IBM Federated Learning: an Enterprise Platform White Paper V0.1

Heiko Ludwig[*], Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, Mark Purcell, Ambrish Rawat, Tran Minh, Naoise Holohan, Supriyo Chakraborty, Shalisha Whitherspoon, Dean Steuer, Laura Wynter, Hifaz Hassan, Sean Laguna, Mikhail Yurochkin, Mayank Agarwal, Ebube Chuba, Annie Abay

IBM Research

Abstract

Federated Learning (FL) is an approach to conduct machine learning without centralizing training data in a single place, for reasons of privacy, confidentiality or data volume. However, solving federated machine learning problems raises issues above and beyond those of centralized machine learning. These issues include setting up communication infrastructure between parties, coordinating the learning process, integrating party results, understanding the characteristics of the training data sets of different participating parties, handling data heterogeneity, and operating with the absence of a verification data set.

IBM Federated Learning provides infrastructure and coordination for federated learning. Data scientists can design and run federated learning jobs based on existing, centralized machine learning models and can provide high-level instructions on how to run the federation. The framework applies to both Deep Neural Networks as well as “traditional” approaches for the most common machine learning libraries. *IBM Federated Learning* enables data scientists to expand their scope from centralized to federated machine learning, minimizing the learning curve at the outset while also providing the flexibility to deploy to different compute environments and design custom fusion algorithms.

1 Introduction

Federated Learning (FL) is an approach to apply machine learning to situations in which data cannot be centralized for a training process. The success of using machine learning to solve a problem depends, to a large extent, on the quality and quantity of available training data. Machine learning approaches typically rely on the central management of training data, even if the training process itself is run on a clustered infrastructure. The process can consider the properties of the total training data set and the availability of representative validation data sets, e.g., for hyperparameter tuning. However, centralizing data for training is often not feasible or practical, for reasons of data privacy, secrecy, regulatory compliance, and the sheer volume of data involved.

Enterprises have their applications deployed in a variety of data centers and clouds. In such a multi-cloud scenario, moving data into one location can be impractical or expensive, particularly if

[*]Corresponding authors: hludwig@us.ibm.com, baracald@us.ibm.com, gegi@us.ibm.com

the data are generated at a high velocity or the amount of data is large. A similar situation holds in edge computing scenarios in which, say, data in telecommunication switches is used for training.

In recent years, privacy regulations limit the use of personal data, make central data repositories expensive to manage, and represent a liability to the company storing the data. Regulations such as the European Union's General Data Protection Regulation (GDPR) [26], the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [6], and others require data holders to keep information private and limit how data can be used. Private and confidential data have been compromised from different systems in the past years, with notable security incidents including [30, 23, 15, 1]. These data breaches result in costly mitigation, fines, and reputation damage [23].

Many privacy regulations also contain limitations on the location where data can be stored. Often, they stipulate that data cannot be taken out of the country to which the regulation applies or at least not to countries that have weaker privacy protections. This even applies to data of the same company if it has, say, customer databases in different countries, and presents a serious obstacle to effective use of data in a training process.

Lastly, in some scenarios, different enterprises cooperate in training a machine learning model, mostly for non-competitive reasons. Financial institutions collaborate to combat money laundering and fraud - as required by regulation - and medical research facilities work jointly on improving diagnostics and treatment design [29]. However, due to regulation and secrecy, financial firms will not share transaction data and medical institutions cannot share patient data.

While practical and regulatory reasons limit the centralization of training data, using these different sources of data is still important in many cases to build models on large and representative training datasets.

Federated learning (FL) [19] is an approach to train machine learning models that do not require sharing datasets with a central entity. In federated learning, a model is trained collaboratively among multiple parties, which keep their training dataset to themselves but participate in a shared federated learning process. The notion of parties might refer to entities as different as data centers of an enterprise in different countries, compute clusters in different clouds, cell phones, cars, or different companies and organizations.

The learning process between parties most commonly uses a single *aggregator*, while peer-to-peer or decentralized models have also been proposed [27]. An aggregator would coordinate the overall process, communicate with the parties, and integrate the results of the training process. Most typically, in particular for neural networks, parties would run a local training process on their training data, share the weights of their model with the aggregator, which would then aggregate the weight vectors of all parties using a *fusion algorithm*. Then, the merged model is sent back to all parties for the next *round* of training.

Classical machine learning models can also be adapted to a federated environment. We have to decide which parts of the *federated learning algorithm* would run locally in parties and which would run in an aggregator. A federated decision tree algorithm might grow the tree in the aggregator, and query the parties for the count information based on the parties' data sets. Based on these counts, it would compute the information gains for all possible splits and select the feature with the largest information gain to split and built the tree node, and then iterate the next round. There are many choices on how to design federated machine learning algorithms, and this is still a very active field of research at the point of writing this paper.

FL stands in contrast to established distributed training systems [21, 17], where all data is transmitted to a central data center and is subsequently distributed among cluster nodes to train in parallel. This clustered, non-federated approach benefits from understanding the characteristics of the entire training data set and the computational capabilities of the nodes in the cluster, as well as its ability to partition the dataset into convenient chunks among nodes in a cluster. These assumptions do not hold in a federated setting.

Challenges of FL arise from different perspectives including data heterogeneity, robustness of the federation process, selection of unbiased fusion operators, security and privacy inference prevention and operational and effective deployment in enterprise and multi-cloud settings, among others. While FL does not require the centralized management of training data, thus enabling machine learning to scenarios in which that is not possible, it also poses some new challenges, such as *data set heterogeneity.* In this context, there is no common view of the stochastic properties of the overall training data set, which precludes machine learning dependent on them, established pre-processing methods, fine-tuning algorithms and evaluation of model performance. This includes differences in distribution of data among participants, different dataset sizes for each party, different attributes and general issues of heterogeneity [11]. [16] provides a good overview of some challenges of federated learning.

