FederatedScope: A Flexible Federated Learning Platform for Heterogeneity

Yuexiang Xie*
Alibaba Group
yuexiang.xyx@alibaba-inc.com

Zhen Wang*
Alibaba Group
jones.wz@alibaba-inc.com

Dawei Gao
Alibaba Group
gaodawei.gdw@alibaba-inc.com

Daoyuan Chen†
Alibaba Group
daoyuanchen.cdy@alibaba-inc.com

Liuyi Yao†
Alibaba Group
yly287738@alibaba-inc.com

Weirui Kuang†
Alibaba Group
weirui.kwr@alibaba-inc.com

Yaliang Li‡
Alibaba Group
yaliang.li@alibaba-inc.com

Bolin Ding‡
Alibaba Group
bolin.ding@alibaba-inc.com

Jingren Zhou
Alibaba Group
jingren.zhou@alibaba-inc.com

ABSTRACT

Although remarkable progress has been made by existing federated learning (FL) platforms to provide infrastructures for development, these platforms may not well tackle the challenges brought by various types of heterogeneity. To fill this gap, in this paper, we propose a novel FL platform, named FederatedScope, which employs an event-driven architecture to provide users with great flexibility to independently describe the behaviors of different participants. Such a design makes it easy for users to describe participants with various local training processes, learning goals and backends, and coordinate them into an FL course with synchronous or asynchronous training strategies. Towards an easy-to-use and flexible platform, FederatedScope enables rich types of plug-in operations and components for efficient further development, and we have implemented several important components to better help users with privacy protection, attack simulation and auto-tuning. We have released FederatedScope at https://github.com/alibaba/FederatedScope to promote academic research and industrial deployment of federated learning in a wide range of scenarios.

PVLDB Reference Format:
Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. PVLDB, 16(5): 1059 - 1072, 2023.
doi:10.14778/3579075.3579081

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/alibaba/FederatedScope.

*Co-first authors.
†Equal contribution, listed in alphabetical order.
‡Corresponding authors.

Proceedings of the VLDB Endowment, Vol. 16, No. 5 ISSN 2150-8097.
doi:10.14778/3579075.3579081

1 INTRODUCTION

As one of the feasible solutions to address the privacy leakage issue when utilizing isolated data from multiple sources, Federated Learning (FL) [41, 57, 89] has rapidly gained enormous popularity in both academia and industry [31, 86, 88]. Such widespread adoption of FL is inextricably tied to the support of FL platforms, such as TFF [10], FATE [89], PySyft [96] and FedML [34], which provide users with functionalities to get started quickly and develop new FL algorithms and applications.

Although the existing FL platforms have made remarkable progress, there are more burgeoning demands from FL research and deployment, which are mainly brought by the heterogeneity of FL. Specifically, we summarize the heterogeneity of FL as the following four aspects.

(1) **Heterogeneity in Local Data**. The isolated data in FL vary a lot among the FL participants in terms of quality, quantity, underlying distributions, etc. Such heterogeneity in data can lead to the sub-optimal performance when applying the vanilla FedAvg [57], i.e., producing one global model for all the participants by the same local training process.

(2) **Heterogeneity in Participants' Resources**. Apart from the heterogeneity of data, the participants' resources can also be very different, including computation resources, storage resources, communication bandwidths, reliability, and so on. Thus, it would be better if FL platforms allow users to implement/execute FL with asynchronous training strategies [17, 82] to ensure both efficiency and effectiveness in real-world FL applications.

(3) **Heterogeneity in Participants' Behaviors**. Although participants only exchange homogeneous information in vanilla FedAvg, the practical and recent FL applications often require to exchange various types of information among participants and execute diverse training processes, which leads to rich behaviors. The heterogeneity in participants' behaviors prompts the FL platforms to support flexible expression for rich behaviors of participants.

(4) **Heterogeneity in Learning Goals**. Towards a more general scope of utilizing isolated data, some recent FL studies [56, 69, 85] propose to allow participants to collaboratively learn common knowledge while optimizing for different learning goals, which brings new challenges to FL platforms.

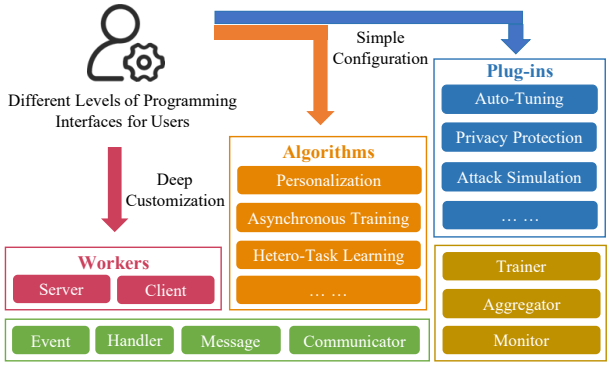

Figure 1: FederatedScope **provides different levels of programming interfaces for users.**

The aforementioned aspects of heterogeneity are commonly observed in real-world FL applications. Although we discuss them in the above four separate aspects, they can appear jointly in a single application. Facing such mixed heterogeneity, users are eager for an FL platform that has great **flexibility**: Participants should be allowed to express their diverse behaviors and different learning goals according to their own local data and system resources, and these participants can be effortlessly coordinated with synchronous or asynchronous training strategies for completing the federal training procedure based on a pre-defined consensus.

To provide such flexibility, we propose FederatedScope, a novel FL platform that employs the event-driven architecture [42, 61] to frame FL courses into <event, handler> pairs. Note that it is not trivial to build up a comprehensive FL platform with such a formalization. In particular, considering the heterogeneity of federated learning, such formalization is expected to express diverse behaviors of servers and clients for handling the heterogeneity, and be well-modularized so that users can conveniently develop new FL algorithms and applications. To fulfill this goal, the events provided in FederatedScope can be categorized into two types, i.e., *events related to message passing* and *events related to condition checking*, which are used to describe what happens in the FL courses from the perspective of an individual participant. The handlers, triggered by the events, describe what actions should be taken when a specific event happens. These events happen in the intended logical order and naturally trigger the corresponding handlers, which can precisely express various FL algorithms and procedures. All the participants can be coordinated with the pre-defined events related to message passing and condition checking to construct suitable FL course for specific scenarios and applications.

Besides the flexibility, as an FL platform, FederatedScope also provide great **usability**, that is, FederatedScope provides different levels of programming interfaces to meet different requirements from users, as demonstrated in Figure 1. For the users who want to design new FL algorithms, as discussed above, FederatedScope allows them to add new <event, handler> pairs to implement their ideas. For the users who want to directly apply existing FL techniques to certain application scenarios, FederatedScope provides rich sets of events and corresponding handlers, core functionalities and several important plug-in components, all of which can be directly called, and thus users only need to focus on a necessary set of interfaces to be integrated or implemented. For example, we have implemented several personalization federated algorithms, including applying client-wise configuration, maintaining client-wise sub-modules, global-local fusing, etc, for users' convenient usage. Besides personalization, FederatedScope provides users with functionalities such as asynchronous training, privacy protection and cross-backend FL, and several important plug-ins such as attack simulation for protection verification, and auto-tuning for helping users to automatically seek suitable hyperparameters.

Last but not least, FederatedScope also has great **extensibility**, which is brought by the fact that the set of <event, handler> pairs can be easily extended by adding new ones. Take personalization again as an example: to add a new personalization, users only need to add new behaviors (e.g., adopting client-specific training course) in the corresponding handlers. Such extension convenience also holds for all the other functions such as federated aggregators, asynchronous training, privacy-protection, etc. Through this way, FederatedScope can be easily extended to include new functions or plug-ins to satisfy new requirements brought by new developments and support a variety of new scenarios.

Contributions. Our contributions can be summarized as: (1) Motivated by the heterogeneity challenges from a wide range of FL applications, we propose and release FederatedScope, a novel FL platform to handle heterogeneity in FL. The proposed FederatedScope promotes the development of FL techniques and the deployment of FL applications. (2) With the event-driven architecture, FederatedScope provides users with rich yet extendable sets of events and corresponding handlers, core functionalities such as asynchronous training, personalization and cross-backend FL, and several important plug-in components. These implementations make it easy for users to apply FL algorithms in both academia and industry applications. (3) FederatedScope brings great flexibility, usability and extensibility to users, broadens the application scope and enables more tasks that would otherwise be infeasible due to challenges brought by various types of heterogeneity in FL.

2 PRELIMINARY

2.1 Problem Definition

Federated Learning (FL) [41, 57, 89], a learning paradigm for collaboratively training models from dispersed data without directly sharing private information, involves multiple participants who are willing to contribute their local data and computation resources. We use *server* to denote the participant(s) who are responsible for coordinating and aggregating, while other participants are *clients*. During a typical *training round* of an FL course, clients update the global model received from the server by training it with local data, and send the model updates back to the server for collaborative aggregation. In repeated training rounds, the (possibly sensitive) training data is always kept locally in each client; the server and clients only exchange aggregated and meta information. To further satisfy different types of formal privacy protection requirements, various privacy protection techniques can be integrated into FL, such as Differential Privacy (DP) [74, 81], Homomorphic Encryption (HE) [26, 32], and Secure Multi-Party Computation (MPC) [11, 59]. In short, the goal of FL is to jointly train a

global model in a privacy-preserving manner and achieve a better performance compared to that without collaboration.

