FLGo: A Fully Customizable Federated Learning Platform

Zheng Wang[1,2] , **Xiaoliang Fan**[1,2*] , **Zhaopeng Peng**[1,2] , **Xueheng Li**[1,2] , **Ziqi Yang**[1,2] ,
Mingkuan Feng[1,2] , **Zhicheng Yang**[1,2] , **Xiao Liu**[3] , **Cheng Wang**[1,2]
[1] Fujian Key Laboratory of Sensing and Computing for Smart Cities, School of Informatics, Xiamen University, Xiamen, China
[2] Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China, Xiamen University, Xiamen, China
[3] School of Information Technology, Deakin University, Geelong, Australia
zwang@stu.xmu.edu.cn, fanxiaoliang@xmu.edu.cn, pengzhaopeng@stu.xmu.edu.cn
22920202202797@stu.xmu.edu.cn, lafinhana@outlook.com, 22920202202770@stu.xmu.edu.cn,
zcyang@stu.xmu.edu.cn, xiao.liu@deakin.edu.au, cwang@xmu.edu.cn

Abstract

Federated learning (FL) has found numerous applications in healthcare, finance, and IoT scenarios. Many existing FL frameworks offer a range of benchmarks to evaluate the performance of FL under realistic conditions. However, the process of customizing simulations to accommodate application-specific settings, data heterogeneity, and system heterogeneity typically remains unnecessarily complicated. This creates significant hurdles for traditional ML researchers in exploring the usage of FL, while also compromising the shareability of codes across FL frameworks. To address this issue, we propose a novel lightweight FL platform called FLGo, to facilitate cross-application FL studies with a high degree of shareability. Our platform offers 40+ benchmarks, 20+ algorithms, and 2 system simulators as out-of-the-box plugins. We also provide user-friendly APIs for quickly customizing new plugins that can be readily shared and reused for improved reproducibility. Finally, we develop a range of experimental tools, including parallel acceleration, experiment tracker and analyzer, and parameters auto-tuning. FLGo is maintained at flgo-xmu.github.io.

1 Introduction

Federated learning (FL) has attracted wide attention from both academics and industries [Kairouz *et al.*, 2019; He *et al.*, 2020; Li *et al.*, 2020]. Its characteristic of privacy protection has made it a popular choice for data security compliance applications, including medicine [Liu *et al.*, 2021a], finance [Long *et al.*, 2021], internet of things [Nguyen *et al.*, 2021], etc. Researchers have proposed numerous FL algorithms to address both data [Wang *et al.*, 2021; Li *et al.*, 2020; Karimireddy *et al.*, 2020] and system heterogeneity[Li *et al.*, 2020; Wang *et al.*, 2020; Wang *et al.*, 2022]. However, the effectiveness of these algorithms is limited [Li *et al.*, 2022], where each algorithm can only bring non-trivial improvement, or

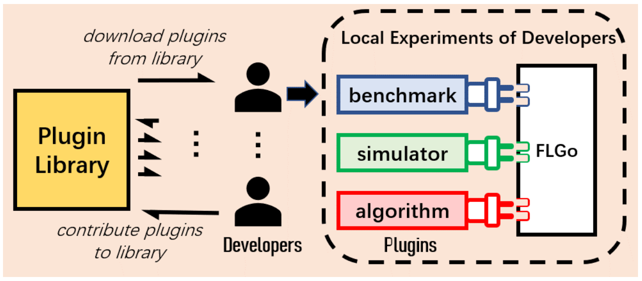

download plugins from library
Plugin Library
contribute plugins to library
Developers
Local Experiments of Developers
benchmark
simulator
algorithm
FLGo
Plugins

Figure 1: FLGo implements benchmarks, simulators, and algorithms as independent plugins, where developers can download plugins and share locally customized plugins through a plugin library.

even be worse than FedAvg in specific contexts [Li *et al.*, 2022]. Therefore, it is essential for engineers to conduct quantitative evaluations on available FL algorithms before deploying them.