Privacy demands might also add additional restrictions to a federated learning system. For example, while membership inference attacks [20], where an adversary may try to infer the training data used during the training process by querying the final model, are possible in both centralized and federated learning settings, additional measures need to be considered in federated learning cases. Some federated learning scenarios require that different parties are not able to derive insights about each other's training data based on messages exchanged during the training process (e.g., weights). Additionally, in some other cases, not even the aggregator may be trusted to see this information. Methods of creating differentially private noise or secure multi-party computation are used to meet these privacy requirements [33, 31, 13, 8]. This is mostly the case when data is owned by different enterprises or individuals, an example being data on phones. Multi-cloud settings may have a lesser need for this.

Another set of challenges relates to needs of the enterprise. (1) FL requires different *skills* from machine learning to distributed systems to cryptography. It's rare to find employees who have all these skills sets. An FL platform must provide a seamless on-ramp for its primary user base, the machine learning professional. (2) An FL platform must deal with *operational complexity.* An FL process requires a more complex infrastructure and processes to enroll parties, distribute models, set up an aggregator, deploy a federated learning algorithm, conduct the federated learning process, and then make a final model available. The system must be resilient to parties with different characteristics and parties dropping out of the training process. (3) *Security and networking* are important to a federated learning solution. Given that FL is typically used where domain boundaries have to be dealt with, networking must suit the deployment needs and facilitate the required security. However, the setup of secure communication should not be burdensome to deploy, avoiding time-consuming processes such as opening ports as much as possible.

A few approaches and libraries for FL address some of these issues, including [9], [28], and [4]. However, *IBM Federated Learning* focuses on enterprise settings where secure deployment, failure tolerance, and fast model specification are paramount; these must use existing machine learning libraries that allow enterprise users to take advantage of a comprehensive set of state-of-the art algorithms without needing to learn new languages.

We propose the **IBM Federated Learning framework** to address these challenges and facilitate an easy integration of FL into productive machine learning workflows in the enterprise. Data scientists and machine learning professionals can benefit from a seamless transition from existing practices of model development to writing federated machine learning models. FL infrastructure must be easy to deploy by an enterprise and fit into the typical operations patterns of IT infrastructure. Finally, researchers in federated learning, those interested in designing novel federated learning algorithms and protocols, can try out their ideas with ease and build on the existing functionality for the specific needs of their organization or the particular application domain. The goal of IBM Federated Learning is to address all of these needs in an easy-to-use framework.

The remainder of this white paper is organized as follows. In section 2, we present the concepts and terminology. Then in section 3, we present the architecture of system. Section 4 demonstrates how to train different types of models; in particular, we show how neural networks and decision trees can be incorporated into the framework. In section 5, we show how *IBM Federated Learning* is implemented and how it can be configured, and we conclude in section 6.

2 Concepts and Terminology

Like any machine learning task, FL trains a model $\mathcal{M}$ over data D. $\mathcal{M}$ can be a neural network or any non-neural model. In contrast to centralized machine learning, D is split over n parties, where each party P_i has its own private training dataset D_i. An FL process involves an *aggregator* A and those n parties $P_1, P_2, ..., P_n$ in a way that no party has knowledge of any other dataset than its own. A has no knowledge of any dataset.

The FL process is shown in Figure 1. To train a global machine learning model $\mathcal{M}_\mathcal{G}$ the aggregator and the parties participate in a *federated learning algorithm* that is executed by the aggregator and the parties by sending messages. The overall process runs as follows:

1. To train $\mathcal{M}_\mathcal{G}$, the aggregator uses a function $\mathcal{Q}$ that takes as input the current model or state of the training $\mathcal{M}_t$ at round t, and generates a next query q_{t+1}[1].

2. One such query, q_t, requests information about a local model or aggregated information about each party's data set. Example queries include requests for gradients or model weights of a neural network, or counts for decision trees.

3. The local training process applies a function $\mathcal{L}$ that takes query q_t and the local dataset D_i and outputs a *model update* $r_{i,t}$. Usually the query, q_t, contains information that the party can use to initialize the local training process, for example, model weights to start with local training, or candidate feature values and/or class labels to compute counts for.

4. $r_{i,t}$ is sent back from party P_i to the aggregator A, which collects all the $r_{i,t}$ from parties P_i.

5. When parties model updates $r_{1,t}, r_{2,t}, ..., r_{n,t}$, where $r_{i,t}$ refers to the model update of party i at round t, are received by the aggregator forming set $R_t = \{r_{1,t}, r_{2,t}, ..., r_{n,t}\}$, they are aggregated by applying fusion function $\mathcal{F}$ that takes as input R_t and returns $\mathcal{M}_t$.

This process can be executed over multiple rounds t and continues until a termination criterion is met, e.g., the maximum number of training rounds k has elapsed, resulting in a final global model $\mathcal{M}_\mathcal{G} = \mathcal{M}_k$. The number of rounds required can vary highly, from a single model merge of a Naive Bayes model to many rounds of training for typical gradient-based machine learning algorithms.

The local training function $\mathcal{L}$, the fusion function $\mathcal{F}$, and the query generation function $\mathcal{Q}$ are typically a complimentary set that are designed to work together. $\mathcal{L}$ interacts with the actual dataset and performs the local training, generating the model update $r_{i,t}$. The content of R_t is the input to $\mathcal{F}$ and, thus, must be interpreted by $\mathcal{F}$, which creates the next model, $\mathcal{M}_t$, from this input. If another round r is required, $\mathcal{Q}$ then creates another query. In case of a neural network, for example, $\mathcal{L}$ is the

[1]Some FL algorithms may include additional inputs for $\mathcal{Q}$ and may tailor queries to each party, but for simplicity of discussion and without loss of generality, we use this simpler notation.

