients training any model on any dataset.

- Pysyft: The pysyft implementation first creates the different clients involved in the training of the dataset.Then, it checks the config class to get the configuration and gets what model and dataset to use as well as get the configuration of the setting. Then the timer starts. The model is distributed to different clients. During that time the models are trained and tested using the dataset. The data is loaded to the model using a federated data loader that loads the data to the different clients. During the training, the CPU, GPU, RAM, and network usage are monitored. After the training, the timer stops. During the testing, the precision, recall, accuracy, loss, and F1 are calculated and retrieved. All of the metrics are then logged. Pysyft can train a CNN both on the MNIST and the CIFAR-10 datasets.

- IBM federated learning: The IBM implementation is only made of YAML files. A file for the server, and a file for each of the clients. All the logic for implementing the training and testing as well as the model is included in the source code of the library. The library is also added to the source code of the benchmark. The library is extended to support the logs of the metrics that are needed to benchmark the training. IBM federated learning can train a CNN on the MNIST dataset. However, adding new models and datasets is easy, and doesn't require any coding. Unless the user wants to develop their own custom model or use their custom dataset. The IBM federated learning implementation can not be accessed through the UI of FMLB. Instead the user needs to use it through the CLI.

- FedML: The FedML implementation only implements the logic of running a new training setting as well as some of the metrics. The implementation takes as an argument a

YAML file that has the different attributes of the setting. The source code of the library logs the precision, recall, accuracy, loss, and F1 values. While the CPU, RAM, GPU, Network usage, and time are monitored through the script. FedML can train both logistic regression and a CNN on the MNIST datasets. In order to use another model or dataset, a YAML file should be created for it. Custom models and datasets, need to be added in their own python script and then called by the training script.

Each of the libraries implements the training differently. That's why logs were chosen as the method to get the metrics as it is universal. The pysyft and the PyTorch training settings could be launched as a normal python script. The IBM training could be done only through the YAML files of the server and the different clients. The fedml training could also be launched like any python script but it needs to take a YAML file as an argument. The file needs to have the config of the training setting. In flower, the server and each of the clients need to be called on their own in order to enable the training. Furthermore, the implementation layer makes use of slightly different versions of the fedml and the IBM federated learning libraries. They were modified to log metrics that are not logged by the original version of the libraries (e.g precision, recall, and F1) to make benchmarking the two libraries easier. No further modification of the behavior of the two libraries was conducted.

The utility layer implements the utilities needed by the libraries to train and benchmark a model. It has three python files. The first one is for the ML models, the second one is for the configuration of the models, and the third one is for the data loaders. They are implemented as follows:

- Models: The application has three models implemented. Two CNN models, the first for the MNIST and the second for the CIFAR-10 datasets, and a logistic regression model for the MNIST dataset. The models and their layers are already defined and can be called directly.
- Configuration: The configuration file implements the config class. The config class takes the different arguments (The model to be trained, the used federated learning strategy, and dataset to be used, batch size, number of epochs, number of clients and communication rounds, the mode, if the training should make use of the GPU or not, and the learning rate), and passes them to the training script. The python script then configures the training setting accordingly and starts the training of the model using that particular configuration.
- Data loaders: There are currently two data loaders implemented in the application, the first one for the CIFAR-10, and the second one for the MNIST datasets. They each load the training dataset and the testing dataset separately. In case the data loader doesn't find the dataset in its respective directory, It automatically downloads it.

The datasets layer is the layer responsible for keeping the raw datasets that will be used by the machine learning models for their training. Each of the datasets has its own directory. In that directory, there are two further directories, one for the training dataset and another one for

the testing dataset.

Currently, FMLB has two distinct datasets:

- The MNIST dataset: This dataset is made of uncolored images. The images are pictures of random numbers from 0 to 9 (each of the numbers corresponds to a class). It is mainly used to benchmark CNN. The CNN models need to predict the number of images. The dataset has 70000 data points in total (60000 for training and 10000 for testing) and weighs 30 MB in total.
- The CIFAR-10 dataset: This dataset is made of colored images. The images are divided into 10 classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks). Each class has 6000 images which make up to 60000 images in total (50000 for training and 10000 for testing) and weigh 175 MB. The dataset is also mainly used to benchmark CNN. The NN needs to predict the class of the given image.

Models and datasets support: Not all of the implemented models and datasets can be used with all of the libraries. Table 12 illustrates which library supports the use of which models and which datasets.

	CNN with CIFAR-10	**CNN with MNIST**	**Logistic Regression with MNIST**
Centralized (Pytorch)	X	X	X
Pysyft	X	X	
IBM federated learning		X	
Flower		X	
FedML	X	X	X

Table 12.: Models and datasets supported by FMLB (source: own work)

As Illustrated in table 12 above, the CNN model with the MNIST dataset is implemented in all the libraries. Thus, it will be used in the benchmarking experiment.

The application could be also extended in the future to support more models, databases, and libraries. Thus, more benchmarking experiments can be carried on with it in the future. The datasets could be directly added to the datasets directory, and a data loader in the data loaders file should be created for them. For the models, they need to be declared as classes in the model's file. Then, a file for the training, testing, and benchmarking logic of the library need to be created under the directory of the library that will be used to train the model. In order to add a new library, a conda environment [90] needs to be created, and the library should be installed in that environment, then a directory should be created for it under the libraries directory (the directory needs to have the same name as the library). The files for the training, testing, and benchmarking logic can be then added under that directory. The ML

layer is designed to be modular so deleting or creating a new model, dataset, or setting, as well as updating an already existing setting will not interfere with the functionality of the other existing settings.

5.3.3. Demonstration of the Benchmarking Suite

FMLB [84] enables CRUD operations over the federated machine learning settings created with the application. The read operation is performed on the home page where the user can see all the federated learning settings created, and on the utility page to see all the models and datasets available to create new settings. The creation operation is performed through the creation dialogs that enable the creation of a new federated machine-learning setting, model, or dataset. the delete operation is performed through the deletion dialogs specific to each setting, ML model, or dataset. The update operation is a backend-only function that can be performed only with the use of an API platform like postman [91]. In addition to that, the user can train the different federated machine learning settings available on the application to get their performance across different non-functional metrics. However, to be able to use the tool, some software programs need to be installed.

Prerequisites to use the benchmarking suite: The benchmark suite is a PoC and is not yet deployed. In order to be able to use it effectively, the user needs to have the following software programs installed on their machine:

- python v3.7 [92]
- conda v22.0 [90]
- pip v22.3 [93]
- npm v6.14 [94]
- node v19.0 [82]
- react v18.0 [79]
- MongoDB v6.0 [83]

After that, the user can clone the project from GitHub[84], enter into both the client and server directories through the CLI, and run the command npm i to install all of the frontend and backend dependencies. Once the dependencies are installed, the user needs to create a local database with MongoDB, and create a .env file in both the server and the client directories with the following environment variables:

[92]python: https://www.python.org/
[90]conda: https://docs.conda.io/en/latest/
[93]pip: https://pypi.org/project/pip/
[94]npm: https://www.npmjs.com/
[82]node: https://nodejs.org/en/
[79]react: https://reactjs.org/
[83]MongoDB: https://www.mongodb.com/

- On the frontend:
  - REACT-APP-API-URL: the endpoint to the backend (e.g http://localhost:5000/api/settings/)
- On the backend:
  - PATH-TO-LIBRARIES: absolute path to the libraries' directory
  - PORT: on which port the backend should run (e.g 5000)
  - DB: the URL to the database (e.g mongodb://localhost:27017)

In addition to that, the benchmarking suite has already some pre-configured implementations of already existing federated learning settings that can be used immediately. For that, the user needs to create a conda environment for each of the already pre-existing libraries and install each of them in that environment. The libraries that need to be installed are the following:

- fedml v0.7.3 [59]
- pysyft v0.2.9 [52]
- PyTorch v4.4.0 (for a centralized on federated setting) [95]
- IBM federated learning v1.0.7 [65]
- flower v1.1.0 [62]

Once the user has successfully installed all the software and added the environment variables required for the tool to run, the benchmark can be used. In order to run the client, the user needs to get to the client directory through the CLI and run the command npm start. In order to run the server, the user needs to get to the server directory and run the command node index.js. The pre-configured settings can be easily added to the database. The important attributes are the identifier, the model, the dataset, and the name of the python script that encapsulates the logic. These attributes need to match exactly the names used in the implementation of the benchmark. The other attributes don't matter as they will be passed as arguments to the machine learning models, and the user has the freedom to give them whatever value they want. Table 13 illustrates the different possible configurations that could be immediately added and trained on the tool.