Recently, several platforms were proposed to enable researchers to fairly evaluate FL algorithms by providing multiple benchmarks. FedScale[Lai *et al.*, 2022] uses systematic datasets to simulate realistic mobile-device settings and real-world federated datasets, while FederatedScope[Xie *et al.*, 2023] employs an architecture driven by events to customize the behaviors of different participants. However, the complexity involved in implementing customized settings, such as different applications, heterogeneous data, and system heterogeneity, is commonly overlooked. This substantial limitation poses a significant barrier for machine learning researchers seeking to employ FL frameworks in practical scenarios. Additionally, the high complexity of customization further harms the code shareability due to the absence of standardized guidelines for customization, even when employing the same FL framework. In short, addressing these issues is crucial in order to develop an adaptable FL framework that facilitates cross-application FL studies with a high degree of shareability.

To tackle these problems, we develop FLGo, a novel lightweight FL platform that streamlines cross-application FL research with a high degree of shareability. By leveraging

	TFF	LEAF	PySyft	FedML	FederatedScope	FedScale	Flower	FLGo
Data Heterogeneity	1	1	1	1	1	1	1	1
System Heterogeneity	0	×	×	0	1	1	0	1
High-Level API	0	×	0	1	1	1	1	1
Multi-Architecture	×	×	×	1	1	0	0	1
Asynchrounous	×	×	×	0	1	×	×	1
Experiment Manager	×	×	×	1	1	0	×	1
Behavior Customization	×	×	×	1	1	0	0	1
Benchmark Customization	×	×	×	0	×	×	×	1
Heterogeneity Customization	×	×	×	0	0	0	0	1

Table 1: Comparison on FL Frameworks. 0 represents partially implemented, 1 represents fully implemented, × represents not implemented.

the *plugin library*, developers can easily distribute their customization as *plugins*, as illustrated in Fig.1. Our main contributions are summerized as follows:

- We propose a novel FL platform, FLGo, to support customization with high shareability, where benchmarks, algorithms and simulators are implemented as plugins enable the fast development of cross-application FL scenarios.
- We provide 40+ comprehensive FL benchmarks, 20+ algorithms, and 2 system simulators in FLGo. We also develop useful tools to support various experimental purposes including parallel acceleration, experiment tracker and analyzer, and parameters auto-tuning.
- We conduct comprehensive experiments on 6 datasets to evaluate the functionality of FLGo platform by customizing various data and system heterogeneity settings.

2 Related Works

Existing frameworks for FL research. Extensive federated frameworks were developed to better support FL studies in both academic and industrial scenarios [He *et al.*, 2020; Liu *et al.*, 2021b]. LEAF [Caldas *et al.*, 2018] provides several open-source federated benchmarks. TFF [Bonawitz *et al.*, 2019] focuses on the application of large-scale mobile devices and PySyft [Ryffel *et al.*, 2018] focuses on deep learning for data security. FedML [He *et al.*, 2020] facilitates equitable comparison of algorithms through the provision of benchmarks, multiple topologies, and diverse computing paradigms. FedScale [Lai *et al.*, 2022] provides real-world datasets to evaluate the statistical efficiency and system efficiency and improve the efficiency under large scalability. FederatedScope [Xie *et al.*, 2023] employs an event-driven architecture to enable arbitrary customization on parties' behaviors. Flower [Beutel *et al.*, 2020] supports different backends and is also scalable. However, the challenge of evaluating FL algorithms for customized applications, data heterogeneity, and system heterogeneity is often overlooked. This issue presents a natural divide between conventional machine learning developers and the FL community. By addressing this challenge and enabling greater compatibility and shareability, FL can become more accessible to a wider range of developers, further promoting its adoption and advancement.