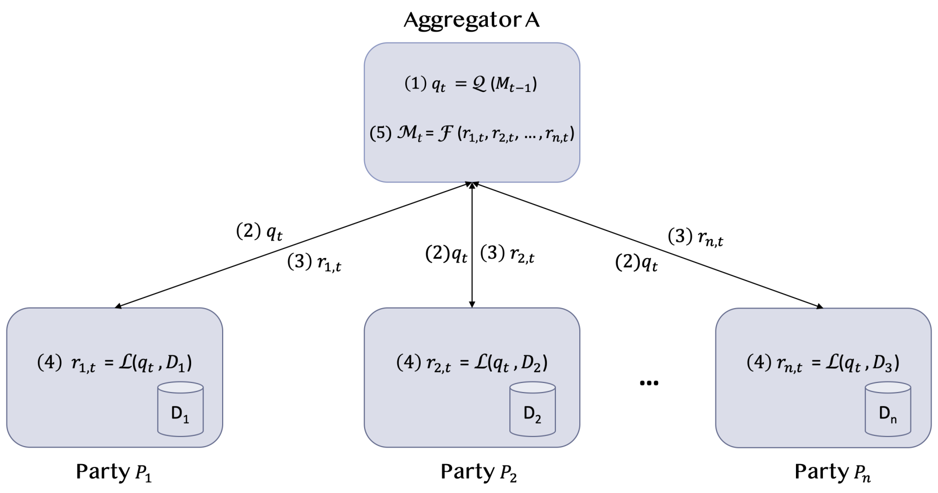
Aggregator A
(1) $q_t = \mathcal{Q}(M_{t-1})$
(5) $\mathcal{M}_t = \mathcal{F}(r_{1,t}, r_{2,t}, \ldots, r_{n,t})$
(2) q_t
(3) $r_{1,t}$
(2)q_t
(3) $r_{2,t}$
(2)q_t
(3) $r_{n,t}$
(4) $r_{1,t} = \mathcal{L}(q_t, D_1)$
D_1
(4) $r_{2,t} = \mathcal{L}(q_t, D_2)$
D_2
...
(4) $r_{n,t} = \mathcal{L}(q_t, D_3)$
D_n
Party P_1
Party P_2
Party P_n

Figure 1: Federated learning concepts where each party P_i owns its own dataset D_i.

local neural network training algorithm that might produce a weight vector as model update. $\mathcal{F}$ is a fusion algorithm such as a federated average that simply averages each weights over the parties P_i, resulting in a new model $\mathcal{M}_t$. For the next round, $\mathcal{Q}$ would then pass $\mathcal{M}_t$ as part of the new query q_{t+1}. In this case, $\mathcal{L}$ is computationally expensive and $\mathcal{F}$ less so. If the decision tree is trained in a federated way, one approach might be to balance the decision tree in the aggregator: implement $\mathcal{F}$, send leaf node definitions as part of q_t, $\mathcal{L}$ create a response based on a number of data samples corresponding to each leaf node in D_i, and then balance the tree again in the aggregator. In this case, $\mathcal{F}$ is computationally expensive and $\mathcal{L}$ is not.

We can introduce different variants to this basic model. While an approach with a single aggregator is the most common and practical for most scenarios, we might choose other configurations. For example, each party P_i might have its own, associated aggregator A_i, querying the other parties. Function $\mathcal{Q}$ might determine the parties to query in each round perhaps based on the merits of prior contributions. Queries to each party might be different, with $\mathcal{F}$ needing to integrate the results of different queries in the creation of a new model $\mathcal{M}_t$. However, for the further discussion of *IBM Federated Learning* we will focus on FL with a single aggregator.

3 Architecture

IBM Federated Learning is a Python library designed to support the machine learning process shown in Figure 1, while enabling an easy set up in a real distributed environment.

IBM Federated Learning is designed to implement a resilient platform as well as to ensure the easy implementation of new FL algorithms. The *IBM Federated Learning* library contains the components implementing an aggregator A and a party P_i as shown in Figure 2.

This modular design allows the framework to provide communication infrastructure independently from the federated learning algorithm and the actual machine learning library that performs local training. It also enables all parties P_i to read and preprocess data from different locations in different formats.

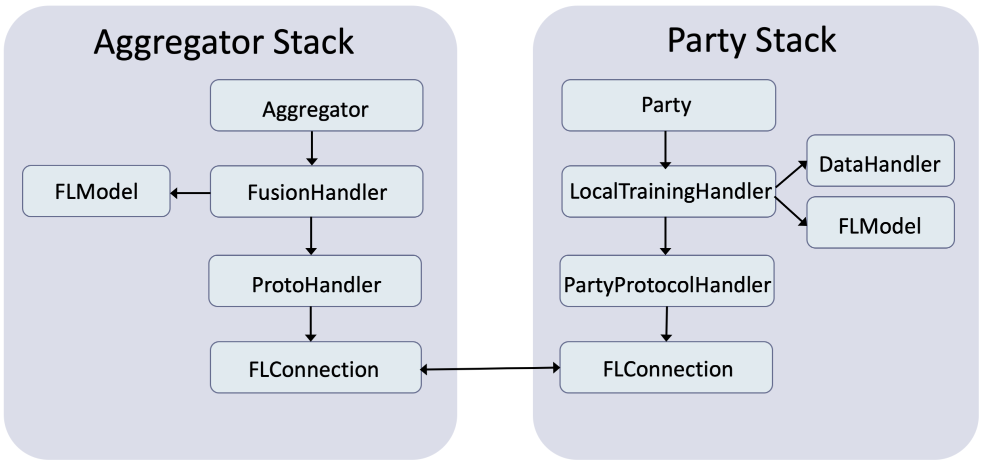

Figure 2: IBM federated learning architecture stack.