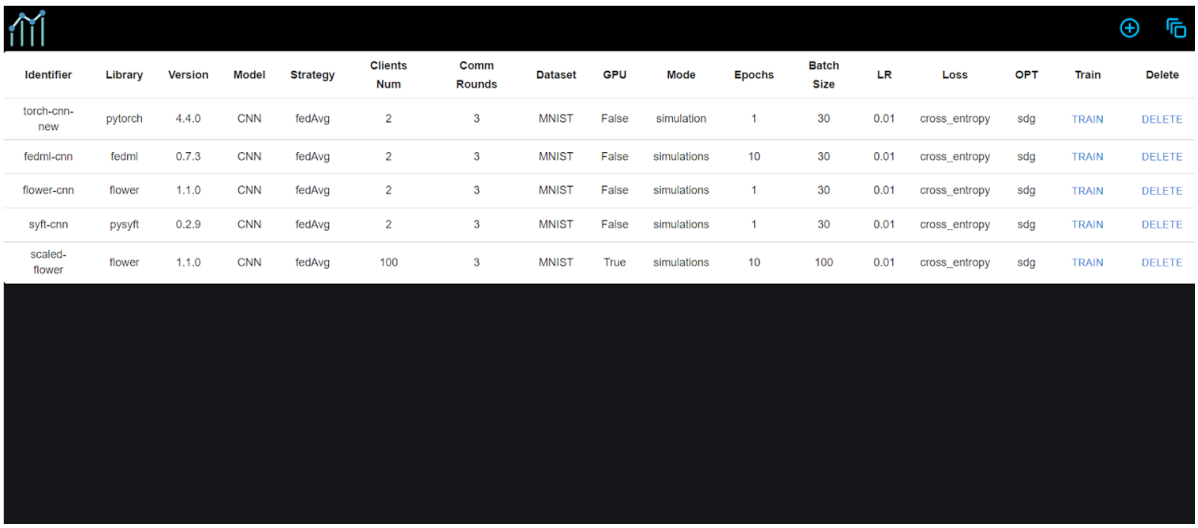
Attribute	**Possible Values**	**Comment**
Identifier	Any unique value	Needs to be unique in order to identify the setting

Dataset	MNIST or CIFAR	CIFAR can be used with all of Fedml, pysyft, and the centralized model. MNIST can be used with all of the libraries. A dataset needs to be created first on the utility page. If the created dataset is MNIST, the data points number should be 70000, the size should be 30 MB, and the URL should be /mnist.If the created dataset is MNIST, the data points number should be 60000, the size should be 163 MB, and the URL should be /cifar-10. Both datasets are image datasets
Model	CNN or LogReg	Only logistic regression and CNN are currently implemented in the benchmark (CNN with the MNIST dataset is implemented in all of the libraries). While the logistic regression is only implemented with IBM, the centralized model, and FedML. To add a model, the model needs to first be created on the utility page. The model needs to have a name, a library, a library version, and a name for the conda environment it runs in. In addition to that, a name for the python script that implements its training in that specific library. For instance, for PyTorch, the name for the script is image-classifier, for flower, the name of the script is mnist-image-classifier-cnn-client, for FedML, it needs to be mnist-cnn-config or cifar-cnn-config, or mnist-logreg-config, and for pysyft, it needs to be image-classifier-cnn. All the names need to be consistent with the implementation, otherwise, the application won't be able to run the experiments

Table 13.: Possible configurations for already implemented models (source: own work)

Home page: As illustrated by figure 29 below the home page retrieves all the settings from the database and displays them on a dashboard. It displays information such as the identifier of the setting, the library used, the version of the library, the number of clients and communication rounds involved, the dataset used, as well as its format, size, and the number of data points. It also displays other important attributes such as the ML model, FL strategy, number of epochs, batch size, loss function, and optimizer used for the experimental setting.

The information specific to the models and datasets like the library used, version, dataset name, and dataset size are retrieved from the ML model and dataset associated with the setting. Each setting has a train and delete button to train or delete that setting. In addition to that, there is a blue plus button on the right side of the header that enables the user to create a new setting, and a blue squared button to navigate to the utility page. The home page acts as the main page of the application. It summarizes the information about all of the existing settings. From the page, a setting can be created, trained, or deleted. The user can also navigate to all of the other pages of the application.



Identifier	Library	Version	Model	Strategy	Clients Num	Comm Rounds	Dataset	GPU	Mode	Epochs	Batch Size	LR	Loss	OPT	Train	Delete
torch-cnn-new	pytorch	4.4.0	CNN	fedAvg	2	3	MNIST	False	simulation	1	30	0.01	cross_entropy	sdg	TRAIN	DELETE
fedml-cnn	fedml	0.7.3	CNN	fedAvg	2	3	MNIST	False	simulations	10	30	0.01	cross_entropy	sdg	TRAIN	DELETE
flower-cnn	flower	1.1.0	CNN	fedAvg	2	3	MNIST	False	simulations	1	30	0.01	cross_entropy	sdg	TRAIN	DELETE
syft-cnn	pysyft	0.2.9	CNN	fedAvg	2	3	MNIST	False	simulations	1	30	0.01	cross_entropy	sdg	TRAIN	DELETE
scaled-flower	flower	1.1.0	CNN	fedAvg	100	3	MNIST	True	simulations	10	100	0.01	cross_entropy	sdg	TRAIN	DELETE

Figure 29.: A screenshot of the home page (source: own work)

Utility page: As illustrated by figure 30 below the utility page retrieves all the ML models and datasets from the database and displays them on the page in a card. In addition to that, there are two “add cards”. The first card can be used to add a model, and the second card can be used to add a dataset. The model card displays information such as the model type, the library it is implemented in and its version, the name of the conda environment it runs in, and the name of the python script that implements the training of the model. The dataset card displays information such as the dataset name, the format of the dataset, the size of the dataset in MB, and the number of data points in the dataset. In addition to that, each of the model cards and dataset cards has a delete button at the bottom that can be used to delete the component.

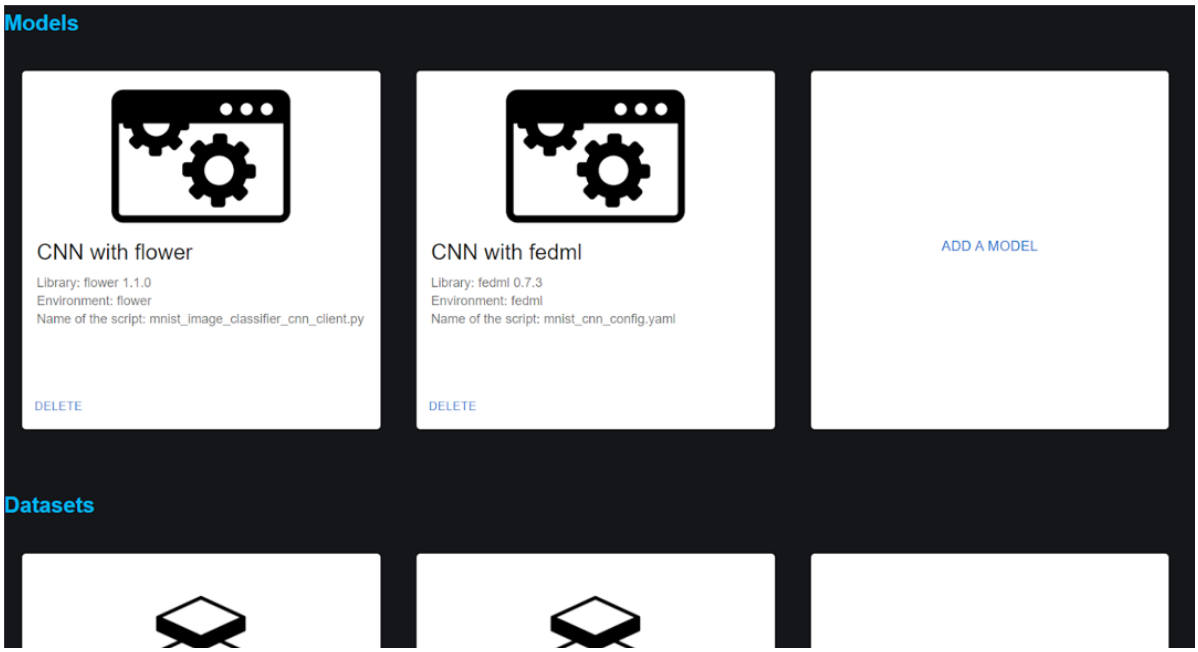


Figure 30.: A screenshot of the utility page (source: own work)

Train an existing federated learning setting: In order to train a setting, the user needs to click on the train button specific to the setting. The user will be redirected to a page that needs some time to load. figure 31 illustrates the training page.