Formally, there are M clients, and the m-th client owns a *private training dataset* $\mathcal{D}_m = \{(x_i^{(m)}, y_i^{(m)}) \in \mathcal{X} \times \mathcal{Y},\ i = 1, 2, \ldots, |\mathcal{D}_m|\}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input feature space and the label space, respectively. $\mathcal{D}_m$ is stored in the m-th client's private space, and $n = \sum_{m=1}^{M} |\mathcal{D}_m|$ is the total number of training instances. Without sharing $\mathcal{D}_m$ directly with each other and the server, the M clients together aim to train a model $h_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by θ, with the *loss* $F : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+ \cup \{0\}$. The *FL loss function* is:

$$\mathcal{L} = \frac{1}{n} \sum_{m=1}^{M} \sum_{(x_i^{(m)}, y_i^{(m)}) \in \mathcal{D}_m} F\left(h_\theta(x_i^{(m)}), y_i^{(m)}\right). \tag{1}$$

Extensions. For the simplicity of presentation, we focus on a *vanilla FL* to minimize the loss function in Equation (1) in most parts of this paper. Our FederatedScope easily supports different federated settings in real-world FL applications, with more complicated loss functions, in order to handle the heterogeneity as discussed in Section 1. For example, for the purpose of personalization, the input feature spaces, the label spaces, and the underlying learning goals can be different for different clients. In FederatedScope, clients can adopt different models and loss functions in local training, and only federally train the shared parts of the models. We will discuss more details in Section 3.4.

2.2 Related Works

Comparisons with existing FL platforms. In the recent years, growing along with the development of federated learning, federated learning platforms, including TFF [10], FATE [89], LEAF [12], PySyft [96], FedML [34], FedScale [44], etc., are proposed to support various kinds of applications. These federated learning platforms provide data, models and algorithms, which saves users' effort on implementing from scratch and makes it easy for developers. Most of these existing platforms adopt a procedural programming paradigm, which requires users to explicitly declare a sequential training process and computational graph from the global perspective. However, such a design makes the existing FL platforms kind of rigid and thus might be unable to provide the required flexibility and extendability for the burgeoning demands from FL research and deployment. Meanwhile, users also expect FL platforms to become more convenient to handle the aforementioned heterogeneity in real-world FL applications.

To tackle these challenges, we propose FederatedScope to provide great flexibility and extendability for users to handle the heterogeneity in FL. FederatedScope provides rich implementations of FL algorithms for convenient usage, and provides different levels of programming interfaces for users to develop new algorithms.

Comparisons with distributed machine learning. In distributed machine learning, the server has the rights to control the behaviors of clients; While in federated learning, all the participants could have their own behaviors following the achieved consensus to collaboratively train the model. For example, given the consensus that participants only need to share parts of the model parameters, clients can apply client-wise training configurations (e.g., training steps, learning rate, regularizer, optimizer, and so on)

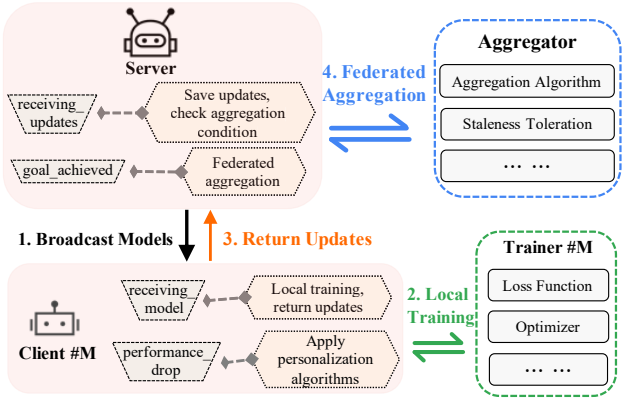

Figure 2: An FL round implemented with FederatedScope.

to locally train the model, and keep the non-shared model parameters and the learning goals (might be different among clients) private. To satisfy such requirements, FederatedScope give the rights to the participants to describe behaviors from their own respective perspectives. Besides, in federated learning, the quality, quantity, and distributions of clients' local data can be very diverse. Such heterogeneity in data makes it challenging to collaboratively learn, which motivates FederatedScope to provide novel functionalities, such as asynchronous training strategies (in Section 3.3) and personalized federated learning algorithms (in Section 3.4), to make better use of their isolated data. Furthermore, there exist more privacy/security protection requirements in federated learning compared to distributed machine learning. To tackle this, FederatedScope provides some Byzantine fault tolerance algorithms to defend the malicious participants (in Section 3.6), and privacy protection component (in Section 4.1) and attack simulation component (in Section 4.2) to enhance and verify the privacy protection strength of FL applications.

3 DESIGN OF FEDERATEDSCOPE

In this section, we introduce the design of FederatedScope, showing how an FL training course can be framed and implemented using an event-driven architecture and why FederatedScope makes it easy to handle heterogeneity in federated learning.

3.1 Overview

An FL course consists of multiple rounds of training, and a typical round implemented with FederatedScope is illustrated in Figure 2, which includes four major steps: (1) **Broadcast Models:** the server broadcasts the up-to-date global model to the involved clients; (2) **Local Training:** once received the global model, clients perform local training using their trainer based on their private data; (3) **Return Updates:** after local training, clients return the model updates to the server; (4) **Federated Aggregation:** with the help of an aggregator, the server performs federated aggregation on the received model updates, and optimizes the global model. To facilitate efficient development and deployment of such an FL course with multiple computation/communication rounds and different roles, there are two important design principles of FederatedScope.

- *Minimal dependency between different roles.* In FederatedScope, each client or the server takes care of only the minimal portion of job it needs to collaboratively accomplish, including the model to be collaboratively learned and the exchanged messages. While allowing both synchronous and asynchronous training, we want to avoid introducing too much duty of coordinating and scheduling to the server. This is important especially when we consider the heterogeneity of resources and learning goals for the clients.
- *Flexible and expressive programming interfaces for algorithm development and plug-in.* FederatedScope aims to enable efficient development of FL algorithms via proper abstraction of FL courses and providing a necessary set of interfaces that developers need to implement. Moreover, for the purpose of privacy protection and other functionalities, operators (e.g., for noise injection and encryption) and components (e.g., for auto-tuning) need to be plugged into the FL course in a flexible way.

Based on these principles, we first give an overview of our design.

Basic infrastructure. FederatedScope employs an *event-driven* architecture within which the behaviors of different clients and the server in an FL course can be programmed (relatively) independently. The information exchange among participants and conditions to be checked by participants during the FL course are described as *events* (trapezoids within pink areas in Figure 2); when an event occurs, the corresponding *handlers* (hexagons within pink areas) that describes the behavior of a participant is triggered. For example, when "receiving_models" occurs, "local training" in a client is triggered; when "goal_achieved" occurs, "federated aggregation" in the server is triggered. It turns out that the pairs of events and handlers are sufficiently expressive to describe all the existing (both synchronous and asynchronous) FL algorithms, as well as new ones we implement in FederatedScope.

With such an infrastructure, FederatedScope can easily support different machine learning backends (e.g., PyTorch and TensorFlow). All users need to do is to transform exchanged information (called *messages*), which might be related to participants' local backends, into backend-independent ones before sharing, and parse the received messages according to the receiver's backend for further usage. We call this procedure *message translation*.

Programming interfaces. Within the above infrastructure FederatedScope provides, for each client or server, we only need to implement a *Trainer* (green dashed rectangles in Figure 2) or *Aggregator* (blue dashed rectangles), respectively, which encapsulates the details of local training or federated aggregation with well-defined interfaces, e.g., the loss function, optimizer, training step, aggregation algorithms, etc. A Trainer can be implemented as if a machine learning model is trained on the local data owned by a client. Besides Trainer and Aggregator, the design of FederatedScope allows flexible plug-in operators and components. For example, in order to ensure differential privacy, noise injection operators can be plugged to perturb the messages to be sent, where the amount of noise can be customized for different training tasks. More details of the programming interfaces can be found in Section 3.6.

In this way, the server performs federated aggregation under flexibly triggered conditions, which can prevent the training process from being blocked by unreliable or slow clients (more details in Section 3.3). Different clients may customize their training configurations according to their own data distributions, tasks, and resources, such as training with different trainers for personalization (Section 3.4.1), learning toward different goals (Section 3.4.2), and running on different backends (Section 3.5). FederatedScope also provides some native plug-in modules (Section 4) for various important functionalities, including privacy protection, attack simulation, and auto-tuning. Before diving into these parts, we first provide more details about the event-driven design of FederatedScope in Section 3.2.