The architecture has the following components:

- **Connection:** Networking is often critical to adoption of any distributed system. The ability to deal with specific requirements and constraints such as available ports, bandwidth, connection stability and latency is important to managing the complexity of deploying a FL system. Multi-Cloud scenarios have different requirements from mobile phone, edge, or company collaboration scenarios. All functionality related to network connectivity for communication between the aggregator and each registered party is handled by the `FLConnection` component. *IBM Federated Learning* can support multiple connection types, including the Flask web framework [2], gRPC [3], and WebSockets.
- **Protocol handler:** This component governs message exchange between parties, i.e. the learning protocol. The message set of the protocol includes a query q, a model update r, and other messages to establish and dismantle the FL configuration, such as party registration. It uses the `FLConnection` for communication and is used by components higher in the stack. It implements which types of messages an aggregator or party can receive at a given point in time. The protocol handler is slightly different for the aggregator and the party stacks due to the differences in messages exchanged. The aggregator contains a `ProtoHandler` to manage the protocol to the set of parties $P_1, ..., P_n$, and on the party side, the `PartyProtoHandler` implements this function. The party and aggregator side protocol handlers can be adapted to the needs of a particular FL scenario by providing a specialized protocol handler pair.
- **Data handler:** A `DataHandler` is responsible for accessing and pre-processing the local dataset D_i at a Party P_i. When running an FL process, the data from each party must

be in the correct format so that the learning algorithm can make use of it. While in some FL scenarios, such as mobile phone applications, one will find the same data structure at each party, enterprise scenarios often face different data structures in applications in different clouds or in data centers of different companies. The `DataHandler` provides the abstraction to import data specifically for a party. The `DataHandler` is designed as an easily customizable module for each party to prepare their data to run. Other *IBM Federated Learning* modules will only interact with the data handler via limited APIs, for instance by *get_data* to access training and testing data sets.

- **FL training modules:** FL machine learning algorithms are incorporated into *IBM Federated Learning* by specifying a `FusionHandler` and a `LocalTrainingHandler` for the aggregator and parties, respectively.
  - `FusionHandler`: This module contains the specification of functions $\mathcal{Q}$ to generate queries and $\mathcal{F}$ to fusion model updates. The module uses high level APIs provided by the `ProtocolHandler` to send and receive messages generated by $\mathcal{Q}$ and also obtain all party replies to aggregate model updates using the specified $\mathcal{F}$ function. Different implementations of $\mathcal{Q}$ and $\mathcal{F}$ may require different information to be exchanged. For example, in some $\mathcal{Q}$ implementations a message may be added in addition to the query hyperparameters. To accommodate this differences of the information exchanged by the `FushionHandler`, *IBM Federated Learning* makes use of a dictionary, a generic data structure with customizable fields.
  - `LocalTrainingHandler`: This module is used to specify function $\mathcal{L}$ that will be run at the parties' side, to generate model updates and send them to the aggregator. The `LocalTrainingHandler` may use a `FLModel`.
  - `FLModel`: This module provides an interface to define a standard API to train, save, evaluate, and update the model, as well as generate model updates to be specified in different ML libraries, such as Keras or scikit-learn. For each ML library, a module is instantiated to wrap all the offered API (e.g., evaluate, train, save, update model). In this way, the `FusionHandler` and the `LocalTrainingHandler` can use the standard API and, as a result, be used without changes for multiple supported ML libraries.

3.1 Aggregator stack

The aggregator stack contains the components implementing an aggregator A. Its role includes the coordination of the federated learning process, the execution of $\mathcal{F}$, the fusion algorithm, and persisting the meta-data of the FL process.

The metadata about a federated learning process includes information such as the list of parties, the current state of the process, logging information, and aggregated models $\mathcal{M}_t$. It also performs management tasks such as starting and closing the training process.

As shown in Figure 2, the aggregator works with the `ProtoHandler` and the `FLConnection` to communicate with data parties. The `FusionHandler` serves to generate queries and to aggregate received replies. In some cases, it is desirable to know the performance of a global model during the training process, for instance to evaluate early termination criteria. In such cases, the aggregator may have access to some testing samples which are accessed through an optional `DataHandler`.

The aggregator provides an interface to the users to issue commands to control the overall federated learning process. Figure 3 shows different phases of the aggregator which include `REGISTERING`, `TRAINING`, `SYNCING`, `EVALUATING`, `STOPPING` and `PROCESSING_ERROR`.

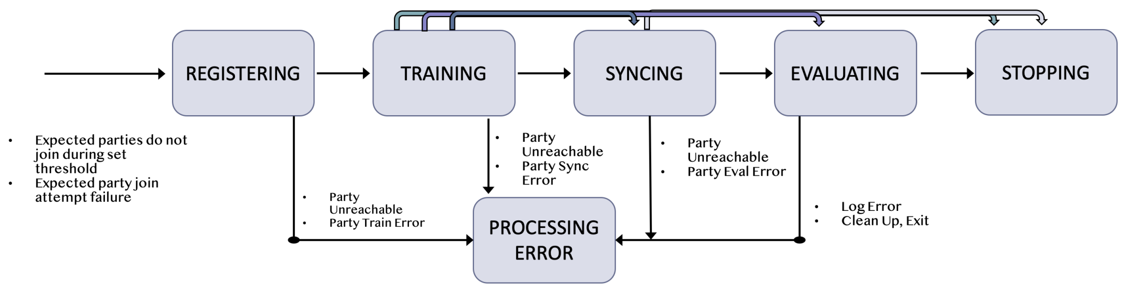

Figure 3: Aggregator phases.

