FedLab: A Flexible Federated Learning Framework

Dun Zeng1,2,†
Siqi Liang4,†
Xiangjing Hu3
Hui Wang2
Zenglin Xu3,2,⋆

1 University of Electronic Science and Technology of China
2 Peng Cheng Lab
3 Harbin Institute of Technology Shenzhen
4 Shenzhen Research Institute, The Chinese University of Hong Kong
† Denotes equal contribution
⋆ Corresponding author

Abstract

FedLab is a lightweight open-source framework for the simulation of federated learning. The design of FedLab focuses on federated learning algorithm effectiveness and communication efficiency. It allows customization on server optimization, client optimization, communication agreement, and communication compression. Also, FedLab is scalable in different deployment scenarios with different computation and communication resources. We hope FedLab could provide flexible APIs as well as reliable baseline implementations and relieve the burden of implementing novel approaches for researchers in the FL community. The source code, tutorial, and documentation can be found at https://github.com/SMILELab-FL/FedLab.

Keywords: distributed learning, federated learning, Python, PyTorch

1. Introduction

Federated learning (FL) is recently a burgeoning research area of machine learning (McMahan et al., 2017). It aims to protect individual data privacy in distributed machine learning applications, especially in finance (Byrd and Polychroniadou, 2020), smart healthcare, (Xu et al., 2021; Brisimi et al., 2018) and edge computing (Jiang et al., 2019; Meng et al., 2021). Unlike traditional data-centered distributed machine learning, participants in the FL settings utilize localized data to train local models and then leverage aggregation strategies with other participants to collaboratively acquire a global model without direct data-sharing.

The training scheme of most FL methods is generally composed of four steps in each round: 1. The server selects a subset of clients and broadcasts global information (e.g., global model parameters). 2. Each client updates the local model with a private dataset. 3. Clients upload local information (e.g., model parameters) to the server. 4. The server updates the global model based on collected information from clients.

We investigate recently proposed federated learning algorithms as shown in Table 1. Most of them improve effectiveness or efficiency by changing only one or several workflow
steps based on different motivations. This indicates that the implementations of many FL algorithms only require modification on several components of the common workflow, without the necessity of repetitive implementation on basic FL workflows.

To relieve the burden of implementing FL algorithms and to free researchers from the burden of the repetitive implementation of basic FL settings, we develop a highly customizable framework FedLab in this work. FedLab provides the necessary modules for FL simulation, including communication, model optimization, data partition, and other functional supports. Users can build an FL simulation environment with customized modules using FedLab in the way of playing with LEGO bricks.

2. Framework Overview

In this section, we introduce the overview of the proposed library, including the major features, the code usage pipelines, and a discussion of FedLab’s impacts on the FL community. For a better understanding, the architecture of FedLab is shown in Figure 1(a), and the functional supports are shown in Figure 1(b).

![Framework Architecture](a) Framework architecture

![Functional Supports](b) Functional supports

Figure 1: Overview of the FedLab framework

2.1 Features

**Communication.** For best compatibility with interfaces of PyTorch (Paszke et al., 2019), communication APIs of FedLab are built for tensor communication. We provide reliable send/recv functions for point-to-point communication, while all details of message packaging
FedLab: A Flexible Federated Learning Framework

and transmission are transparent to users. Users can use communication APIs provided by FedLab without knowing the information processing and transmission mechanism.

**Communication Compression.** Based on the point-to-point communication module, the communication compression algorithms could be easily implemented. A sender could conduct tensor compression before calling the `send` function. Then, the receiver could conduct the corresponding decompression procedure. Additionally, we implement the baseline quantification compressor (Alistarh et al., 2017) and sparsification (Shi et al., 2019) compressor in FL, as shown in *Compression Method* of Figure 1(b).

**Communication Agreement.** The flexibility of communication management in FedLab is given by `NetworkManager` module, which offers users customizable interfaces of communication agreement. Besides, we provide basic communication patterns, namely synchronous and asynchronous FL. Meanwhile, FedLab standardizes the implementation of FL communication agreements, and it is open to users’ customization needs. Users can define the communication flow between the server and clients for advanced algorithm development.

**Federated Optimization.** Trainer/ServerHandler in FedLab takes charge of FL optimization procedure on client/server respectively. Trainer manages local datasets and performs PyTorch training process; ServerHandler works as a computation backend of the server, taking charge of the model aggregation. To demonstrate the usage of Trainer, FedLab provides SGDClientTrainer as a standard implementation of Trainer for users, and it can be used as the default Trainer in many tasks. Additionally, FedLab provides reproduction of different algorithms for uses, as shown in the *FL Algorithm* part in Figure 1(b).