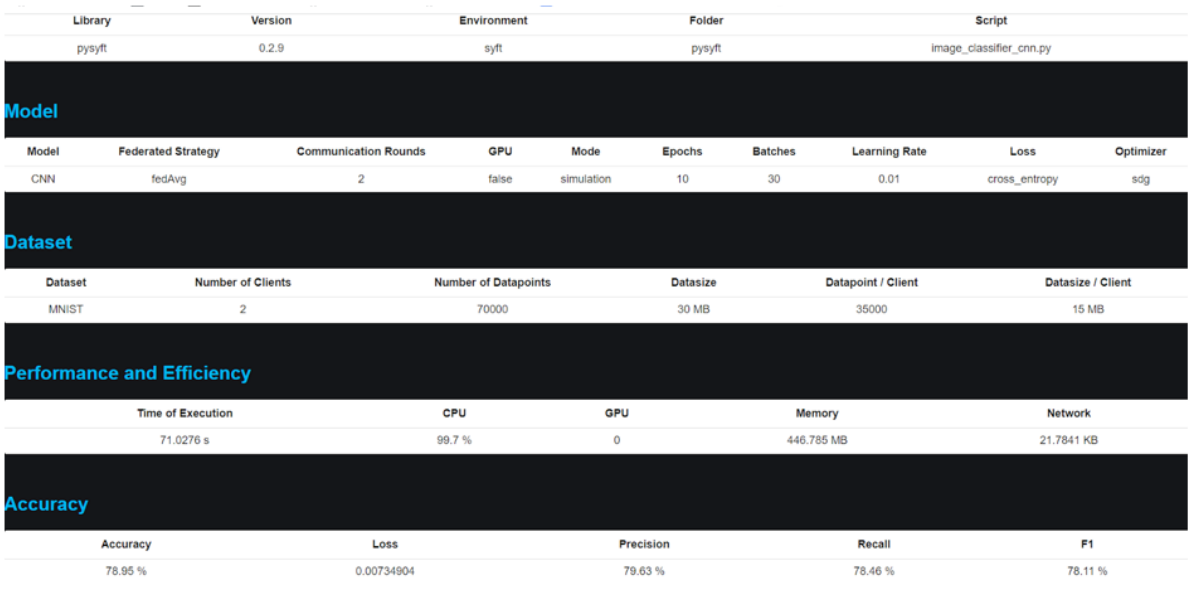


Figure 31.: A screenshot of the training page (source: own work)

Once loaded, the page will display the following information:

- General information: Information that the user has entered when creating a new setting like: the library and its version, the ML model and FL strategy used, if a GPU is used, the number of epochs, the batch size, the environment name, folder name, and file name ...
- Scalability metrics: Metrics for how many clients are involved in a setting, and how much each client is handling.
  - Number of clients: The number of clients involved in the setting.
  - The number of data points per client: The number of data points each client is handling.
  - The data size per client: How much data each client is handling in MB.
- Performance and efficiency metrics: Metrics for the time of execution and the resource utilization of the setting.
  - Time of execution: How much time did the model take to be trained?
  - RAM: How much RAM was used during the training?
  - CPU: How much CPU was used during the training?
  - GPU: How much GPU was used during the training?
  - Network: How much network was used during the training?
- Accuracy metrics: Metrics for how well the model has performed.
  - Accuracy: The percentage of the correct prediction.
  - Loss: How much the model fits the dataset.
  - Precision: Reflects the quality of the correct prediction.
  - Recall: Reflects the percentage of true positives recalled.
  - F1: A score that averages out the precision and recall to give an idea about the overall quality of the model.

Create a new federated learning setting, model, or dataset: In order to create a setting, the user needs to click on the blue "plus" button on the right of the header. If the user wants to create a dataset or a model, the user needs to click on the "add model" card or add "dataset card" on the utility page. A dialog as displayed in figure 32 will appear. Most of the setting attributes are preconfigured to make it easier to configure the setting quickly. The attributes that the user needs to enter are the following:

- Identifier: A text input unique to the setting.
- Model: A dropdown menu with the available ML models and the library they are implemented in.
- Federated strategy: A dropdown menu with the available FL strategies.

- Client number: A numerical input with the number of clients involved.
- Communication rounds: A numerical input with the number of communication rounds involved.
- Dataset: A dropdown menu with the available dataset.
- GPU: A drop-down menu to indicate if the setting makes use of the GPU or not.
- Mode: A drop-down menu with the training mode (e.g simulation, cross-silo, cross-device).
- Epochs: A numerical input with the number of epochs.
- Batch size: A numerical input with the batch size.
- Learning rate: A numerical input with the learning rate.
- Loss function: A drop-down menu with the loss functions (e.g cross entropy).
- Optimizer: A drop-down menu with the optimizer (e.g SDG).

For the model creation, the required attributes are the following:

- Model: A dropdown menu with the available ML models.
- Library: A dropdown menu with the available libraries.
- Version: A text input reflecting the version of the library being used. It needs to have the following format X.X.X.
- Environment: A drop-down menu with the conda environment's names.
- Script: A text input for the name of the python script that implements the training of the ML model.

For the dataset creation, the required attributes are the following:

- Dataset: A dropdown menu with the available datasets.
- Dataset URL: A text input with the path to the dataset. It needs to have the following format path/to/dataset.
- Data format: A drop-down menu with the formats of the dataset (e.g image, text, time-series, tabular, sound ...).
- Data size: A numerical input with the size of the dataset in MB.
- Datapoints number: A numerical input with the number of data points in the dataset.

After configuring the attributes the user can click cancel, to cancel the operation or click create to create a new setting, model, or dataset. A pop-up will appear to inform the user of the successful operation.

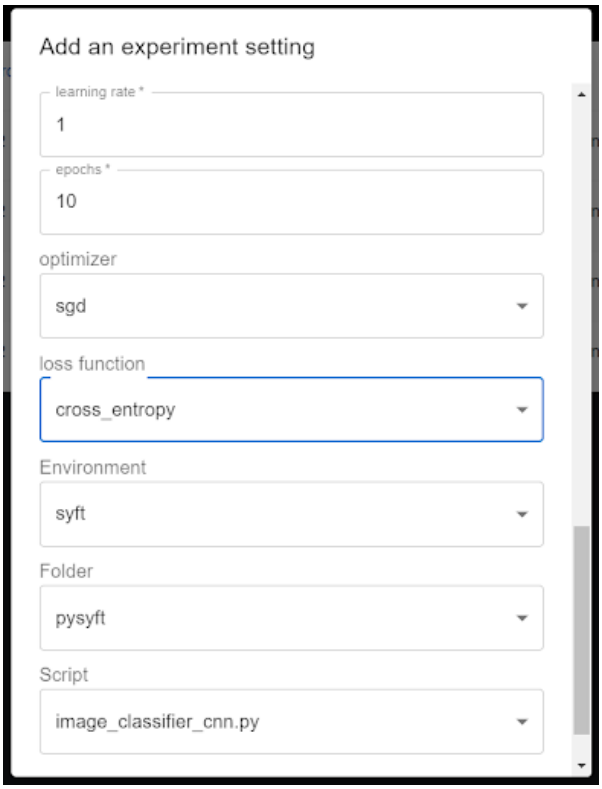


Figure 32.: A screenshot of the create dialog (source: own work)

Delete an existing federated learning setting, model, or dataset: In order to delete a component the user needs to click on the delete button specific to the component. A dialog as displayed in figure 33 will appear. If the user clicks cancel, the operation is canceled. If the user clicks delete, the component is deleted from the database. A pop-up will appear to inform the user of the successful delete operation.

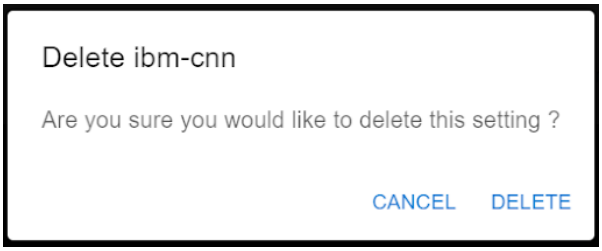


Figure 33.: A screenshot of the delete dialog (source: own work)

Update existing federated learning setting The update operation is a backend-only opera-

tion that can be only used on settings. It is not implemented for the models and datasets. In order to perform it, the user needs to use an API platform as displayed in figure 34 below. The request needs to be a put request. The user needs to enter the correct URL for the request as well as the following key-value pairs as a request body:

- id: the Id of the library that needs to be updated
- key: Which attribute the user wants to change (e.g dataset, model, FL strategy . . .)
- value: The new value of the changed attribute (e.g FedAvg, LSTM . . .)