Once the aggregator is started, it waits for the data parties to register. Once the registration process is completed and there is a quorum, a user can issue a `TRAINING` request to start the training process, which triggers the execution of the `FusionHandler`. During this process, the aggregator uses the `ProtoHandler` and the `FLConnection` to send the queries generated by the `FusionHandler`. Each party responds to this request by sending the trained model updates. Making use of the `FusionHandler`, the aggregator computes $\mathcal{F}$ to aggregate all model updates and generate the queries for parties. This process is repeated until we reach a termination criteria, i.e. desired number of training rounds or specific model accuracy.

Once termination criteria is reached, the aggregator sends a final model update to all the parties via a `SYNCING` command which distributes the final global model $\mathcal{M}_{\mathcal{G}}$. Next, the aggregator may request parties to perform a local evaluation before stopping the federated learning process. If an error arises during any of these steps, the aggregator logs it and stops the federated learning process. *IBM Federated Learning* collects metadata that allows for failure recovery by going to a previous valid phase.

3.2 Party stack

Similar to the aggregator, the party side consists of its own protocol handler, connection, model, local training, and data handler. Unlike the aggregator, the `DataHandler` is mandatory for parties. The party provides a command interface for users to start interacting with the application. Each party starts with issuing a `REGISTER` command to join a federated learning process. After the registration process is completed, the party waits for the aggregator to send a message with a query q to start running the `LocalTrainingHandler`, which executes $\mathcal{L}$ based on the received q and produces a reply r. Then, r is sent back to the aggregator. This process is repeated until the aggregator sends a request to stop the federated learning process.

In some cases, the aggregator may also issue a `SYNC` request, which includes the final global model $\mathbf{M}_{\mathcal{G}}$.

4 Supporting different learning paradigms

In the past few years, a variety of algorithms have emerged for training different types of ML models via federated learning, e.g., [19, 10, 33, 31, 35, 34, 7, 32, 14]. Since FL is an approach to perform collaborative learning with the help of an aggregator for coordination, existing FL algorithms usually contain two parts: 1) A global training module, the `FusionHandler`, mainly contains the fusion function $\mathcal{F}$, and the query generation function $\mathcal{Q}$; 2) A local training module, the `LocalTrainingHandler`, contains the local training function $\mathcal{L}$. They come in pairs to execute a FL algorithm in *IBM Federated Learning*. However, different federated learning algorithms vary significantly in several aspects: how and where the global model $\mathcal{M}_{\mathcal{G}}$ is updated; the query generation function $\mathcal{Q}$; the fusion function $\mathcal{F}$; the function $\mathcal{L}$ preformed by the parties and the content of model update r, among others. *IBM Federated Learning* is designed to be flexible enough to accommodate different learning paradigms in FL. We cover the design of several different FL algorithms for *IBM Federated Learning* below.

4.1 Neural networks

In this subsection, we focus on the discussion of training neural networks via iteratively updating $\mathcal{M}_{\mathcal{G}}$ by the (weighted) average of local models' parameters.

In particular, this type of FL algorithm requires the aggregator to generate a query, q_t, containing the current global model weights - and hyperparameters, optionally - and send it to all or a subset of parties at round t.

This query is implemented as a dictionary. Once parties receive q_t, each party initializes its local model with the global model weights, performs several epochs of local training with the received hyperparameters, if any, to obtain the model update r_t. The resulting r_t contains the new set of local model weights, and shares it with the aggregator. In some variations, where a weighted average is used (*FedAvg* [18]), the number of data points used for training at each party is included in r_t. The model update r_t is implemented as a dictionary, and so, when *FedAvg* is executed, two key values are included into the dictionary with key values $weights, nsamples$. In some cases, for privacy considerations, the number of samples at each party cannot be transmitted to the aggregator. In those cases, the model update only has a single-item dictionary. The aggregator then follows the corresponding fusion function $\mathcal{F}$ to conduct a (weighted) average over the collected $r_{i,t}$'s and updates the global model's weights with the fusion results. Both the global and local models' structure, i.e., the neural network architecture, stay the same.

What does this mean for the `FusionHandler`? Implementing the aforementioned type of FL algorithms, `IterAvgFusionHandler`, a subclass of `FusionHandler`, which, after being invoked by the aggregator, iteratively constructs a query containing the current weights of $\mathcal{M}$, and sends the query via the API offered by the protocol handler; this, in turn utilizes the connection layer to send the messages to parties upon receiving enough reply updates $\mathcal{M}_{\mathcal{G}}$ with the fusion results of $\mathcal{F}$. Depending on the fusion algorithms, the fusion function $\mathcal{F}$ can be a simple average of the collected $r_{i,t}$'s as implemented in `IterAvgFusionHandler`, or a weighted average whose weights depend on the parties' sample size as implemented in `FedAvgFusionHandler`. Note that the latter fusion function expects the replies $r_{i,t}$ to contain the parties' sample size in addition to the local model's weights.

The `LocalTrainingHandler` module implements the local training function $\mathcal{L}$ on the party side. The `LocalTrainingHandler` triggers the party to perform a predefined number of training epochs on the local model, and constructs a model update r containing the information requested by the corresponding `FusionHandler`. For example, a model update replying to the query constructed by `IterAvgFusionHandler` will contain the new set of local model weights, while a model update replying to the query constructed by `FedAvgFusionHandler` will contain the local sample size along with the local model weights.