3.2 Event-driven Architecture

Event-driven architectures are widely adopted in distributed systems [42, 61]. With such an architecture, an FL training course in FederatedScope can be framed into <event, handler> pairs: the participants wait for certain *events* (e.g., receiving model parameters broadcast from the server) to trigger corresponding *handlers* (e.g., training models based on the local data). Hence, developers can express the behaviors of a participant (a server or a client) independently from its own perspective, rather than sequentially from a global perspective (considering all the participants together), and the implementations can be better modularized.

The events in FederatedScope are categorized into two classes. One is related to message passing, which is also considered in previous FL platforms, e.g., receiving user-defined messages in FedML [34] and invoking requests in FedKeeper [13]. The other class of events checks the satisfaction of customizable conditions (e.g., whether a pre-defined percentage of feedback from clients has been received). Some examples of events provided in FederatedScope are presented in the full version [1].

• *Events Related to Message Passing.* The exchanged information among participants are abstracted as messages, and an FL training course consists of several rounds of message passing. Multiple types of messages are involved in an FL course, including but not limited to building up (e.g., *join_in* and *id_assignment*), training (e.g., *model_param* and *gradients*), and evaluating (e.g., *metrics*). For the participants, receiving a message can be regarded as an event, and their follow-up behaviors can be described in handling functions (i.e., the handlers) to handle the received messages. A handling function can be invoked by the event of receiving one or more types of messages, while receiving a certain type of message should only trigger one handling function directly.

Take FedAvg as an example, the clients' handling function for the event "receiving_models" can be "*locally train the global model and return the model updates*", and the servers' handling function for the event "receiving_updates" can be described as "*save the model updates and check whether all the feedback has been received*".

Generally, by defining the events related to message passing, FederatedScope provides users with expressiveness to flexibly describe heterogeneous message exchange, such as exchanging model parameters, gradients, public keys, embeddings, generators, and so on. Meanwhile, through customizing the operations in the corresponding handlers, users can conveniently describe rich behaviors of participants, including training models based on the local data with personalized configurations, performing federated aggregation, predicting, clustering, generating, etc.

• *Events Related to Condition Checking.* Apart from the events related to message passing, the events related to condition checking are also indispensable for FL implementations. These events and the corresponding handlers describe the participants' behaviors when certain conditions are satisfied. For example, in an FL course, for the purpose of synchronization in training, the server checks whether the updated gradients or model parameters have been received from all the clients; if yes, it invokes an event "all_received", and this event triggers the federated aggregation and pushes forward the training process.

One important usage of the events related to condition checking is to express the customizable conditions for triggering the federated aggregation. Besides "all_received", in order to support asynchronous training, FederatedScope also provides events "goal_achieved" and "time_up" for such purpose. Specifically, "goal_achieved" indicates that a certain percentage of feedback (so-called aggregation goal) has been received, and "time_up" denotes that the user-allocated time budget for each training round has run out. Different from the event "all_received" that forces the server to wait for feedback from all the clients, "goal_achieved" allows the training process to move forward once the server has received enough feedback, while "time_up" encourages the server to collect as much feedback as possible within the time budget, both of which enable different asynchronous training strategies in FL.

Furthermore, the events related to condition checking also can be used to describe the behaviors of participants. For example, the server can be equipped with the events "all_joined_in" and "early_stop" to describe when to start and when to terminate the training process, respectively.

FederatedScope provides warnings if there exist conflicts, and adopts a default resolution following the "overwriting" principle. Specifically, in an FL course implemented with FederatedScope, each event is only permitted to be linked with one handler directly during the execution process. If an event is linked with more than one handler, which might cause conflicts in an FL course, a warning would be raised for users by FederatedScope, and the latest linked handler would overwrite the older ones (e.g., the default handler is overwritten by the user-customized handlers). Finally, the handlers that take effect in an FL course would be printed out and recorded in the experimental logs. Users can remove some handlers or adjust the linked orders to make sure the intended handlers would take effect in the constructed FL courses.

FederatedScope provides lots of predefined <event, handler> pairs, which cover the rich implementation of existing FL algorithms, such as FedAvg [57], personalization [48, 51, 70], federated graph learning approaches [79], and so on. Users can implement their own algorithms based on these provided <event, handler> pairs. However, it is out of our scope here to exhaustively list all the possible events related to message passing and condition checking. The most important advantage is that the event-driven design of FederatedScope provides users with expressiveness and flexibility to implement and customize diverse FL algorithms. Next, with FederatedScope, we will demonstrate how to execute asynchronous federated training (Section 3.3), how to describe rich behaviors of the participants (Section 3.4) and how to conduct cross-backend FL (Section 3.5) in order to handle the heterogeneity of FL.

3.3 Supporting Asynchronous Training

The asynchronous training strategies have been successfully applied in distributed machine learning to improve training efficiency [17, 53, 92]. Considering the aforementioned heterogeneity of FL in Section 1, the asynchronous training strategy is important to balance the model performance and training efficiency, especially in cross-device scenarios that involve a large number of unreliable and diverse clients. With the provided events and handlers, which specify what actions to take (i.e., handlers) when certain customizable conditions are satisfied (i.e., events), FederatedScope supports users to conveniently design and implement suitable asynchronous training strategies for their FL applications.

Compared with the synchronous training, several unique behaviors of participants might happen in asynchronous FL, which are modularized and provided in FederatedScope as follows:

(i) *Tolerating staleness in federated aggregation.* The term "staleness" denotes the version difference between the up-to-date global model maintained at the server and the model that a client starts from for local training, which should be tolerable to some extent in asynchronous FL. Specifically, in the federated aggregation, the staled updates from slow clients might be discounted in the aggregator but they still contribute to the aggregation. Of course, when the staleness is larger than a pre-defined threshold, the updates become outdated and thus can be directly dropped out.

(ii) *Sampling clients with responsiveness-related strategies.* The uniform strategy for sampling clients [57] might bring model bias in asynchronous FL, since the clients with low response speeds would contribute staled updates with higher probabilities compared with those who respond fast, which implies that the contributions of slow clients would be discounted or even dropped out in federated aggregation. Similar phenomena are happened in synchronous FL using over-selection mechanism [10], as pointed out by previous studies [38, 49, 63]. To tackle such an issue, with the prior knowledge of response speeds (it can be estimated from device information or historical responses), FederatedScope provides a responsiveness-related sampling strategy (i.e., the sampled probabilities are related to the response speeds) and a group sampling strategy (i.e., clients with similar response speeds are grouped).

(iii) *Broadcasting models after receiving update.* With the synchronous training strategy, the server broadcasts the up-to-date model to the sampled clients after performing federated aggregation. Such a broadcasting manner, denoted as *after aggregating* here, can also be adopted in asynchronous FL [82]. We also provide another broadcasting manner to achieve asynchronous FL, named *after receiving* [63], in which the server sends out the current (up-to-date) model to a sampled idle client once the feedback is received. Compared with *after aggregating*, the *after receiving* manner can keep the consistent concurrency and promotes an efficient FL systems [38].

An example of asynchronous FL with FederatedScope and a theoretical analysis of convergence when applying the asynchronous training strategies can be found in the full version [1]. To the best of our knowledge, with the provided events and the corresponding handlers that describe the above behaviors, most of the existing studies on asynchronous FL can be conveniently implemented with FederatedScope. For example, FedBuff [63] proposes

to register the event "goal_achieved" and apply the *after receiving* broadcasting manner, while SAFA [82] suggests to equip *after aggregating* broadcasting manner with event "goal_achieved" and manages clients based on their stalenesses. Particularly, a synchronous FL course with the over-selection mechanism can be easily implemented in FederatedScope by using event "goal_achieved" and setting the toleration to 0 (i.e., dropout all staled update).

In a nutshell, FederatedScope is well-modularized toward flexibility and extensibility for handling the heterogeneity of FL via applying asynchronous training strategies.

3.4 Supporting Personalization & Multi-Goal

In many real-world applications, handling the heterogeneity of FL brings the requirements of the flexibility of participants' training behaviors. That is, clients need client-specific training processes and/or different formats of loss functions to meet their resource limitations, data properties and learning goals, all of which can be diverse as discussed in Section 1. Formally speaking, for the m-th client, the local training dataset $\mathcal{D}_m$ might correspond to client-specific feature space $\mathcal{X}_m$ and label space $\mathcal{Y}_m$, which can lead to sub-optimal performance of the global model h_θ or even makes it unusable. To tackle this, the client could (1) maintain a local model h_{θ_m} with personalized parameters θ_m (i.e., personalization) and/or (2) minimize the local loss function F_m (i.e., multiple learning goals), while only sharing parts of the models with others for federal training. Therefore, the loss function in Equation (1) can be extended as:

$$\mathcal{L}' = \frac{1}{n}\sum_{m=1}^{M}\sum_{(x_i^{(m)},y_i^{(m)})\in\mathcal{D}_m} F_m\left(h_{\theta_m}(x_i^{(m)}), y_i^{(m)}\right). \quad (2)$$

Note that there exists some shared parameters among clients, i.e., $\bigcap_{m=1}^{M}\theta_m \neq \emptyset$, and all the clients collaboratively learn $\theta_1, \theta_2, \ldots, \theta_M$ to jointly minimize $\mathcal{L}'$.