If the operation is successful the following message will be displayed as a response " id is updated successfully".

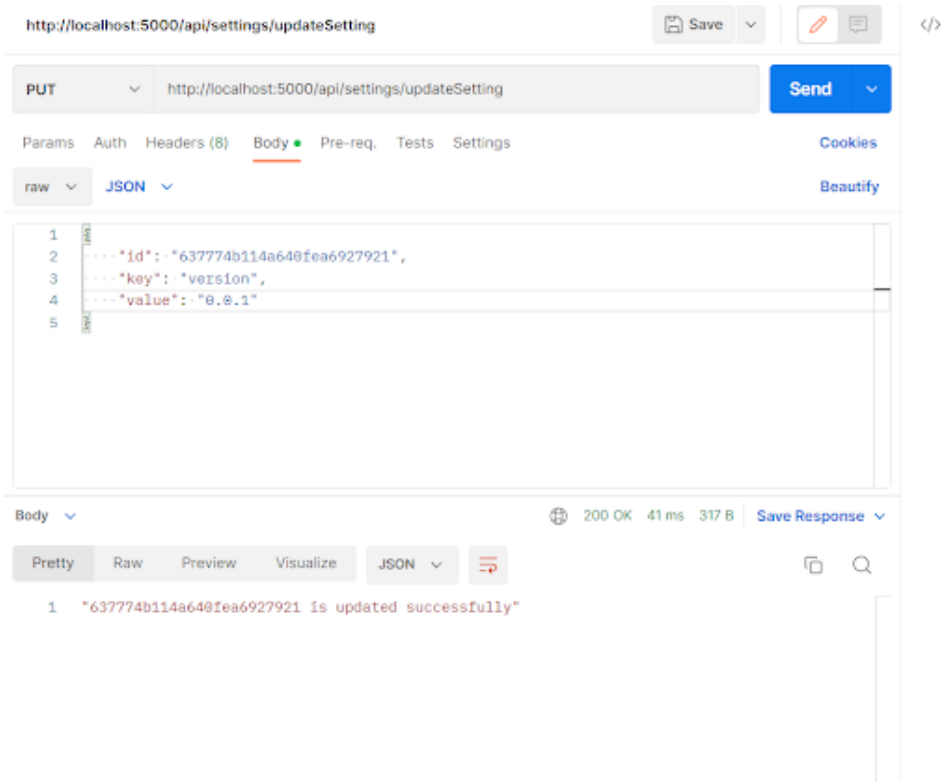


Figure 34.: A screenshot of the update operation (source: own work)

Exception handling when the user performs a wrong operation a pop-up will appear to inform them. The popup has a red border color. Which is distinct from the other green popups. Figure 35 illustrates an example of such a pop-up.

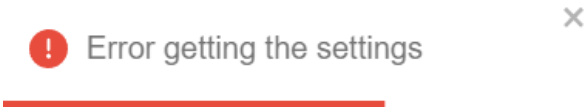


Figure 35.: A screenshot of the error pop-up (source: own work)

5.3.4. Evaluation of the Benchmarking Suite

FMLB is primarily a web application that promises an easy-to-use benchmarking suite for federated learning setups. It measures different non-functional metrics of different federated learning settings, and reports back the findings. The tool is designed to be easy to use, extend, and modify. The only other comparable benchmark currently available in the scientific literature is UniFed [29]. However, FMLB differs from UniFed on multiple dimensions. Table 14 illustrates the key differences between FMLB and UniFed.

FMLB has a graphical user interface delivered through a react web application. It can be used through the UI or the CLI. While Unifed can only be used through the CLI. FMLB has persistent data storage while Unifed does not. FMLB has a client-server architecture and can be easily maintained and extended compared to Unifed. It also implements different metrics that Unifed does not have (it monitors the CPU, GPU, Precision, F1, Recall), and benchmark libraries that Unifed does not benchmark (IBM federated learning and pysyft). It can also benchmark non-federated ML learning models. While Unifed can not. In order to add a new FL setting, the user only needs to fill out a form on the web application of FMLB. While in Unifed this could only be done through the use of docker. Users can also directly make use of the GPU in FMLB without any additional configuration. Which is only possible in Unifed with additional configuration. On the other hand, Unifed is richer in functionalities. It implements more ML models and federated learning strategies, supports more federated learning libraries, and has more datasets included in it. It also implements metrics that do not exist in FMLB (AUC and MSE).

The two benchmark suites differ on other dimensions like the underlying technologies used. FMLB is built using a mix of web technologies (react, node, and MongoDB) and the python programming language. While Unifed is built with only python.

Dimension	**UniFed**	**FMLB**
UI	Command line interface	Web application
Components	Environment launcher, scenario loader, logs analyzer, code patch	A client and a server application
Configurability	Through docker	Locally
GPU	use Through configuration	Can be used directly
ML Models	CNN, RNN, decision tree, LogReg	CNN, LogReg

Persistent data storage	No	Yes
FL Strategies	FedAvg, HistSecAgg, SecureBoost	FedAvg
Datasets	celeba, fedmnist, reddit, breast-horizontal, default-credit-card, give-credit-horizontal, vehicule-scale-horizontal, student-horizontal, breast-vertical, default-credit-vertical, dvisits-vertical, give-credit-vertical, motor-vertical, student-vertical, vehicle-scale-vertical	CIFAR-10, MNIST
Pre-existing libraries	FATE, FedML, PaddleFL, Fedlearner, FederatedScope, TFF, Flower, FLUTE, FedScale, CrypTen, and FedTree	FedML, IBM federated learning, Pysyft, Flower, and Pytorch (non-federated)
Metrics	MSE, Accuracy, AUC, Clients number, training time, communication cost,memory usage.	Precision, Accuracy, Recall, F1, Loss, Client number, data size per client, data points per client, training time, communication cost, memory usage, CPU usage, and GPU usage.

Table 14.: The difference between UniFed [29] and FMLB (source: own work)

Overall it can be concluded that Unifed focuses more on functionality while FMLB focuses more on usability.There are also some key similarities between the two benchmarking suites. Both make use of logs to scrape the metrics. They both use multiple environments for training and benchmarking the different models, and configure the different FL settings with the use of arguments. In order to qualify as an adequate benchmark, FMLB needs to fulfill other criteria. According to Kistowski J.V. et al. [25] to be considered a high-quality benchmark, a benchmark needs to fulfill the following 5 criteria. Namely relevance, reproducibility, fairness, verifiability, and usability. FMLB fulfills all of the mentioned criteria:

- Relevance: The tool measures the most important measurable non-functional metrics for federated learning settings highlighted by the federated learning experts during the interviews.
- Reproducibility: Given the same configuration the tool gives back reproducible results with an extremely small margin of error:
- Fairness: The benchmark implements exactly the same models and training and testing algorithms as well as metrics and settings for all of the libraries involved. No

artificial limitations are added to the benchmark of any of the pre-existing libraries implementations

- Verifiability: The benchmark uses the Scikit-learn metrics [96], as well as the Time [97], PSUtil [98], and GPUtil [89] python packages to measure the performance of the libraries across different federated learning settings. All of the packages used are well-established packages with high ratings, and tens of thousands of users, as well as regular maintenance. The results are accurate and verifiable.
- Usability: The benchmarking environment might require some technical knowledge to set up. But once set up, the benchmarking suite is extremely easy to use. Only one the button click is enough to launch an experiment.

5.4. Federated Learning Libraries' Benchmarking Experiments:

FMLB was used to compare the non-functional quality metrics of the different libraries (e.g Accuracy, Precision, Recall, F1, RAM usage, Network usage, CPU usage, GPU usage, and Time of execution). A series of thirteen experiments were done, the first one was done with PyTorch without the involvement of any client, and twelve experiments were done with the four libraries in three scalability levels (2, 16, and 100 clients).

The experiments included implementations of a CNN with PyTorch and four of the five prominent libraries. TFF was excluded because of the difficulty of the setup. the experiments were conducted on the hardware specifications described in table 6, and they were configured as follows:

- Mode: Simulation
- Model: CNN
- Dataset: MNIST
- Batch size: 100
- Epochs: 10
- Learning rate: 0.01
- Number of communication rounds: 3
- Optimizer: SDG
- Loss function: Cross entropy

Furthermore the GPU was not used in any of the experiments.