IBM Federated Learning employs the `FLModel` module to allow the users to provide model definitions in standard ML libraries and to provide a single API to a pair of a `FusionHandler` and a `LocalTrainigHandler`. As a concrete example, a data scientist can specify a convolutional neural network using the standard Keras library [12]. `KerasFLModel` wraps the functionality for initializing the model according to the specified Keras model definition, training, creating a model update (extracting model weights from the neural network) and saving the final model.

Finally, it is worth mentioning that *IBM Federated Learning* also includes other `FLModel` modules. Our design ensures that the same FL algorithm, i.e., the same pair of a `FusionHandler` and a `LocalTrainigHandler`, can be used to train different models specified using different ML libraries. For instance, the two FL algorithms we discuss above can also be applied to train Scikit-learn [24] linear models in *IBM Federated Learning*.

4.2 Decision trees

Training a decision tree in a federated setting requires a different approach to implementing a federated learning algorithm than neural networks do. We illustrate this for an adapted ID3 algorithm. Since this type of decision tree deals with discrete feature values, its FL algorithm is very different than those we have discussed in the previous section.

IBM Federated Learning supports a FL variant of the ID3 algorithm [25], where the tree is grown at the aggregator side and the local training function, $\mathcal{L}$ only performs light computations to generate replies r containing the counts information. No initial model structure is provided to the aggregator and parties, and thus the aggregator starts with a null tree root node.

During the training process, the query generation function $\mathcal{Q}$ takes the current list of candidate feature values for splitting and class labels to query the parties for their counts information. The local training function $\mathcal{L}$ computes the corresponding counts based on the party's local dataset, and the resulting model update r is shared with the aggregator. The fusion function $\mathcal{F}$ splits the current tree node according to the information gain computed based on the collected counts. When the tree reaches the maximum depth or all tree nodes have no candidate feature values to split, the training process ends.

The `ID3FusionHandler`, invoked by the aggregator, implements the core parts of the ID3 FL algorithm, i.e., the fusion function $\mathcal{F}$ and query generation function $\mathcal{Q}$. $\mathcal{Q}$ recursively takes the updated list of candidate feature values, and constructs the query q with the candidate list. The query q is sent the same way to parties as by another `FusionHandler`.

After collecting a quorum of replies $r_{i,t}$, i.e., the list of counts sent by the parties, the fusion function $\mathcal{F}$ computes the sums of the counts over all parties with respect to their corresponding feature values and recovers the overall counts for the candidate feature values. Following the information gain formula used by the ID3 algorithm, $\mathcal{F}$ chooses the feature value with the maximum information gain to split and construct the resulting tree nodes according to the predefined criterion, whether reaching the maximum tree depth or having a empty candidate list. It also updates the candidate list and moves on to the new constructed tree node. When the training process finalizes all leaf nodes, the tree is grown at the aggregator side.

As we can see `ID3FusionHandler` performs the computation-heavy operations at the aggregator, different from the two FL algorithms we have discussed in Section 4.1 to train neural networks where the computation expensive training happens at the party side. The `LocalTrainingHandler` triggers a light operation conducted by the parties to compute counts information and constructs the model update r containing these counts.

In the case of an ID3 decision tree, *IBM Federated Learning* implements the `DTFLModel` independently of any existing ML library. It follows the same convention as other `FLModel` to implement a list of methods, such as *fit_model*, *update_model*, *get_model_update*, *evaluate_model* and *save_model*, etc., so that other modules can interact with it the same way as with other types of ML models.

4.3 Other models, differential privacy and multi-party computation

The flexible architecture of *IBM Federated Learning* also supports the implementation of other FL algorithms as well as the use of differential privacy and multi-party computation. In the following, we discuss how to use these approaches are implemented based on this library.

Other models and training algorithms: *IBM Federated Learning* supports the training of different machine learning models besides neural networks and decision trees. These include adaptations of XGBoost for binary and multi-class classification as well as regression, linear classifiers and regressions with regularizers including logistic regression, linear SVM, ridge regression and more. The framework also comes with Naïve Bayes and Deep Reinforcement Learning algorithms including DQN, DDPG, and PPO, among others.

With respect to FL algorithms, several common and advanced algorithms are now part of the framework's algorithm library. The set of common algorithms includes average and weighted aggregation, FedAvg [18], as described above. We have implemented multiple advanced algorithms including SPAHM [34] which uses a statistical model aggregation via parameter matching for supervised and unsupervised learning, and PFNM [35], which provides a communication efficient method for federated learning of fully-connected networks with adaptive global model size.

Algorithms to improve robustness such as Krum [7], a Byzantine tolerant gradient descent coordinate-wise median fusion [14], and Zeno, a distributed stochastic gradient descent with suspicion-based fault-tolerance [32] are also available. New algorithms, as needed, can be easily implemented within the existing framework and added back to the library.

Privacy-aware algorithms Real federated learning scenarios may require different techniques to ensure the federated learning process can be performed.

In particular, in some cases, participants may have limited trust amongst each other or in the aggregator. In these cases, it is imperative to add additional protection mechanisms, including secure multi-party computation and differential privacy.

The right technique to apply largely depends on the threat model of choice. Techniques such as local differential privacy [8], multi-party computation (SMC) [13, 33] and hybrid approaches have been proposed for this purpose [31].

Secure multi-party computation techniques allow for the private aggregation of inputs among parties, where a function value can be computed without revealing the function's inputs. Among relevant algorithms for this purpose include [13], which utilizes partial homomorphic encryption, [33], which makes use of functional encryption to drastically speed up the training process in comparison to homomorphic-based approaches, and [31], where differential privacy and SSM (threshold-based Pailler scheme) are combined to create highly accurate models without compromising the offered differential privacy guarantee.