Benefited from the event-driven architecture, FederatedScope provides users with flexible expressiveness to describe the behavior of an individual participant from its own perspective, which is crucial for handling the heterogeneity of FL via allowing the differences among participants. In this section, we present how FederatedScope supports such differences among participants for handling the heterogeneity of FL through the following two ways.

3.4.1 Personalized training behaviors. As discussed by previous work [15, 71], the heterogeneity of FL might hurt the model performance for some clients and lead to the sub-optimal performance when sharing the same global model among all participants, such as vanilla FedAvg [57], which motivates the study of personalized federated algorithms [48, 51, 56, 70]. Specifically, personalized federated algorithms are proposed to apply client-specific local training courses based on their private data, including client-wise training configuration, sub-modules, global-local fusing weights, etc. Therefore, users are expected to describe diverse behaviors of clients to develop personalized federated algorithms, which might be rather complicated and inconvenient when using a procedural programming paradigm since lots of effort is put into sequentially coordinating and describing the participants.

With the event-driven architecture, FederatedScope allows users to describe the behaviors of participants independently, which provides great flexibility to develop new personalization algorithms. Users are able but not limited to (1) specify the training configurations, such as local training steps and learning rate, for an individual client; (2) define new events related to new types of exchanged messages and/or events related to customized conditions to apply personalization algorithms (e.g., performance_drop); (3) add personalized behaviors into handlers that are triggered for local training, such as fusing the received global model with local models before performing local training. In most cases, such customization can be inherited from the general training behaviors and only need to focus on the differences. Considering that clients might have different privacy protection requirements, some privacy protection techniques can be adopted. For example, clients might choose to inject noise into the model parameters before sharing them. More details of the privacy protection of messages can be found in Section 4.1.

We provide several representative personalized federated algorithms [15] in FederatedScope for handling the heterogeneity in FL, including pFedMe [70], FedBN [51], FedEM [56], and Ditto [48]. These built-in algorithms serve as examples for showing how to easily and flexibly develop new personalized federated algorithms, and can also be conveniently adopted via configuring by users in real-world applications.

3.4.2 Multiple Learning Goals. Note that the scope of FL also covers the scenarios where participants learn common knowledge while optimizing different learning goals [56, 69, 85, 90]. The participants of an FL course reach a consensus on what needs to be shared while keeping other learning parts private, especially in cross-silo scenarios. For example, several medical research institutes would like to collaboratively learn a graph neural network for capturing the common structure knowledge of molecules, but they will not disclose what is the usage of the learned structure knowledge. They might exchange the update of the graph convolution layers while maintaining the encode layers, readout layers, and headers (such as classifier) private. In this and more similar scenarios from model pre-training, it can be difficult or even intractable for users to develop with a procedural programming paradigm via defining the static computation graph of the FL course.

Fortunately, the event-driven design of FederatedScope makes it easy to express and implement the FL courses with multiple learning goals. Each participant owns its local model and private data, defines its computation graph, locally trains with private learning objective, and only exchanges messages of the shared layers with others through FL.

Currently, FederatedScope provides three representative scenarios of FL with multiple learning goals, including graph classification, molecular property inference, and natural language understanding (NLU). In the graph classification scenario, clients own different graph classification tasks and aim to collaboratively improve their own performance due to the limitation of available training data. In the molecular property inference scenario, different clients have different property inference goals, such as the solubility (regression task), the enzyme type (classification task), and the penetration (classification task), which leads to heterogeneity

in terms of task type. In the NLU scenario, clients are also heterogeneous in terms of task type, and they own different NLU tasks, including sentiment classification, reading compression, and sentence pair similarity prediction. Since the development of FL with multiple learning goals is still in the early stage, FederatedScope provides these scenarios to broaden the scope of FL applications and promote the development of innovative methods. More details of these scenarios of FL with multiple learning goals can be found in our open source repository [3].

In summary, FederatedScope allows users to describe participants' behaviors from their respective perspectives and thus provides flexibility in applying different training processes and learning goals to the participants to handle the heterogeneity of FL.

```
class CustomizedServer(Server):
    def customized_handler(args):
        Do sth. # Describe the operations for handling the event
    ... ...
    # Register the customized handlers for customized events
    registered_handlers = dict() # Expected type {event: handler}
    register(customized_event, customized_handler)
    ... ...
    if customized_event == True:
        # Call the corresponding customized_handler
        registered_handlers[customized_event](args)
```

Figure 3: Behaviors description with events and handlers.

3.5 Supporting Cross-backend FL

Motivated by the strong need from real-world applications, FederatedScope supports constructing cross-backend FL courses. For example, in an FL task, some of the involved clients are equipped with TensorFlow while others might run with PyTorch. Thanks to the event-driven architecture, FederatedScope can conveniently provide such functionality via a mechanism called *message translation*. Note that such support of cross-backend FL is different from those provided by the universal languages such as ONNX [7] and the existing FL platforms such as TFF [10].

Conceptually, ONNX and TFF adopt a global perspective of constructing an FL course, which implies that the complete computation graph is globally defined and shared among all participants. In order to make it compatible with different (versions of) machine learning backends on different clients, the global computation graph is serialized into platform-independent and language-independent representations, sent to the clients, and interpreted or compiled accordingly for different backends.

Message translation. FederatedScope, in contrast, gives each participant the right of describing the computation graph on its own (for the portion it takes charge of). Hence, participants can define the computation graph based on their running backends. Following a pre-defined consensus on the format of messages, the participants transform the messages, e.g., gradients and model parameters, generated from the local backends into the pre-defined backend-independent format, e.g., an array of pairs of parameters and values, before sharing them with others. This procedure is called *encoding*. For the other direction, once an encoded message is received, the participant parses the message, e.g., the above array, into backend-dependent tensors in its own computation graph and backend, which is called the *decoding* procedure.

The encoding and decoding procedures are abstracted as two special programming interfaces in FederatedScope with default implementations; they can also be customized for each participant based on its backend and the FL algorithm to be deployed. FederatedScope provides several examples of constructing cross-backend federated learning [2].

In supporting cross-backend FL, the advantage of FederatedScope is two-fold: (1) FederatedScope provides more flexibility to handle the heterogeneity of FL than other platforms that adopt a global perspective since each participant has the right to declare its computation graph independently. Specifically, the developer of each participant can focus on expressing its own computation graph, such as client-specific embedding layers and output layers, adapting to its input instance and task. There is no need to declare a super graph (i.e., the global perspective) and care about how to distribute it, reducing the implementation difficulty. (2) FederatedScope follows the principle of information minimization, where participants only need to achieve a consensus on the format of messages and exchange necessary information. Thus, the exchanged model parameters will not leak the whole model architecture, the local training algorithm, or the personalization-related operators to other participants, which would otherwise be inferable from the global computation graph of ONNX and TFF. When such information leakage happens, malicious participants benefit from it because they can conduct a white-box attack rather than the more challenging black-box one in FederatedScope. We will talk more about privacy attacks in Section 4.1.

3.6 Usage of FederatedScope

In this section, we give a full example of how to set up an FL course, so that users can gain a clear and vivid understanding of FederatedScope. At a high level, users should define a series of events and their corresponding handlers, which characterize the behaviors of participants. As shown in Figure 3, the handlers are expressed as callable functions and bound to the corresponding events with a register mechanism. When an event happens, the corresponding handler will be called to handle it. The example is as follows:

Example 3.1. Consider that a server and several clients would like to construct an FL course and they agree to exchange certain model parameters during the training process.

For clients, the event related to message passing is "receiving_models", and the corresponding handler can be "*train the received global models based on local data, and then return the model updates*". The local training process is executed by a *Trainer* object held by the client. As illustrated in Figure 4, the trainer encapsulates the training details, entirely decoupled from the client's behaviors. Hence, the training process can be described as those of the centralized learning case, and the trainer can be flexibly extended with fancy optimizers, regularizers, personalized algorithms, etc. Such a design makes it easy for user customizations.

For the server, the event related to message passing is "receiving_updates" and the corresponding handler can be "*save the*

```
class Client(object):
    trainer = CustomizedTrainer(args)
    ... ...
    def handler_for_receiving_models(args):
        # Perform local training when receiving the global models
        model_update = trainer.train(args.model, args.data)
        send(message=model_update, receiver=server)
```

```
class CustomizedTrainer(Trainer):
    ... ...
    # Describe training behaviors (same as centralized training)
    def train(received_models, data):
        # Personalized algorithms might be applied here
        local_model = update_from_global_models(received_models)
        preds = local_model.forward(data.x)
        args = [optimizer, loss_function, regularizer, ...]
        model_updates = local_model.backward(data.y, preds, args)
        return model_updates
```

Figure 4: The training behaviors and clients are decoupled for supporting flexible customization.

model updates, and check the aggregation condition", which requires another event related to condition checking. For the synchronous training strategy, such event can be "all_received" and the corresponding handler will be "*perform federated aggregation, and broadcast the updated global models*". For the asynchronous training strategies, the event "all_received" can be replaced with "goal_achieved" or "time_up", which adds flexible behaviors during sampling clients or performing aggregation (More details can be found in Section 3.3). The federated aggregation is executed by an aggregator, which is also decoupled with the server for flexibly supporting various state-of-the-art (SOTA) aggregation algorithms, such as FedOpt [5], FedNova [77], FedProx [50], etc.