5.4.1. PyTorch non-federated results

The Pytorch non-federated implementation was benchmarked for reference. It gave the following results:

- Accuracy: 98.69 %
- Precision: 98.69%
- Recall: 98.67%
- F1: 98.68%
- Loss: 0.0004
- Time of execution: 3 minutes and 46 seconds
- CPU: 52.0%
- GPU: 1.0%
- RAM: 267.91 MB
- Network: 148.35 KB

5.4.2. FL Libraries Benchmark with Two Clients

Table 15 illustrates the results of the experiments conducted with a hundred clients of the four different libraries. The cells of the table were color coded as follows: green for the best-performing library on the row's metric, yellow for the second, orange for the third, red for the worst-performing library, and grey for the metrics where there was no difference between the performances of the libraries.

	Pysyft	Fedml	Flower	IBM FL
Accuracy	97.48%	10.51%	99.03%	99.26%
Precision	97.45%	4.26%*	99.02%	99.25%
Recall	97.48%	1.78%*	99.02%	99.25%
F1	97.46%	2.56%*	99.02%	99.25%
Loss	0.0008	4.8105	0.0003	0.0003
Time of execution	22m 19s 890ms	1m 13s 670ms	20m 23s 600ms	37m 21s 50ms
CPU consumption	91.8%	34.5%	33.9%	99.9%
RAM consumption	604.01MB	620.96MB	1GB 232.97MB	856.94MB
Network consumption	136.21MB	25.64MB	465.14MB	1025,68MB
GPU consumption	1.0%	1.0%	1.0%	1.0%

Table 15.: The results for the training experiments of federated learning settings with two clients with different libraries (source: own work)

Data analysis: Table 15 illustrates the results of the experiments conducted with two clients of the four different libraries. The GPU consumption is 1% for all of the experiments because the models don't make use of the GPU when they are training. FedML took the least time and resources to train (1m 13s to run, 34.5% of the CPU, 25.64MB of the network, 620MB of the RAM) the CNN but it also had the lowest accuracy of all of the libraries (around 10%). The results of the precision, recall, and F1 are not consistent with accuracy for FedML. They are quite low which suggests that the figures may not be true, and they can be ignored. All of Flower, IBM federated learning, and Pysyft had high accuracy, precision, recall, and F1 metrics (all above 97%). Flower and IBM both scored above 99% across these metrics. IBM federated learning and Pysyft were CPU intensive in comparison to Flower (Both consumed three times more CPU computational resources than Flower). While for network consumption, IBM FL consumed the most, followed by flower (60% lower than IBM FL), and Pysyft (87% lower than IBM). For the RAM, Pysyft consumed the least (around 600MB), followed by Flower (around 850MB). IBM FL consumed the most RAM (1.2GB). In terms of the time of execution, both Pysyft and Flower took around 20 minutes to train the model. IBM federated learning took almost double the time to execute (around 37 minutes). For two clients, FedML consumes the least resources, and takes the least time to execute, but has low accuracy in comparison to the other libraries. The other libraries have comparable accuracy metrics to each other. However, they differ in terms of resource consumption and time of execution. IBM FL consumes the most resources and takes the most time to train the model. Pysyft and Flower have a comparable execution time. However, Flower is more RAM and Network intensive than PySyft while Pysyft is more CPU-intensive than Flower.

5.4.3. FL Libraries Benchmark with Sixteen Clients

Table 16 illustrates the results of the experiments conducted with a hundred clients of the four different libraries. The cells of the table were color coded as follows: green for the best-performing library on the row's metric, yellow for the second, orange for the third, red for the worst-performing library, and grey for the metrics where there was no difference between the performances of the libraries.

	Pysyft	**Fedml**	**Flower**	**IBM FL**
Accuracy	97.23%	27.43%	99.31%	99.0%
Precision	97.22%	4.26%*	99.33%	99.98%
Recall	97.20%	2.42%*	99.24%	99.99%
F1	97.20%	3.09%*	99.29%	99.99%
Loss	0.0009	1.95	0.0006	0.0003
Time of execution	22mins 23s	1mins 43s	1 hours 27mins 10s	4 hours 18 mins 32s
CPU consumption	99.0%	40.0%	99.3%	99.9%
RAM consumption	858.48MB	670.17MB	4GB 321.37MB	4GB 410.64MB
Network consumption	592.3MB	105.23MB	974,94MB	2332.45MB
GPU consumption	1.0%	1.0%	1.0%	1.0%

Table 16.: The results for the training experiments of federated learning settings with sixteen clients with the different libraries (source: own work)

Data analysis: Like the previous experiment the GPU consumption is 1% for all of the experiments. FedML had an increased accuracy but it is still a lot less than other libraries (27.43%). It also took more time to train the model, and consumed more resources than in the previous experiment but still less than the other libraries (took 1m 43s to run, 40% of the CPU, 105MB of the network, 670MB of the RAM). The precision, recall, and F1 are still not consistent with accuracy for FedML. All of Flower, IBM federated learning, and Pysyft had high accuracy, precision, recall, and F1 metrics. Pysyft scored again 2% lower than Flower and IBM FL across these metrics. It also took fewer resources and time to execute the training of the models. It consumed 15% of the RAM of the two other libraries. It also consumed half of the network bandwidth consumed by Flower, and one-fourth of the network consumed by IBM FL. Additionally, It took around 80% less time than flower and around 92.5% less time than IBM FL to train the model. All of Pysyft, IBM FL, and Flower consumed almost the entirety of the CPU computational power. For two sixteen, FedML consumes the least resources, and takes the least time to execute again, but still has low accuracy in comparison to the other libraries. The other libraries have comparable accuracy metrics and CPU consumption. However, PySyft consumes way less RAM, Network, and takes less time

to execute than IBM and Flower. Flower consumes fewer resources and takes 75% less time to train the model than IBM federated learning.

5.4.4. FL Libraries Benchmark with a Hundred Clients

Table 17 illustrates the results of the experiments conducted with a hundred clients of the four different libraries. The cells of the table were color coded as follows: green for the best-performing library on the row's metric, yellow for the second, orange for the third, red for the worst-performing library, and grey for the metrics where there was no difference between the performances of the libraries.

	Pysyft	**Fedml**	**Flower**	**IBM FL**
Accuracy	96.82%	80.35%	99.03%	99.22%
Precision	96.8%	4.11%*	99.98%	99.11%
Recall	96.82%	4.08%*	99.98%	99.11%
F1	96.81%	4.09%*	99.98%	99.11%
Loss	0.0009	0.6249	0.0002	0.0003
Time of execution	23m 46s 23s	4mins 40s	3hours 47mins 3s	25hours 3mins 45s
CPU consumption	97.8%	60.7%	99.4%	99.9%
RAM consumption	861.1MB	749.43MB	25.68GB	24.37GB
Network consumption	748.86MB	543.04MB	3345.85MB	3543.04 MB
GPU consumption	1.0%	1.0%	1.0%	1.0%

Table 17.: The results for the training experiments of federated learning settings with a hundred clients with different libraries (source: own work)

Data analysis: The GPU consumption is always around 1% for all the libraries. FedML had an 80% accuracy but still considerably less than the other libraries. It also still consumes fewer resources and takes less time to execute than the other libraries. All of Flower, IBM federated learning, and Pysyft had high accuracy, precision, recall, and F1 metrics. Pysyft scored again 3% lower than Flower and IBM FL across these metrics. It also took fewer resources and time to execute the training of the models. Both IBM FL and Flower consumed around 25GB of RAM, while Pysyft only consumed 861MB. They both also consumed around 3.5GB of Network bandwidth, while PySyft only consumed 750MB. They all consumed comparable CPU computational power. For a hundred clients, FedML consumes the least resources, and takes the least time to execute (took 4m 40s to run, 60.7% of the CPU, 105MB of the network, 749.43MB of the RAM), but it also has low accuracy in comparison to the others (around 80%). The other libraries have comparable accuracy metrics. However, PySyft consumes fewer resources and takes less time to execute than IBM FL and Flower. Flower consumes less Network and CPU than IBM FL, while IBM FL consumes less RAM than Flower. IBM FL

takes the most time to train the model (around 25 hours), followed by Flower (around 3.75 hours), followed by PySyft (around 23 minutes), and FedML (around 4 minutes).

5.4.5. Insights from the Data Analysis

The prominent libraries included in this thesis are Fedml, Pysyft, IBM FL, TFF, and Flower. The experiments included all of the prominent libraries except for TFF. There was an additional experiment run with PyTorch to be used as a reference for the other experiments. TFF was not included in the benchmark because its installation was not successful, as the installation process in the documentation was meticulously followed but was full of errors. There was no further support for the installation other than in its official documentation.