Some of the previously mentioned SMC techniques require a different number of messages exchanged. For instance, while the approach presented in [33], based on functional encryption, only requires the aggregator to send a single query, q and participants to send a single encrypted model update, r, techniques like [31] require multiple communication rounds between the aggregator and parties. Ideally, all these nuances should be hidden, ensuring that a crypto-aware `Fushion_Handler` and `LocalTrainingHandler` do not need to be specifically adapted for each type of cryptosystem

IBM Federated Learning provides a simple API to incorporate a diverse set of cryptosystems that enable machine learning professionals to invoke crypto operations in the `Fushion_Handler` and `LocalTrainingHandler`. We designed this part of the system to ensure cryptosystems can be interchanged without modifying a crypto-aware pair of a `Fusion_Handler` and a `LocalTrainingHandler`.

We have incorporated several SMC techniques including partial homomorphic encryption [13], the approach proposed in [33] and the functional approach presented in [31].

IBM Federated Learning also provides the building block to add differential privacy for different types of models varying from very simple Naive Bayes to more complex differential privacy mechanisms as those required for neural networks. For the first type of cases, a one-shot differential privacy addition that can be applied by a mechanism at the `LocalTrainingHandler`. For other models, an accounting module is added at the aggregator side to keep tract of the differential privacy budget. Multiple variations on differential privacy methods may be implemented including [22, 5].

Machine learning libraries Finally, *IBM Federated Learning* was designed to ensure ML professionals can specify their models using common ML libraries. Our de-coupled design ensures that implemented `FusionHandler`s and `LocalTrainingHandler`s that use the APIs offered by `FLModel` can be reused for different libraries. Currently, we include `FLModel` implementations for Keras, Pytorch, Tensorflow, Scikit-learn and RLlib. This list can be extended by users implementing the interface of `FLModel` for other libraries.

5 Deployment and Configuration

Deploying the *IBM Federated Learning* environment involves deploying an aggregator stack and a party stack at each of the environments involved. Since *IBM Federated Learning* is a Python library, its deployment follows the same workflow as other Python packages, configuring its dependencies. The party library is typically installed where the local training will take place.

IBM Federated Learning is designed to allow different modules to be swapped in and out without interfering with the functionality of other components. This requires users to configure the choices that work for their configuration. Party and aggregator stacks can be configured via an external API configuration, or driven by a configuration file (YAML), which are different for the aggregator and each party. *IBM Federated Learning* provides scripts to generate configuration files with default settings which users can modify according to their requirements.

We now briefly discuss a list of building blocks that can be configured in *IBM Federated Learning*.

Connection: It is configured to provide information needed to initiate the connection between the aggregator and parties. For example, flask server information, synchronization mode for training requests, Transport Layer Security (TLS) configurations, etc.

Data: It is configured to provide information needed to locate and load the local data sets and perform any pre-processing techniques.

Federated learning algorithm: *IBM Federated Learning* supports a variety of FL algorithms to train different types of machine learning models. Configuring a federated learning algorithm usually contains two parts: 1) `FusionHandler` is configured to provide information about the Fusion algorithm used at the aggregator; 2) `LocalTrainingHandler` is configured to provide information about the $\mathcal{L}$ function.

Hyperparameters: It is configured to set up global and local training hyperparameters, e.g., global training round number, local training epochs, batch-size, learning rate, etc.

Model: It is configured to provide information needed to initiate a machine learning model, which can be defined via existing ML libraries, like Keras and Scikit-learn among others, or a self-implemented model class.

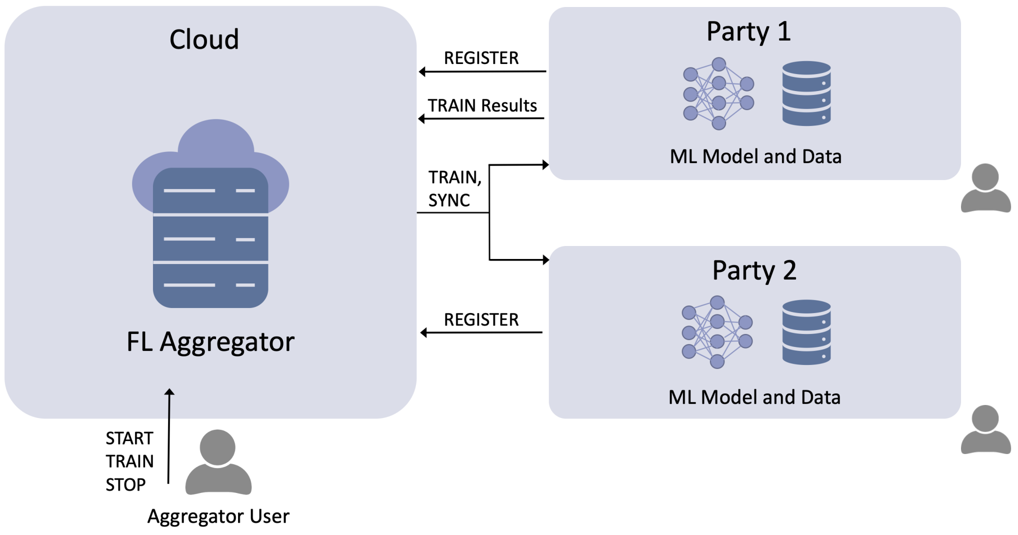

Figure 4: Flow to execute a FL job using *IBM Federated Learning*.

Protocol handler: It is configured to provide information needed to initiate a protocol which oversees the message exchange between the aggregator and parties.

After setting up the FL job details in configuration files, users can run a federated learning job in *IBM Federated Learning*.

The aggregator and party stacks come with a main application to run independently, or can be integrated into another application. To start the aggregator, a user sets up the proper environment with *IBM Federated Learning* installed, and launches the aggregator application with the aggregator configuration file. As shown in Figure 4, the aggregator application can be launched in a cloud or cluster environment.