Note that when events such as "all_received" or "goal_achieved" happens, the clients would receive the up-to-date global models after the server performs federated aggregation, which naturally causes the following event "receiving_models" and triggers the handlers for performing a new round of local training. In this way, although we have not explicitly declared a sequential training process, the events happen in the intended logical order to trigger the corresponding handlers, which can precisely express the FL procedure and promote modularization. Further, events such as "maximum_iterations_reached" or "early_stopped" can be adopted to specify when the FL courses should be terminated. △

With such event-driven architecture, FederatedScope allows users to use existing or add new <event, handler> pairs for flexible customization, rather than inserting the new behaviors into the sequential FL course carefully as those in the procedural programming paradigm. For example, by simply changing the event "all_received" to other events related to condition checking such as "goal_achieved", users can conveniently apply asynchronous training strategies. Users also can add some new events related to message passing to enable the heterogeneous information exchange, such as node embeddings in graph federated learning [85] and encrypted results in cross-silo federated learning [32].

The details of the adopted algorithms in trainer and aggregator are decoupled with the behaviors of participants. Therefore, when users develop their own trainer/aggregator with FederatedScope, they only need to care about the details of training/aggregating algorithms. For example, users are expected to implement several basic interfaces of trainers, including train, evaluation, update model, etc., which is the same as those in centralized training and serves as "must-do" items. For the aggregator, which takes the received messages as inputs and returns the aggregated results, users only need to implement how to aggregate.

Programming Interfaces and Completeness Checking. FederatedScope provides base classes to aware users of the necessary interfaces for an FL course, such as BaseTrainer and in BaseWorker. These base classes can be used to check the completeness of the defined FL courses, since an "Not Implementation Error" would be raised to abort the execution if users fail to implement the necessary interfaces. With the base classes, FederatedScope provides rich implementation of existing FL algorithms. Therefore users can inherit the provided implementation and focus on the development of new functions and algorithms, which also ensures the completeness of FL courses. Besides, FederatedScope provides a completeness checking mechanism to generate a directed graph to verify the flow of message transmission in the constructed FL course.

Robustness Against Malicious Participants. To defend malicious participants and make the system more robust, some Byzantine fault tolerance algorithms are provided in FederatedScope. For example, we can apply the Krum [9] aggregation rule in federated aggregation. Note that these Byzantine fault tolerance algorithms can be regarded as the aggregation behaviors of server and implemented in the aggregator, which is decoupled with other behaviors to make it flexible and extendable for users to develop their own fault tolerance algorithms.

4 IMPORTANT PLUG-IN COMPONENTS

In this section, we present several important plug-in components in FederatedScope for convenient usage. These components provide functionalities including privacy protection, attack simulation, and auto-tuning, all of which are tightly coupled with the design of FederatedScope and serve as plug-ins.

4.1 Behavior Plug-In: Privacy Protection

Real-world FL applications might prefer different privacy protection algorithms due to their diversity in types of private information, protection strengths, computation and communication resources, etc., which motivates us to provide various privacy protection algorithms in FederatedScope.

With the design of FederatedScope, privacy protection algorithms can be implemented as behavior plug-ins, which indicates that the privacy protection algorithms bring new behaviors of participants. For example, before the participants share messages, the encryption algorithms might be applied on the messages, or the messages would be partitioned into several frames, or certain noise can be injected into the messages. These behaviors have been predefined in FederatedScope (so-called the behavior plug-in), and can be easily called to protect privacy via simple configuration.

Specifically, we implement a widely-used homomorphic encryption algorithm Paillier [65] and apply it in a cross-silo FL task [32];

```
class Client(object):
    def handler_for_receiving_models(args):
        ... ...
        if config.inject_noise_before_sharing == True:
            # Inject certain noise before sharing the message
            args = [noise_distribution, budget, ...]
            protected_messages = add_noise(messages, args)
            send(message=protected_messages, receiver=server)
        else:
            send(message=messages, receiver=server)
```

Figure 5: Behavior plug-in: injecting noise.

```
class Fed_Runner(object):
    ... ...
    def setup_client(config):
        if config.is_malicious == True:
            # Instantiate a malicious client with attack behavior
            client = MaliciousClient(attack_algorithms, args)
        else:
            # Instantiate a normal client
            client = Client(args)
        client.join_in_FL_course()
```

Figure 6: Participant plug-in: malicious client.

and we develop a secret sharing mechanism for FedAvg. These provided examples demonstrate how to apply privacy protection algorithms with FederatedScope. Furthermore, to satisfy the heterogeneity in privacy protection strengths, we provide tunable modules for applying Differential Privacy (DP) in FL, which has been a popular technique for privacy protection and has achieved great success in database and FL applications [22, 23, 74, 81]. An example is illustrated in Figure 5, from which we can see that users can utilize the configuration to modify the client's behavior: injecting certain noise into the messages before sharing. Users can combine different behaviors together to implement fancy DP algorithms such as NbAFL [81]. Note that to achieve a theoretical guarantee of privacy protection, users still need to specify some necessary settings according to their own data and tasks, including the noise distribution [24, 66] and privacy budget allocation [52, 55, 78].

4.2 Participant Plug-In: Attack Simulation

Attacks, growing along with the development of FL, are important for users to verify the availability and the privacy protection strength of their FL systems and algorithms. Typical attacks include privacy attack and performance attack: the former aims to steal the information related to clients' private data, while the latter aims to intentionally guide the learned model to misclassify a specific subset of data for malicious purposes such as back-door. However, most of the existing FL platforms ignore such an important functional component.

Note that it is non-trivial to provide attack simulation in an FL platform, since the diversity of privacy and performance attacks brings challenges to the platform's flexibility and extensibility. Benefited from the design of FederatedScope, the behaviors of malicious participants can be expressed independently, thus the attack simulation can be implemented as the participant plug-in in FederatedScope. To be more specific, as shown in Figure 6, users can conveniently choose some of the participants to become malicious clients via configuring, and attack algorithms can be added to their own trainers. These malicious clients are able to collect or inject certain messages among victims, and further recover or infer the target information accordingly. The simulated attacks provided in FederatedScope can be used to verify the privacy protection strength of their FL systems and algorithms. For example, when users develop a new FL algorithm, they want to know the protection level of the proposed algorithm from some perspectives, such as whether the dataset properties or private training samples would be inferred by attacks. They can use several state-of-the-art attack algorithms, which have been provided in FederatedScope for convenient usage, to check the privacy protection strength of their FL algorithms, and enhance the privacy protection strength if necessary according to the results of simulated attacks.

FederatedScope provides rich types of attack. For privacy attack, FederatedScope provides the implementation of the following algorithms: (i) Gradient inversion attack [62] for membership inference; (ii) PIA [60] for property inference attack; (iii) DMU-GAN [35] for class representative attack; (iv) DLG [95], iDLG [94], GRADINV [27] for training data/label inference attack. In terms of performance attack, FederatedScope currently focuses on the backdoor attack, a representative type of performance attack, whose objective is to mislead the model to classify some selected samples to the attacker-specified class. The implementations of SOTA backdoor attacks include: (i) Edge-case backdoor attacks [76], BadNets [29], Blended [16], WaNet [64], NARCISSUS [91], which perform back-door attack by poisoning the dataset; (ii) Neurotoxin [93] and DBA [83], which perform back-door attack by poisoning the model.

4.3 Manager Plug-In: Auto-tuning

FL algorithms generally expose hyperparameters that can significantly affect their performance. Without suitable configurations, users cannot manage their FL applications well. Hyperparameter optimization (HPO) methods, both traditional methods (e.g., Bayesian optimization [67] and multi-fidelity methods [4, 6, 25, 47]) and Federated-HPO methods [20, 39] (denoting very recent ones that deliberately take the FL setting into account) can help users manage FL applications by automatically seeking suitable hyperparameter configurations.