The GPU consumption is a constant 1% across all experiments for all the libraries. The different libraries behave differently across different scalability levels. For instance, FedML improves its accuracy drastically as it scales, while PySyft loses accuracy as it scales. The accuracy of IBM FL and Flower stays constant. In terms of resources, FedML consumes the least resources and takes the least time to train the model but the resource consumption and the execution time increases with scale. The precision, recall, and F1 of FedML are not consistent with its accuracy. Thus, they were not taken into account.

IBM FL, PySyft, and Flower consume around 97% to 99.9% of CPU computational power and have 97% to 99% accuracy across the different scalability levels. IBM FL consumes the most resources and takes the most time, followed by Flower. PySyft takes the least time to execute and consumes the least resources. Scalability does not have a big impact on the training time and resource consumption of PySyft but it plays a bigger role for Flower and IBM FL. IBM FL's training time increases drastically with scale compared to Flower but in terms of resources, both libraries consume the same resources as they scale. Compared to the PyTorch experiment, all of the libraries except for FedML take more time to train their ML models and consume more resources. IBMFL, Flower, and PySyft also have a higher accuracy than the PyTorch implementation.

Overall, it can be concluded that each of the libraries is suitable for different use cases. FedML and Pysyft are more useful to have quick results in a simulation with a high number of clients. Pysyft gives more realistic results than FedML. However, both libraries are not production ready yet as their simulation mode does not mimic the actual behavior of client-server systems. IBM FL and Flower have high accuracy and more realistic results but they come with more training time and resource consumption. They mimic more the behavior of an actual FL setting. They are more production ready.

6. Discussion

The results of this thesis have both theoretical and practical implications for both academia and the industry but they also have certain limitations. Thus, a reflection on the thesis and its results is needed.

6.1. Reflection

The thesis aimed to answer three research questions. Namely:

1. What are the functional and non-functional requirements relevant for a federated learning library, and what are the most important metrics to benchmark them?
2. What are the different federated libraries available, and how do they differ in terms of functionality?
3. How could a modular software application that benchmarks the different federated learning libraries using the metrics be developed?

A series of semi-structured interviews with federated learning experts to identify what is important for the federated learning community regarding features and quality requirements for the different libraries was conducted. After that, the scientific literature around federated learning systems as well as the official documentation of the libraries used for its development was explored. A list of the currently available FL libraries and their status and description was compiled. Their functionalities and features were also compared. Additionally, a benchmarking suite that includes two datasets, five different reference implementations of federated learning and machine learning models, a web application to configure and test federated learning setups, as well as, different quality metrics were developed. The benchmarking suite was named FMLB. In the end, FMLB was used to compare the quality of the different FL libraries across multiple dimensions.

6.2. Theoretical Implications

The results for the first research question included a series of interviews that reflected the preferences of the FL community. It showed that the FL community prefer the libraries to support basic FL functionalities like handling server-client communication, security, data aggregation, and supporting multiple ML framework. The community also values scalability and accuracy over performance and efficiency. The results could be used as secondary data for further research about the preferences of the FL community. The list of questions

could also be used as a basis for future studies. The answer to the second and the third research questions explored the different federated learning libraries and the functional and non-functional differences between them. The conclusion was that some libraries support more functionalities than others. For instance, IBM supports all prominent ML models, while TFF only supports DL models. Also, the libraries differ in terms of non-functional quality. FedML is the fastest and least resources intensive among the libraries but that comes with an accuracy tradeoff. IBM FL is the slowest and most resources intensive but overall it is production ready and have a high accuracy. These results could be used by researchers to choose the appropriate FL library for their specific use case. Lastly, the researcher could use the benchmarking suite to further conduct experiments on different federated learning libraries.

6.3. Practical Implications

The results of the interviews could be used by the library designers and maintainers to know what is important for the FL practitioners and community and design their libraries accordingly. The answer to the second and the third research questions could be used by FL practitioners to compare the different libraries and choose one that is appropriate for their use case. Even though the libraries are still in active development and the results may be irrelevant in the future, the methodology for the comparison can be used as a guideline for future comparisons. The tool can be used to compare newer versions of the libraries to get more up-to-date information about the libraries in the future.

6.4. Limitations and Future Works

The sample size for the interviews in the study was small. It can be used to get an overview of the preferences of the FL community but it is not an accurate reflection of the preferences. The sample was also not checked for potential biases as all of the interviewees worked for Munich-based companies and came from a research background. Another study that has a bigger sample size and more specific questions could give a clearer idea to FL library designers about the direction they want to steer their products. The answers to the second and third research questions could be irrelevant in the future since the libraries are still under active development. Most of the libraries are not even production ready, and the field is still in its infancy. Another comparison could be carried out once each of the libraries is production ready and have at least a stable release. The benchmark suite is limited to the most popular ML models and only FedAvg. It has a limited number of datasets and implementations and does not include all of the FL libraries, only the prominent ones. New libraries will emerge, and more use cases for FL will also emerge that will require other data aggregation algorithms and machine learning models. The benchmark is also not stable yet. It was not tested enough and was not used outside the scope of this thesis. It could be extended to include more options and to be more customizable, and it should be rigorously tested before being used in production.

7. Conclusion

The expert interviews revealed that the FL community expects the FL library to cover nothing beyond the basic FL features like communication and the orchestration between the server and the client, security, and data aggregation. They also expect the FL libraries to be highly interoperable with the different ML frameworks. All of the interviewees expected the different libraries to have comparable accuracy metrics, which was disproved by the experiments.

In terms of features, all of the prominent libraries cover basic features like data aggregation with FedAvg. They mostly all have an encryption-based security mechanism (except for TFF which only uses DP). FedML has the most features suited for experimental settings (like network and topology customization, as well as ready already implemented ML models). While IBM FL covers the most of ML models and simulates better the client-server interaction. Pysyft and TFF only support deep learning models and have the least features in all of the reviewed libraries. Flower supports deep learning models and some of the traditional machine learning models. It also has high interoperability with different ML frameworks like sci-kit learn, Pytorch, and TF. In terms of features, flower and IBM FL are the most suited for a production setting. While FedML is more suited for an experimental setting. PySyft and TFF are the least production-ready because of the lack of features and support despite having the biggest communities of practitioners.

The experiments to benchmark the different non-functional quality metrics of the libraries were conducted with FMLB, a web application developed as part of the thesis. They gave an insight about the non-functional differences between the different libraries. In terms of non-functional quality, PySyft and FedML have the best resource consumption and time of execution, especially for settings with a high number of clients. FedML, however, has a relatively low accuracy compared to PySyft. Flower and IBM FL consume the most resources and take the most time to train but also have higher accuracy. TFF was not part of the benchmarking experiment as It was not possible to install it. However, according to the literature [30], TFF has lower accuracy and a shorter training time than PySyft.

Each of the prominent libraries has its advantages and disadvantages. Flower and IBM FL are the most production-ready as they support many ML frameworks and models, and they have client-server architecture embedded in their implementation. However, they both come with a higher time of execution and resource consumption. Flower is more suited for cross-client setups as it consumes fewer resources overall and scales more easily. While IBM FL is more suited for cross-silo setups since it has more features than Flower.

TFF, FedML, and Pysyft are all more suited for experimental setups. FedML is more useful for quick prototyping. While PySyft and TFF are more suitable for higher-fidelity simulations. The choice between TFF and PySyft depends on the preferred ML framework of the FL practitioner. TFF is suited for practitioners who use TF and PySyft is suited for practitioners who use PyTorch. More experiments need to be conducted in real-life production setups with the different libraries as they evolve to further define the use cases suitable for the different libraries.

A. Addenda

A federated learning libraries benchmark called FMLB was developed as part of the thesis. It can be found under: https://github.com/sdn98/BFML .