**Data Partition & Datasets.** The non-IID of data is a key challenge of FL, while realistic Non-IID datasets are not always accessible to researchers. Researchers tend to manually create Non-IID data partitions in the experiment environment (Caldas et al., 2018). For users’ convenience, FedLab offers DataPartitioner for FL data partition. A series of data partition schemes for both IID and Non-IID from different data distribution settings (Yurochkin et al., 2019; Acar et al., 2021; Caldas et al., 2018; Li et al., 2021a) are already provided, including more than 12 data partition schemes for 15 datasets, as shown in *Data Partition and Dataset* of Figure 1(b).

**Scalability.** To meet the requirements of simulation for different scales and simulation acceleration, we design three simulation schemes: Standalone, Cross-process, and Hierarchical. Standalone uses SerialClientTrainer, allowing one single process to simulate multiple clients sequentially with limited computing resources. Cross-process allows larger-scale FL simulation with multiple processes parallelly across computers in the same LAN. Each computer could simulate an arbitrary number of clients’ calculation tasks depending on hardware resources. Hierarchical provides a communication module Scheduler, which enables processes in different Local Area Networks (LANs) to communicate with the global server in the Wide Area Network (WAN).

2.2 Pipelines

Building an FL system with FedLab includes two parts. The first part is the definition of a communication agreement. The prototypes of synchronous and asynchronous communication patterns have been implemented for users already. With effortless modification on `NetworkManager` of client/server, users can fulfill the requirements of specific communication
agreements. The second part is ServerHandler/Trainer, which represents the FL optimization procedure on the server/client. The flexibility of FedLab is embodied in practical usages that the user can customize only one or two target components in FL workflow while relying on defaults for the remaining parts. Code examples are shown in Listing 1.

```python
# ==== Server Example
smodel = AlexNet()
shandler = ServerHandler(smodel) # Optimization part
snetwork = DistNetwork((server_ip, server_port), world_size, server_rank) # Configuration
smanager = ServerManager(handler, network) # Communication part
smanager.run()

# ==== Client Example
cmodel = AlexNet()
ctrainer = Trainer(cmodel, train_loader, optimizer, criterion) # Optimization part
.cnetwork = DistNetwork((server_ip, server_port), world_size, client_rank) # Configuration
cmanager = ClientManager(trainer, network) # Communication part
cmanager.run()
```

Listing 1: Code examples for server and client

2.3 Impacts

The impacts of FedLab on the FL community can be broadly summarized as follows:

**Flexible FL Framework.** Via the flexible APIs and highly customized modules in FedLab, researchers can easily verify their research ideas with simulation acceleration and implementation flexibility.

**Comprehensive Learning Materials.** Through the provided comprehensive demos and tutorials for FL learners, FedLab could support the development of the FL community by helping both beginners and experts.

**Integrated Baseline Algorithms and Benchmarks.** Researchers can easily conduct intensive comparison experiments based on benchmark datasets and baseline algorithms integrated in FedLab.

For the most benefits of the machine learning community, FedLab is built on the most popular ML framework PyTorch (according to a study on HuggingFace and PageswithCode 1), instead of other existing parameter-server frameworks (Chen et al., 2015; Abadi et al., 2016; Jiang et al., 2018). Moreover, a number of federated learning models (Zhang et al., 2022a; Sultana, 2022) have been built upon FedLab.

3. Conclusion and Future Work

In this paper, we introduce a flexible and lightweight FL framework FedLab. FedLab provides necessary FL modules, reproduction of FL methods for use, as well as detailed documentation with tutorials online. Thus, FedLab can make it painless for researchers to verify their ideas in any stage of FL. Future work directions include providing more FL algorithms and utility experiment toolkits in FedLab.

---

Acknowledgments

This work was partially supported by the National Key Research and Development Program of China (No. 2018AAA0100204), a key program of fundamental research from Shenzhen Science and Technology Innovation Commission (No. JCYJ20200109113403826), the Major Key Project of PCL (No. PCL2021A06), an Open Research Project of Zhejiang Lab (NO.2022RC0AB04), and Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (No. 2022B121201005).

References


Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. *CoRR*, abs/2102.02079, 2021a.