Therefore, in FederatedScope, we provide an auto-tuning plug-in, which incorporates various HPO methods. Conceptually, Bayesian optimization, multi-fidelity, and Federated-HPO methods treat a complete FL course, a few FL rounds, and client-wise local update procedures as black-box functions to be evaluated, respectively. FederatedScope provides a unified interface to manage the underlying FL procedure in various granularities so that different HPO methods can interplay with their corresponding black-box functions. This unification is nontrivial for the last case, where we leverage our event-driven architecture to achieve the client-wise exploration of Federated-HPO methods. When they are plugged in, the exchanged messages are extended with HPO-related samples/models/feedback, and the participants would handle them with extended behaviors accordingly.

```
class Server(object):
    def handler_for_receiving_updates(args):
        ... ...
        if config.apply_fedex == True:
            # Choose hyperparameters for the client
            cfg = sample_cfg(cfg_candidates, args.hpo_feedback)
        # Continue to handling the message accordingly
        ... ...
```

```
class Client(object):
    def handler_for_receiving_models(args):
        ... ...
        if config.apply_fedex == True:
            # Apply the received hyperparameters
            trainer.apply_cfg(args.received_config)
        # Continue to handling the message accordingly
        ... ...
```

Figure 7: Manager plug-in: re-specify configuration.

For Bayesian optimization methods, we showcase applying various open-sourced HPO packages to interact with FederatedScope. Each time they propose a specific configuration, FederatedScope executes an FL course accordingly and returns a specified metric (e.g., validation loss) as the function's output. As for multi-fidelity methods, we have implemented Hyperband [47] and PBT [46]. Specifically, FederatedScope can export the snapshot of a training course to a corresponding checkpoint, from which another training course can restore. With such a checkpoint mechanism, these multi-fidelity methods can evaluate the configurations that have survived previous low-fidelity comparisons by restoring from the last checkpoints rather than learning from scratch.

Furthermore, FederatedScope provides FedEx [39] as an exemplary implementation of Federated-HPO methods. Specifically, once FedEx is plugged in, we sample configurations for each client independently in each FL round. Then each client re-specifies its native configuration and conducts local updates accordingly, as shown in Figure 7. Finally, the feedback is aggregated to update the policies responsible for determining the optimal configuration(s).

In summary, the auto-tuning plug-in can manage FL applications in various granularities. Traditional HPO methods interplay with FederatedScope by configuring and running one or more complete FL rounds, while Federated-HPO methods explore client-wise configurations concurrently in a single FL round. With flexibility provided by the event-driven architecture, we have implemented these HPO methods in a unified way [80], and novel HPO methods can be easily developed and contributed to FederatedScope.

5 EXPERIMENTS

5.1 DataZoo and ModelZoo

For convenient usage, we collect and preprocess ten widely-used datasets from various FL application scenarios, including computer vision datasets (FEMNIST [19], CelebA [54] and CIFAR-10 [43]), natural language processing datasets (Shakespeare [57], Twitter [28] and Reddit [58]) from LEAF [12], and graph learning datasets (DBLP [73], Ciao [72] and MultiTask [85]) from FederatedScope-GNN (FS-G) [79]. The statistics of these datasets can be found in the full version [1]. Meanwhile, we provide off-the-shelf neural network models via our ModelZoo, which includes widely-adopted model architectures, such as ConvNet [45] and VGG [68] for computer vision tasks, BERT [21] and LSTM [36] for natural language processing tasks, and various GNNs [18, 30, 40, 75, 87] for graph learning. Such ModelZoo allows users to conveniently develop various trainers for clients.

5.2 Experiment Settings

Here we conduct a series of experiments with FederatedScope on three representative datasets as follows:

FEMNIST. FEMNIST consists of 805,263 handwritten digits in 62 classes, which are partitioned into 3,597 clients according to the writers. With FL, a CNN with two convolutional layers is trained for image classification task on this dataset.

CIFAR-10. As suggested by previous studies [37], we partition the dataset into 1,000 clients with a Dirichlet distribution, and federally train a CNN with two convolutional layers for image classification.

Twitter. We sample a subset from Twitter, which consists of 6,602 twitter users' 16,077 texts. Each twitter user can be regarded as a client for constructing an FL course. Following previous study [12], we embed the texts with a bag-of-words model [33] and collaboratively train a logistic regression model for sentiment analysis.

More implementation details can be found in the full version [1].

5.3 Results and Analysis

5.3.1 Asynchronous Federated Learning. We first conduct experiments to compare the performance of applying synchronous and asynchronous training strategies in FL.

Virtual Timestamp. Following the best practice in prior FL works [44], we conduct the experiments by simulation while tracking the execution time with virtual timestamps. Specifically, the server begins to broadcast messages containing initial model parameters at timestamp 0. Then each client sends updates back with a timestamp as the received one plus the execution time of local computation and communication estimated by FedScale [44]. The server handles the received messages in the order of their timestamps and lets the next broadcast inherit the timestamp from the message that triggers it, assuming the time cost of the server is negligible. Along with an FL course, we record the performance of the global model with respect to such virtual timestamps.

Baselines. We implement FedAvg with two synchronous training strategies including *Sync-vanilla* (i.e., the vanilla synchronous strategy) and *Sync-OS* (i.e., the synchronous strategy with over-selection mechanism [10]). As Sync-OS is originally proposed and implemented in FedScale [44], we also adapt it for our experiments and report its performance (denoted as *Sync-OS (FedScale)*) for correctness verification.

For asynchronous FL, we instantiate different asynchronous behaviors discussed in Section 3.3, and different strategies are named in the format of *Async-AdoptedEvent-BroadcastManner-SampleStrategy*. For example, *Async-Goal-Rece-Unif* denotes that this strategy adopts the event "goal_achieved", the *after receiving* broadcasting manner and the uniform sampling strategies for asynchronous FL, which can be regarded as the implementation of FedBuff [63]; and *Async-Time-Aggr-Group* denotes we adopt the event

Table 1: The comparison between applying synchronous and asynchronous training strategies in federated learning, in terms of the virtual time cost (hours) to achieve the targeted test accuracy.

Dataset (Target Acc.)	Sync.			Async.			
	Vanilla	OS	OS (FedScale)	Goal-Aggr-Unif	Goal-Rece-Unif	Time-Aggr-Unif	Goal-Aggr-Group
FEMNIST (85%)	61.46	27.34 2.25×	28.78 2.14×	11.29 5.44×	11.36 5.41×	11.70 5.25×	10.42 5.90×
CIFAR-10 (70%)	66.99	26.42 2.54×	28.98 2.31×	7.73 8.67×	7.98 8.39×	8.87 7.55×	7.54 8.88×
Twitter (69%)	9.41	3.84 2.45×	4.14 2.27×	0.78 12.06×	0.64 14.70×	0.50 18.82×	0.65 14.48×

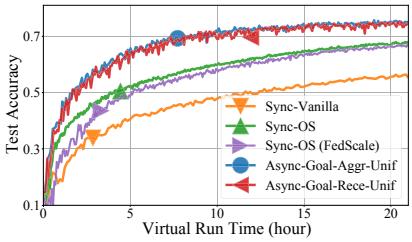

Figure 8: The comparison between synchronous and asynchronous strategies.

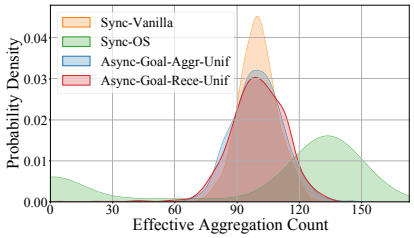

Figure 9: The distributions of the aggregated count of the clients.

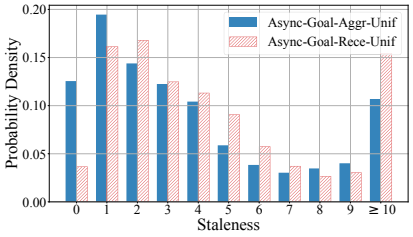

Figure 10: The distributions of the staleness in asynchronous strategies.

"time_up", the *after aggregating* broadcasting manner and a group sampling strategy (the client would be grouply sampled according to their responsiveness [14]).

Analysis. We adopt the virtual time cost (hours) to achieve the targeted test accuracy as the performance metric for comparing synchronous and asynchronous FL. The experimental results are shown in Table 1, from which we can observe that asynchronous training strategies achieve significant efficiency improvements (5.25×~18.82×) compared to the vanilla synchronous training strategy on all the benchmark datasets. Meanwhile, we plot the learning curves in Figure 8. Due to the space limitation, we only show some asynchronous training strategies on CIFAR-10 dataset and omit other similar results. From Figure 8, we can observe the existence of noticeable gaps between synchronous and asynchronous training strategies for a long time during the training process. These experimental results are consistent with previous studies [38, 84] and confirm that the asynchronous training strategies provided in FederatedScope can significantly improve the training efficiency while achieving competitive model performance.