The different machine learning algorithms were implemented with the help of the subsequent tutorials:

- for PyTorch: https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118
- for PySyft: https://colab.research.google.com/drive/1dRG3yNAlDar3tll4VOkmoU-aLslhUS8d
- for FedML: https://github.com/FedML-AI/FedML
- for IBM FL: https://github.com/IBM/federated-learning-lib/tree/main/examples/fedavg
- for Flower: https://flower.dev/docs/quickstart-pytorch.html

List of Figures

List of Tables

Acronyms

TF: Tensorflow
TFF: Tensorflow Federated
FATE: Federated AI Technology Enabler
FLUTE: Federated Learning Utilities for Testing and Experimentations
DP: Differential Privacy
PFL: PaddleFL
FL: Federated Learning
ML: Machine Learning
IoT: Internet of Things
DNN: Deep Neural Networks
FNN: Feedforward Neural Networks
RNN: Recurrent Neural Networks
CNN: Convolutional Neural Networks
FTL: Federated Transfer Learning
IID: Independent and Identically Distributed
FR: Functional Requirements
NFR: Non-functional Requirements
LSTM: Long Short-term Memory
BERT: Bidirectional Encoder Representations from Transformers
GRU: Gated Recurrent Unit
ROC: Receiver Operating Characteristic
AUC: Area Under the (ROC) Curve
MSE: Mean Square Error
MAE: Mean Absolute Error
RMSE: Root Mean Square Error
NLP: Natural Language Processing
API: Application Programing Interface
CLI : Command Line Interface
PoC: Proof of Concept
GPU: Graphics Processing Unit
CPU: Central Processing Unit
RAM: Random Access Memory
OS: Operating System
AI: Artificial Intelligence
UI: User Interface
FMLB: Federated Machine Learning Benchmark

SDG: Stochastic Gradient Descent
JSON: JavaScript Object Notation
YAML : YAML ain't markup language (a recursive acronym)
CRUD: Create Read Update Delete
gRPC: General-purpose Remote Procedure Call
SVM: Support Vector Machine
RSA: Rivest–Shamir–Adleman
HE: Homomorphic Encryption
SDK: Software Development Kit
DQN: Deep Q-Network
DDPG: Deep Deterministic Policy Gradient
PCA: Principal Component Analysis
UMAP: Uniform Manifold Approximation and Projection
KNN: K-nearest Neighbors
P2P: Peer to Peer
FURPS: Functionality Usability Reliability Performance Supportability
ISO: International Standards Organisation
IDE: ID-discriminative embedding
IT: Information Technology
DSR: Design Science Research
DSRM: Design Science Research Methodology
R-Factor: Relevance Factor
RQ: Research Question
GDPR: General Data Protection Regulation
MPC: Multi-party Computation
MPI: Message Passing Interface
gRPC: general Remote Procedure Call
SecAgg: Secure Aggregation
SK-learn: SciKit Learn

Bibliography

[1] Nilsson, N. J. (2012). A Biographical Memoirs. National Academy of Sciences.

[2] Badillo, S., Banfai, B., Birzele, F., Davydov, I. I., Hutchinson, L., Kam-Thong, T., ... / Zhang, J. D. (2020). An introduction to machine learning. Clinical pharmacology / therapeutics, 107(4), 871-885.

[3] Nilsson, N. J. (1996). Introduction to machine learning. An early draft of a proposed textbook.

[4] Lewis-Beck, C., / Lewis-Beck, M. (2015). Applied regression: An introduction (Vol. 22). Sage publications..

[5] Hastie, T., Tibshirani, R., Friedman, J. H., / Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction (Vol. 2, pp. 1-758). New York: springer.

[6] Soofi, A. A., / Awan, A. (2017). Classification techniques in machine learning: applications and issues. Journal of Basic Applied Sciences, 13, 459-465.

[7] Zhou, Z. H. (2021). Machine learning. Springer Nature.

[8] Kotsiantis, S. B., Zaharakis, I. D., / Pintelas, P. E. (2006). Machine learning: a review of classification and combining techniques. Artificial Intelligence Review, 26(3), 159-190.

[9] Krose, B., / Smagt, P. V. D. (2011). An introduction to neural networks.

[10] O'Shea, K., / Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

[11] Shaheen, M., Farooq, M. S., Umer, T., / Kim, B. S. (2022). Applications of federated learning; Taxonomy, challenges, and research trends. Electronics, 11(4), 670.

[12] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., ... / Zhao, S. (2021). Advances and open problems in federated learning. Foundations and Trends® in Machine Learning, 14(1–2), 1-210.

[13] Yang, Q., Liu, Y., Cheng, Y., Kang, Y., Chen, T., / Yu, H. (2020). Federated Transfer Learning. In Federated Learning (pp. 83-93). Springer, Cham.

[14] Lalitha, A., Shekhar, S., Javidi, T., / Koushanfar, F. (2018, December). Fully decentralized federated learning. In Third workshop on Bayesian Deep Learning (NeurIPS).

[15] Śmietanka, M., Pithadia, H., / Treleaven, P. (2020). Federated Learning for Privacy-Preserving Data Access. Available at SSRN 3696609.

[16] Scopus advanced search, https://www.scopus.com/search/form.uri?display=advanced, last accessed 20.07.2022.

[17] Li, T., Sahu, A. K., Talwalkar, A., / Smith, V. (2020). Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine, 37(3), 50-60.

[18] Li, L., Fan, Y., Tse, M., / Lin, K. Y. (2020). A review of applications in federated learning. Computers / Industrial Engineering, 149, 106854.

[19] Borzymowski, M. (2019) : Federated Learning , the blue.ai blog post, https://theblue.ai/blog/federated-learning/

[20] Mills, J., Hu, J., / Min, G. (2020). Faster Federated Learning with Decaying Number of Local SGD Steps.

[21] Dwork, C. (2008, April). Differential privacy: A survey of results. In International conference on theory and applications of models of computation (pp. 1-19). Springer, Berlin, Heidelberg.

[22] Wiegers, K., Beatty, J. (2013). Software requirements. Pearson Education.

[23] Habibullah, K. M., Horkoff, J. (2021, September). Non-functional requirements for machine learning: understanding current use and challenges in industry. In 2021 IEEE 29th International Requirements Engineering Conference (RE) (pp. 13-23). IEEE.

[24] Miguel, J. P., Mauricio, D., Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. arXiv preprint arXiv:1412.2977.

[25] v. Kistowski, J., Arnold, J. A., Huppler, K., Lange, K. D., Henning, J. L., Cao, P. (2015, January). How to build a benchmark. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (pp. 333-336).

[26] Roy, S. (2022): a baeldung article, https://www.baeldung.com/cs/framework-vs-library

[27] Meyer, B. (2001). Software engineering in the academy. Computer, 34(5), 28-35.

[28] Bouraqqadi, H., Berrag, A., Mhaouach, M., Bouhoute, A., Fardousse, K., Berrada, I. (2021). PyFed: extending PySyft with N-IID Federated Learning Benchmark. In The 34th Canadian Conference on Artificial Intelligence.

[29] Liu, X., Shi, T., Xie, C., Li, Q., Hu, K., Kim, H., ... Song, D. (2022). UniFed: A Benchmark for Federated Learning Frameworks. arXiv preprint arXiv:2207.10308.

[30] Kholod, I., Yanaki, E., Fomichev, D., Shalugin, E., Novikova, E., Filippov, E., Nordlund, M. (2020). Open-source federated learning frameworks for IoT: A comparative review and analysis. Sensors, 21(1), 167.

[31] Wei, W., Liu, L., Loper, M., Chow, K. H., Gursoy, M. E., Truex, S., Wu, Y. (2020). A framework for evaluating gradient leakage attacks in federated learning. arXiv preprint arXiv:2004.10397.

[32] Zhuang, W., Wen, Y., Zhang, X., Gan, X., Yin, D., Zhou, D., ... Yi, S. (2020, October). Performance optimization of federated person re-identification via benchmark analysis. In Proceedings of the 28th ACM International Conference on Multimedia (pp. 955-963).

[33] Nilsson, A., Smith, S., Ulm, G., Gustavsson, E., Jirstrand, M. (2018, December). A performance evaluation of federated learning algorithms. In Proceedings of the second workshop on distributed infrastructures for deep learning (pp. 1-8).

[34] Basu, P., Roy, T. S., Naidu, R., Muftuoglu, Z., Singh, S., Mireshghallah, F. (2021). Benchmarking differential privacy and federated learning for bert models. arXiv preprint arXiv:2106.13973.

[35] Hao, T., Huang, Y., Wen, X., Gao, W., Zhang, F., Zheng, C., ... Zhan, J. (2018, December). Edge AIBench: towards comprehensive end-to-end edge computing benchmarking. In International Symposium on Benchmarking, Measuring and Optimization (pp. 23-30). Springer, Cham.

[36] He, C., Balasubramanian, K., Ceyani, E., Yang, C., Xie, H., Sun, L., ... Avestimehr, S. (2021). Fedgraphnn: A federated learning system and benchmark for graph neural networks. arXiv preprint arXiv:2104.07145.

[37] Gao, Y., Kim, M., Abuadbba, S., Kim, Y., Thapa, C., Kim, K., ... Nepal, S. (2020). End-to-end evaluation of federated learning and split learning for internet of things. arXiv preprint arXiv:2003.13376.