Similarly, the party applications are launched in an environment with *IBM Federated Learning* installed and the individual parties' configuration files are provided to specify the FL job settings. After the aggregator application starts running, parties can register themselves via the `REGISTER` command. The aggregator waits for a quorum to start the FL training process using the `TRAIN` command.

The following table provides an overview of the commands of aggregator and party. In the applications included in the library they can be issued on the command line.

Table 1: A list of commands for *IBM Federated Learning*

IBM FL Command	**Participant**	**Description**
`START`	aggregator / party	Start accepting connections
`REGISTER`	party	Join an FL job
`TRAIN`	aggregator	Initiate training process
`SYNC`	aggregator	Synchronize model among parties
`STOP`	aggregator / party	End experiment process
`EVAL`	party	Evaluate model
`SAVE`	aggregator / party	Save the model locally at party side

The users can observe the FL training process via log information. Once the FL training finishes, the main applications provide a list of commands to be used to collect the FL results, see Table 1. If the library is embedded in an application it can be controlled and monitored in its contained application context.

6 Conclusion

This white paper introduces *IBM Federated Learning*, a Python framework for federated learning in the enterprise. This framework is a platform for executing federated learning algorithms by providing all of the basic elements of a federated learning infrastructure in an easily configurable way. On the one hand, it provides hooks to implement the party and aggregator parts of a federated learning algorithm. On the other hand, it is configurable to a variety of deployment scenarios, from mobile and edge scenarios, to multi-Cloud environments in an enterprise, to use cases across organizational boundaries. *IBM Federated Learning* is independent of a particular machine learning library or machine learning paradigm. It can be used for deep neural networks as well as "traditional" approaches such as decision trees or support vector machines. The abstractions for different model types provide a high level of reuse of federated learning algorithms for different machine learning libraries. A fusion algorithm for a deep neural network in Keras works as well with a native PyTorch one. Data access can be defined for each party independently. This is particularly important in the enterprise space where data stored in different data centers, Clouds or application silos can be brought together for a common learning task.

The key design point of *IBM Federated Learning* is fast start-up time for enterprise applications. There is a large library of federated learning algorithms implemented on *IBM Federated Learning* to get users started. Machine learning models tested successfully in a centralized learning application can often be extended to use data in different parties without rewriting the model code, but simply by editing configuration files. Party code is easy to deploy and, depending on the communication layer chosen, might not require opening ports on the party side, which makes deployment of a federated project faster and easier. Using *IBM Federated Learning*, enterprise practitioners can deploy federated learning quickly. In addition, researchers in the field can build on its existing platform to try out new federated learning algorithms and benchmark them against the existing ones in the library.

Acknowledgements

It takes the contributions of many to get a system ready for enterprise production. Thanks to Marius Danciu, Jim Rhyness, Frank Li, Calin Radu Coca, Rania Khalaf, Sandeep Gopisetty, Dinesh Verma, and Greg Filla for their contribution to IBM Federated Learning.

References

[1] Macy's is warning customers that their information might have been stolen in a data breach, July 2018. https://www.businessinsider.com/macys-bloomingdales-hack-disclosed-2018-7.

[2] Flask, (Accessed July, 01, 2020). https://flask.palletsprojects.com/.

[3] grpc, (Accessed July, 01, 2020). https://grpc.io/.

[4] FATE Github, (Accessed July, 01, 2020). https://github.com/FederatedAI/FATE.

[5] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[6] Accountability Act. Health insurance portability and accountability act of 1996. *Public law*, 104:191, 1996.

[7] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pages 119–129, 2017.

[8] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138, 2005.

[9] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[10] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. *To appear in ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2020.

[11] Zheng Chai, Hannan Fayyaz, Zeshan Fayyaz, Ali Anwar, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig, and Yue Cheng. Towards taming the resource and data heterogeneity in federated learning. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, pages 19–21, 2019.

[12] François Chollet et al. Keras. `https://keras.io`, 2015.

[13] I Damgard and M Jurik. A generalisation, a simplification, and some applications of paillier's probabilistic public-key system, presented at the 4th international workshop on practice and theory in public key cryptosystems, cheju island. *Korea*, 2001.

[14] Sumit Goel and Wade Hann-Caruthers. Coordinate-wise median: Not bad, not bad, pretty good. *arXiv preprint arXiv:2007.00903*, 2020.

[15] The Wall Street Journal. Google exposed user data, feared repercussions of disclosing to public, Oct. 2018.

[16] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[17] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pages 583–598, 2014.

[18] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[19] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[20] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[21] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[22] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017.

[23] NBC News. Yahoo to pay $50 million, offer credit monitoring for massive security breach, Oct. 2018.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[26] General Data Protection Regulation. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46. *Official Journal of the European Union (OJ)*, 59(1-88):294, 2016.

[27] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning, 2019.

[28] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[29] Toyotaro Suzumura, Yi Zhou, Natahalie Barcardo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Daniel Klyashtorny, Heiko Ludwig, et al. Towards federated graph learning for collaborative financial crimes detection. *arXiv preprint arXiv:1909.12946*, 2019.

[30] New York Times. Facebook security breach exposes accounts of 50 million users, September 2018. https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html.

[31] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, and Rui Zhang. A hybrid approach to privacy-preserving federated learning. *arXiv preprint arXiv:1812.03224*, 2018.

[32] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *International Conference on Machine Learning*, pages 6893–6901, 2019.

[33] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 13–23, 2019.

[34] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, and Nghia Hoang. Statistical model aggregation via parameter matching. In *Advances in Neural Information Processing Systems*, pages 10956–10966, 2019.

[35] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. *arXiv preprint arXiv:1905.12022*, 2019.