Both our implementation *Sync-OS* and the original implementation in FedScale show that applying over-selection mechanism in synchronous FL can improve the efficiency to some degree. However, it might cause unfairness among participants and then lead to model bias, as demonstrated in Figure 9. From the figure we can observe that when applying over-selection mechanism *Sync-OS*, some clients never contribute to the federated aggregation, i.e., $\Pr[\text{effective\_aggregation\_count} = 0] > 0$. The reason is that these clients need more computation or communication time, and thus their feedback would always be dropped since the server has finished the federated aggregation with the feedback from those clients having faster response speeds. In other words, these clients always become the victims among the over-selected clients, which results in unfairness among participants, and then causes the learned models to bias towards those clients with fast response speeds. In contrast, the asynchronous learning strategies provided in FederatedScope can improve the efficiency without introducing such unfairness and model bias, due to the fact that staled feedback would be tolerated in the federated aggregation. Hence the distribution of effective aggregation count of asynchronous learning strategies plotted in Figure 9 is more concentrated and similar to that of the vanilla synchronous training strategy.

Further, in Figure 10, we illustrate the characteristics of different asynchronous training strategies in terms of staleness (i.e., the version difference between the up-to-date global model and the model used for local training) of the updates when performing federated aggregation. By comparing *Async-Goal-Aggr-Unif* and *Async-Goal-Rece-Unif*, we can see that *after aggregating* broadcasting manner causes less staleness than *after receiving*. It implies that *after aggregating* is more suitable for those FL tasks with a low staleness toleration threshold, but such a broadcasting manner requires more available bandwidths at the server since multiple messages are sent out at the same time.

5.3.2 Personalization. To demonstrate how personalization can handle the heterogeneity among participants, we compare FedAvg [57] with several built-in SOTA personalized FL algorithms, including FedBN [51], FedEM [56], pFedMe [70] and Ditto [48].

The experimental results are illustrated in Figure 11, which shows the client-wise test accuracies on FEMNIST dataset. We can observe that the average accuracy (denoted as the red dots) and the 90% quantile accuracy (denoted as the red horizontal lines) of vanilla FedAvg are both significantly lower than those of personalized FL algorithms. This indicates that applying personalized FL algorithms can improve the client-wise performance, also covering the bottom clients, and then lead to a better overall performance. Besides, in terms of the standard deviation among the client-wise accuracy (shown as σ at the top of the figure), personalized FL

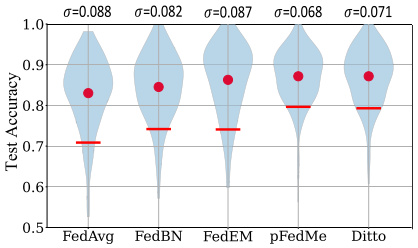

Figure 11: Client-wise test accuracy on FEMNIST dataset.

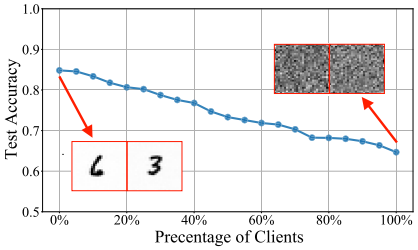

Figure 12: Accuracy w.r.t. varying protection strength and recovered images.

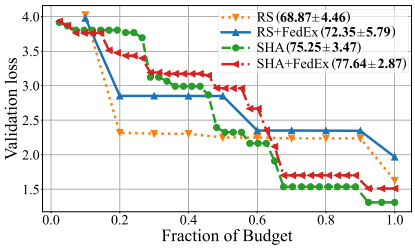

Figure 13: Best-seen validation loss over time on FEMNIST dataset.

algorithms can reduce the performance differences to some degree, which confirms the advantages of enabling personalization behaviors for handling the heterogeneity among participants in real-world FL applications.

Personalized federated learning algorithms might need different computation and computation resources compared to vanilla FedAvg [57]. The computation and communication costs in each training round are determined by the adopted algorithms. Take the comparisons between vanilla FedAvg and two representative Personalized federated learning algorithms FedBN [51] and Ditto [48] as examples, in each training round [15], (i) FedBN needs the same computation but fewer communication costs, since it proposes to not share the parameters of BacthNorm layer; and (ii) Ditto needs the same communication but more local computation costs, since it suggests to train the local models additionally. Further, from the perspective of an FL course, i.e., iteratively performing the FL training rounds until termination, the communication and computation costs depend on the convergence of learned models.

5.3.3 Privacy Protection and Attack. We conduct an experiment to show the effect of applying privacy protection algorithms provided in FederatedScope. We take DP as an example, and study its effect on the utility of learned model and the effectiveness in defending against privacy attack. Specifically, we train a ConvNet2 model on FEMNIST, and randomly choose some of the clients to inject Gaussian noise into the returned model updates to strengthen their privacy. We construct multiple FL courses with respect to varying the percentage of clients that injects noise, changing from 0% to 100%, and plot the performance of the learned models in Figure 12. From this figure we can observe that as more and more clients choose to inject the noise into the returned model updates, the test accuracy achieved by the learned global model decreases gradually, from 84% to 65%, which shows the trade-off between the privacy protection strength and model utility.

Moreover, we apply the DLG algorithm [95] implemented in FederatedScope to conduct privacy attack, aiming to reconstruct private training data of other users. As shown on the left-hand side of Figure 12, the reconstructed images from the clients who have not injected noises are clear and the privacy attacker successfully recovers clients' training data to the extent that the groundtruth digits are exposed. On the right-hand side of the figure, we plot the reconstructed images from those clients injecting noises, which confirms the effectiveness of the privacy protection provided by DP since the attacker fails to recover meaningful information.

5.3.4 Auto-Tuning. As mentioned in Section 4.3, we have implemented several HPO methods in FederatedScope, which enables users to auto-tune hyperparameters of FL courses. Here we experimentally compare some representative HPO methods, including random search (RS) [8], successive halving algorithm (SHA) [47] and recently proposed Federated-HPO method FedEx [39], by applying them to optimize hyperparameters of FedAvg on FEMNIST dataset. We follow the protocol used in FedHPO-B [80], where RS and SHA try configurations one by one, and FedEx wrapped by RS/SHA manages the search procedure in a fine-grained granularity to explore hyperparameter space concurrently.

We present the results in Figure 13, where the best-seen validation loss is depicted, and the test accuracy of the searched optimal configuration is reported in the legend. The best-seen validation losses of wrapped FedEx decrease slower than their corresponding wrappers, where such a poorer regret seems to indicate poorer searched hyperparameter configurations. However, their searched configurations' test accuracies are remarkably better than their wrappers, implying the superiority of managing the search procedure in a fine-grained granularity.

6 CONCLUSIONS

In this paper, we introduce FederatedScope, a novel federated learning platform, to provide users with great supports for various FL development and deployment. Towards both convenient usage and flexible customization, FederatedScope exploits an event-driven architecture to frame an FL course into <events, handlers> pairs so that users can describe participants' behaviors from their respective perspectives. Such an event-driven design makes FederatedScope suitable for handling various types of heterogeneity in FL, due to the advantages that (i) FederatedScope enables participants to exchange rich types of messages, express diverse training behaviors, and optimize different learning goals, and (ii) FederatedScope offers rich condition checking events to support various coordinations and corporations among participants, such as different asynchronous training strategies. Further, the design of FederatedScope allows us to conveniently implement and provide several important plug-in components, such as privacy protection, attack simulation, and auto-tuning, which are indispensable for practical usage. We have released FederatedScope to help researchers and developers quickly get started, develop new FL algorithms, and build new FL applications, with the goal of promoting and accelerating the progress of FL.

REFERENCES

[1] Full version of paper *FederatedScope: A Flexible Federated Learning Platform for Heterogeneity.* https://arxiv.org/abs/2204.05011

[2] The examples of cross-backend FL in FederatedScope. https://github.com/alibaba/FederatedScope/tree/master/federatedscope/cross_backends

[3] The examples of multiple learning goals FL in FederatedScope. https://github.com/alibaba/FederatedScope/tree/master/benchmark/B-FHTL

[4] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'19)*. 2623–2631.

[5] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. 2020. FedOpt: Towards Communication Efficiency and Privacy Preservation in Federated Learning. *Applied Sciences* 10, 8 (2020).

[6] Noor Awad, Neeratyoy Mallik, and Frank Hutter. 2021. DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization. In *Proc. of the International Jont Conference on Artifical Intelligence (IJCAI'21)*. 2147–2153.

[7] Junjie Bai, Fang Lu, Ke Zhang, et al. 2019. ONNX: Open Neural Network Exchange. https://github.com/onnx/onnx.

[8] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, 2 (2012), 281–305.

[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'17)*.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems* 1, 0 (2019), 374–388.

[11] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. 1175–1191.

[12] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[13] Mohak Chadha, Anshul Jindal, and Michael Gerndt. 2020. Towards federated learning using faas fabric. In *Proc. of the the 2020 Sixth International Workshop on Serverless Computing*. 49–54.

[14] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. 2021. FedAT: a high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (ResilientFL'21)*. 1–16.

[15] Daoyuan Chen, Dawei Gao, Weirui Kuang, Yaliang Li, and Bolin Ding. 2022. pFL-Bench: A Comprehensive Benchmark for Personalized Federated Learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[16] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[17] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous online federated learning for edge devices with non-iid data. In *Proc. of the IEEE International Conference on Big Data (BigData'20)*. 15–24.