[38] Zhang, Z., Yang, Y., Yao, Z., Yan, Y., Gonzalez, J. E., Ramchandran, K., Mahoney, M. W. (2021, December). Improving semi-supervised federated learning by reducing the gradient diversity of models. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 1214-1225). IEEE.

[39] Wu, S., Li, T., Charles, Z., Xiao, Y., Liu, Z., Xu, Z., Smith, V. (2022). Motley: Benchmarking Heterogeneity and Personalization in Federated Learning. arXiv preprint arXiv:2206.09262.

[40] Chen, D., Gao, D., Kuang, W., Li, Y., Ding, B. (2022). pFL-Bench: A Comprehensive Benchmark for Personalized Federated Learning. arXiv preprint arXiv:2206.03655.

[41] Yao, L., Gao, D., Wang, Z., Xie, Y., Kuang, W., Chen, D., ... Li, Y. (2022). Federated Hetero-Task Learning. arXiv preprint arXiv:2206.03436.

[42] Liu, L., Zhang, F., Xiao, J., Wu, C. (2020). Evaluation framework for large-scale federated learning. arXiv preprint arXiv:2003.01575.

[43] Lai, F., Dai, Y., Zhu, X., Madhyastha, H. V., Chowdhury, M. (2021, October). FedScale: Benchmarking model and system performance of federated learning. In Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning (pp. 1-3).

[44] Liang, Y., Guo, Y., Gong, Y., Luo, C., Zhan, J., Huang, Y. (2020, October). Flbench: A benchmark suite for federated learning. In BenchCouncil International Federated Intelligent Computing and Block Chain Conferences (pp. 166-176). Springer, Singapore.

[45] Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konečný, J., McMahan, H. B., ... Talwalkar, A. (2018). Leaf: A benchmark for federated settings. arXiv preprint arXiv:1812.01097.

[46] He, C., Li, S., So, J., Zeng, X., Zhang, M., Wang, H., ... Avestimehr, S. (2020). Fedml: A research library and benchmark for federated machine learning. arXiv preprint arXiv:2007.13518.

[47] Hu, S., Li, Y., Liu, X., Li, Q., Wu, Z., He, B. (2022). The oarf benchmark suite: Characterization and implications for federated learning systems. ACM Transactions on Intelligent Systems and Technology (TIST), 13(4), 1-32.

[48] Chai, D., Wang, L., Chen, K., Yang, Q. (2020). Fedeval: A benchmark system with a comprehensive evaluation model for federated learning. arXiv preprint arXiv:2011.09655.

[49] Ziller, A., Trask, A., Lopardo, A., Szymkow, B., Wagner, B., Bluemke, E., ... Kaissis, G. (2021). Pysyft: A library for easy federated learning. In Federated Learning Systems (pp. 111-139). Springer, Cham.

[50] PySyft GitHub page, https://github.com/OpenMined/PySyft , last accessed 02.09.2022.

[51] PySyft documentation page, https://openmined.github.io/PySyft/index.html , last accessed 02.09.2022.

[52] Tensorflow Federated GitHub page, https://github.com/tensorflow/federated , last accessed 03.09.2022.

[53] Tensorflow Federated documentation page, https://www.tensorflow.org/federated , last accessed 03.09.2022.

[54] Tensorflow Federated Analytics documentation page, https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html , last accessed 03.09.2022.

[55] FATE website front page, https://fate.fedai.org/, last accessed 05.09.2022.

[56] FATE GitHub page, https://github.com/FederatedAI/FATE, last accessed 05.09.2022.

[57] FATE documentation page, https://fate.readthedocs.io, last accessed 05.09.2022.

[58] Liu, Y., Fan, T., Chen, T., Xu, Q., Yang, Q. (2021). FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection. J. Mach. Learn. Res., 22(226), 1-6.

[59] FedML GitHub page, https://github.com/FedML-AI/FedML , last accessed 07.09.2022.

[60] FedML website frontpage, https://fedml.ai/ , last accessed 07.09.2022.

[61] Flower website frontpage, https://flower.dev/ , last accessed 10.09.2022.

[62] Flower GitHub page, https://github.com/adap/flower , last accessed 10.09.2022.

[63] Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., ... Lane, N. D. (2022). Flower: A friendly federated learning framework.

[64] FedLearner GitHub page, https://github.com/bytedance/fedlearner , last accessed 12.09.2022.

[65] Ludwig, H., Baracaldo, N., Thomas, G., Zhou, Y., Anwar, A., Rajamoni, S., ... Abay, A. (2020). Ibm federated learning: an enterprise framework white paper v0. 1. arXiv preprint arXiv:2007.10987.

[66] EasyFL GitHub page, https://github.com/WwZzz/easyFL, last accessed 13.09.2022.

[67] FLUTE GitHub page, https://github.com/microsoft/msrflute, last accessed 13.09.2022.

[68] OpenFL GitHub page, https://github.com/openfl/openfl, last accessed 13.09.2022.

[69] FedTree GitHub page, https://github.com/Xtra-Computing/FedTree, last accessed 13.09.2022.

[70] PaddleFL documentation page, https://paddlefl.readthedocs.io/en/latest/introduction.html, last accessed 15.09.2022.

[71] PaddleFL GitHub page,https://github.com/PaddlePaddle/PaddleFL, last accessed 15.09.2022.

[72] Voigt, P., Von dem Bussche, A. (2017). The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing, 10(3152676), 10-5555.

[73] Müller, T., Gaertner, N., Verzano, N., Matthes, F. (2022). Barriers to the Practical Adoption of Federated Machine Learning in Cross-company Collaborations. In ICAART (3) (pp. 581-588).

[74] Ramdhani, A., Ramdhani, M. A., Amin, A. S. (2014). Writing a Literature Review Research Paper: A step-by-step approach. International Journal of Basic and Applied Science, 3(1), 47-56.

[75] Bowen, G. A. (2009). Document analysis as a qualitative research method. Qualitative research journal.

[76] Newcomer, K. E., Hatry, H. P., Wholey, J. S. (2015). Conducting semi-structured interviews. Handbook of practical program evaluation, 492, 492.

[77] Guest, G., MacQueen, K. M., Namey, E. E. (2011). Applied thematic analysis. sage publications.

[78] Peffers, K., Tuunanen, T., Rothenberger, M. A., Chatterjee, S. (2007). A design science research methodology for information systems research. Journal of management information systems, 24(3), 45-77.

[79] React Home page, https://reactjs.org/ , last accessed 23.11.2022.

[80] Zustand NPM page, https://www.npmjs.com/package/zustand , last accessed 23.11.2022.

[81] AXIOS Home page, https://axios-http.com/docs/intro , last accessed 23.11.2022.

[82] Node Home page, https://nodejs.org/en/ , last accessed 23.11.2022.

[83] MongoDB Home page, https://www.mongodb.com/, last accessed 23.11.2022.

[84] FMLB Github page, https://github.com/sdn98/FMLB, last accessed 06.01.2023.

[85] MUI home page, https://mui.com/, last accessed 23.11.2022.

[86] MNIST home page, http://yann.lecun.com/exdb/mnist/, last accessed 23.11.2022.

[87] CIFAR-10 home page, https://www.cs.toronto.edu/ kriz/cifar.html, last accessed 23.11.2022.

[88] Benchmark logo, https://www.shareicon.net/chart-graph-seo-benchmark-8627362. , last accessed 23.11.2022.

[89] GPUtil pip page, https://pypi.org/project/GPUtil/, last accessed 23.11.2022.

[90] Conda home page, https://docs.conda.io/en/latest/, last accessed 23.11.2022.

[91] Postman home page, https://www.postman.com/, last accessed 23.11.2022.

[92] Python home page, https://www.python.org/, last accessed 23.11.2022.

[93] Pip home page, https://pypi.org/project/pip/, last accessed 23.11.2022.

[94] NPM home page, https://www.npmjs.com/, last accessed 23.11.2022.

[95] Pytorch home page, https://pytorch.org/, last accessed 23.11.2022.

[96] Scikit-learn pip page, https://pypi.org/project/scikit-learn/, last accessed 23.11.2022.

[97] Times pip page, https://pypi.org/project/times/, last accessed 23.11.2022.

[98] PSUtil pip page, https://pypi.org/project/psutil/, last accessed 23.11.2022.

[99] MUI icons, https://icons8.com/icons/set/material-ui, last accessed 23.11.2022.

[100] IBM Federated Learning GitHub page, https://github.com/IBM/federated-learning-lib, last accessed 03.01.2023.