[18] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *Proc. of the International Conference on Learning Representations (ICLR'21)*.

[19] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. 2017. EMNIST: Extending MNIST to handwritten letters. In *Proc. of the International Joint Conference on Neural Networks (IJCNN'17)*. 2921–2926.

[20] Zhongxiang Dai, Bryan Kian Hsiang Low, and Patrick Jaillet. 2020. Federated Bayesian Optimization via Thompson Sampling. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'20)*.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT'19)*. 4171–4186.

[22] Bolin Ding, Marianne Winslett, Jiawei Han, and Zhenhui Li. 2011. Differentially private data cubes: optimizing noise sources and consistency. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*. 217–228.

[23] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *Proc. of the International Conference on Theory and Applications of Models of Computation (TAMC'08)*. 1–19.

[24] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[25] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proc. of the International Conference on Machine Learning (ICML'18)*. 1437–1446.

[26] Wenjing Fang, Derun Zhao, Jin Tan, Chaochao Chen, Chaofan Yu, Li Wang, Lei Wang, Jun Zhou, and Benyu Zhang. 2021. Large-scale Secure XGB for Vertical Federated Learning. In *Proc. of the ACM Conference on Information and Knowledge Management (CIKM'21)*. 443–452.

[27] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting gradients-how easy is it to break privacy in federated learning?. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'20)*. 16937–16947.

[28] Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford* 1, 12 (2009), 2009.

[29] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access* 7 (2019), 47230–47244.

[30] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'17)*. 1024–1034.

[31] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[32] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017).

[33] Zellig S. Harris. 1954. Distributional Structure. *WORD* 10, 2-3 (1954), 146–162.

[34] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A Research Library and Benchmark for Federated Machine Learning. *arXiv preprint arXiv:2007.13518* (2020).

[35] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 603–618.

[36] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[37] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. 2019. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335* (2019).

[38] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems* 4, 0 (2022).

[39] Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina F Balcan, Virginia Smith, and Ameet Talwalkar. 2021. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'21)*. 19184–19197.

[40] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proc. of the International Conference on Learning Representations (ICLR'17)*.

[41] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).

[42] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proc. of the NetDB workshop*. 1–7.

[43] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. *Technical report, University of Toronto* (2009).

[44] Fan Lai, Yinwei Dai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. FedScale: Benchmarking model and system performance of federated learning. In *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*. 1–3.

[45] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[46] Ang Li, Ola Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. 2019. A generalized framework for population based training. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'19)*. 1791–1799.

[47] Lisha Li, Kevin G Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: Bandit-Based Configuration Evaluation for Hyperparameter Optimization. In *Proc. of the International Conference on Learning Representations (ICLR'17)*.

[48] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. 2021. Ditto: Fair and robust federated learning through personalization. In *Proc. of the International Conference on Machine Learning (ICML'21)*. 6357–6368.

[49] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[50] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems* 2 (2020), 429–450.

[51] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. 2021. Fedbn: Federated learning on non-iid features via local batch normalization. In *Proc. of the International Conference on Learning Representations (ICLR'21)*.

[52] Zitao Li, Bolin Ding, Ce Zhang, Ninghui Li, and Jingren Zhou. 2021. Federated matrix factorization with privacy guarantee. *PVLDB* 15, 4 (2021), 900–913.

[53] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'15)*. 2737–2745.

[54] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep learning face attributes in the wild. In *Proc. of the IEEE international conference on computer vision (ICCV'15)*. 3730–3738.

[55] Tao Luo, Mingen Pan, Pierre Tholoniat, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. 2021. Privacy budget scheduling. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*. 55–74.

[56] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. 2021. Federated Multi-Task Learning under a Mixture of Distributions. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'21)*. 15434–15447.

[57] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proc. of the Artificial intelligence and statistics (AISTATS'17)*. 1273–1282.

[58] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *Proc. of the International Conference on Learning Representations (ICLR'18)*.

[59] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *Proc. of the IEEE Symposium on Security and Privacy (SP'19)*. 691–706.

[60] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *Proc. of the IEEE Symposium on Security and Privacy (SP'19)*. 691–706.

[61] Brenda M Michelson. 2006. Event-driven architecture overview. *Patricia Seybold Group* 2, 12 (2006), 10–1571.

[62] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *Proc. of the IEEE Symposium on Security and Privacy (SP'19)*. 739–753.

[63] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated learning with buffered asynchronous aggregation. In *Proc. of the Artificial intelligence and statistics (AISTATS'22)*. 3581–3607.

[64] Tuan Anh Nguyen and Anh Tuan Tran. 2021. WaNet - Imperceptible Warping-based Backdoor Attack. In *Proc. of the International Conference on Learning Representations (ICLR'21)*.

[65] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of the international conference on the theory and applications of cryptographic techniques (EUROCRYPT'19)*. 223–238.

[66] NhatHai Phan, Xintao Wu, Han Hu, and Dejing Dou. 2017. Adaptive laplace mechanism: Differential privacy preservation in deep learning. In *Proc. of the IEEE international conference on data mining (ICDM'17)*. 385–394.

[67] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.

[68] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. of the International Conference on Learning Representations (ICLR'15)*.

[69] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'17)*. 4424–4434.

[70] Canh T Dinh, Nguyen Tran, and Josh Nguyen. 2020. Personalized federated learning with moreau envelopes. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'20)*. 21394–21405.

[71] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. 2021. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems* PP (2021).

[72] Jiliang Tang, Huiji Gao, and Huan Liu. 2012. mTrust: Discerning multi-faceted trust in a connected world. In *Proc. of the ACM International Conference on Web Search and Data Mining (WSDM'12)*. 93–102.

[73] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. 990–998.

[74] Aleksei Triastcyn and Boi Faltings. 2019. Federated learning with bayesian differential privacy. In *Proc. of the IEEE International Conference on Big Data (Big-Data'19)*. 2587–2596.

[75] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proc. of the International Conference on Learning Representations (ICLR'18)*.

[76] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris S. Papailiopoulos. 2020. Attack of the Tails: Yes, You Really Can Backdoor Federated Learning. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'20)*, Vol. 33. 16070–16084.

[77] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'20)*. 7611–7623.

[78] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. 2019. Answering multi-dimensional analytical queries under local differential privacy. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD'19)*. 159–176.

[79] Zhen Wang, Weirui Kuang, Yuexiang Xie, Liuyi Yao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. FederatedScope-GNN: Towards a Unified, Comprehensive and Efficient Package for Federated Graph Learning. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'22)*.

[80] Zhen Wang, Weirui Kuang, Ce Zhang, Bolin Ding, and Yaliang Li. 2022. FedHPO-B: A Benchmark Suite for Federated Hyperparameter Optimization. *arXiv preprint arXiv:2206.03966* (2022).

[81] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. 2020. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3454–3469.

[82] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen Jarvis. 2020. SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning With Low Overhead. *IEEE Trans. Comput.* 70, 5 (2020), 655–668.

[83] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2020. DBA: Distributed Backdoor Attacks against Federated Learning. In *Proc. of the International Conference on Learning Representations (ICLR'20)*.

[84] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).

[85] Han Xie, Jing Ma, Li Xiong, and Carl Yang. 2021. Federated graph classification over non-iid graphs. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'21)*. 18839–18852.

[86] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. 2021. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research* 5, 1 (2021), 1–19.

[87] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proc. of the International Conference on Learning Representations (ICLR'19)*.

[88] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2 (2019), 12:1–12:19.

[89] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. 2019. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 13, 3 (2019), 1–207.

[90] Liuyi Yao, Dawei Gao, Zhen Wang, Yuexiang Xie, Weirui Kuang, Daoyuan Chen, Haohui Wang, Chenhe Dong, Bolin Ding, and Yaliang Li. 2022. A Benchmark for Federated Hetero-Task Learning. *arXiv preprint arXiv:2206.03436* (2022).

[91] Yi Zeng, Minzhou Pan, Hoang Anh Just, Lingjuan Lyu, Meikang Qiu, and Ruoxi Jia. 2022. Narcissus: A Practical Clean-Label Backdoor Attack with Limited Information. *arXiv preprint arXiv:2204.05255* (2022).

[92] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-Aware Async-SGD for Distributed Deep Learning. In *Proc. of the International Joint Conference on Artifical Intelligence (IJCAI'16)*. 2350–2356.

[93] Zhengming Zhang, Ashwinee Panda, Linyue Song, Yaoqing Yang, Michael W. Mahoney, Joseph E. Gonzalez, Kannan Ramchandran, and Prateek Mittal. 2022. Neurotoxin: Durable Backdoors in Federated Learning. *arXiv preprint arXiv:2206.10341* (2022).

[94] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. iDLG: Improved Deep Leakage from Gradients. *arXiv preprint arXiv:2001.02610* (2020).

[95] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. In *Proc. of the Advances in Neural Information Processing Systems (NeurIPS'19)*. 14747–14756.

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