Existing frameworks are inadequate. Although great efforts have been made in developing FL community, these

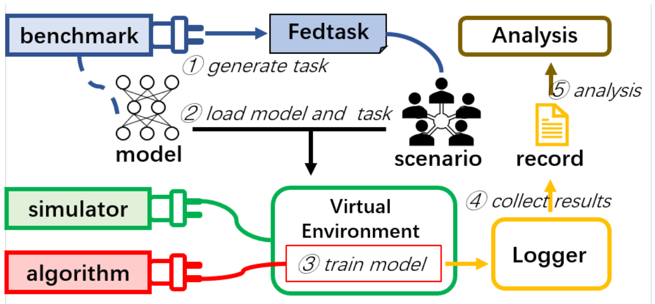


Figure 2: The main workflow in FLGo.

frameworks failed to 1) fast convert any ML task into a federated benchmark to attract more researchers in the traditional ML community, and 2) provide high-level APIs for customization on heterogeneity. To simulate more realistic and complex scenarios when adopting FL, we propose FLGo. We make a comparison between FLGo and other frameworks to distinguish the advantages of FLGo in Table.1.

3 FLGo Features

3.1 Overview

The workflow in FLGo framework is as shown in Fig.2. First, the blue *benchmark* plugin generates a static federated task on the disk, which describes how the data is distributed among participants. Second, FLGo system will load the model and the stored task to construct the simulation. Then, the red *algorithm* plugin will optimize the model under the virtual environment created by the green *simulator* plugin. Finally, the *logger* will trace the running-time variables and save them as a record on the disk for further analysis.

3.2 Benchmark Module

Comprehensive Benchmarks

We have provided 40+ out-of-the-box FL benchmarks in Table.2. The details of each dataset are on the website[1]. These benchmarks enable FLGo to support a wide range of studies through three characteristics:

- **Cross-Category.** We implement the templates of benchmark plugins for different types of input data including images, text, graphs, series, and tables.

[1]https://flgo-xmu.github.io

Category	Task	Scenario	Datasets
CV	Classification	Horizontal & Vertical	CIFAR10, CIFAR100, MNIST, FashionMNIST, FEMNIST, EMNIST, SVHN
	Detection	Horizontal	Coco, VOC
	Segmentation	Horizontal	Coco, SBDataset
NLP	Classification	Horizontal	Sentiment140, AG_NEWS, sst2
	Text Prediction	Horizontal	Shakespeare, Reddit
	Translation	Horizontal	Multi30k
Graph	Node Classification	Horizontal	Cora, Citeseer, Pubmed
	Link Prediction	Horizontal	Cora, Citeseer, Pubmed
	Graph Classification	Horizontal	Enzymes, Mutag
Rec	Rating Prediction	Horizontal&Vertical	Ciao, Movielens, Epinions, Filmtrust, Douban
Series	Time series forecasting	Horizontal	Electricity, Exchange Rate
Tabular	Classification	Horizontal	Adult, Bank Marketing, Heart Disease
Synthetic	Regression	Horizontal	Synthetic, DistributedQP

Table 2: An overview of benchmarks in FLGo.

Name	Feature Skew	Label Skew	Concept Drift	Concept Shift	Quantity Skew
IIDPartitioner [2017]					✓
DiversityPartitioner [2017]	✓	✓			
DirichletPartitioner [2019]	✓	✓			✓
GaussianPerturbationPartitioner [2022]	✓				✓
IDPartitioner	✓	✓	✓	✓	✓
VerticalPartitioner	✓				✓
NodeLouvainPartitioner [2021]	✓				

Table 3: Partitioners for data heterogeneity in FLGo.

- **Cross-Application.** Tasks of different applications like finance, healthcare, and IoT scenarios are integrated.
- **Cross-Architecture.** Benchmark plugins can also support FL with different architectures like horizontal FL, vertical FL, decentralized FL, and hierarchical FL.

Customization

```
/*Code.1
import flgo
flgo.gen_benchmark(name, config, target_path,
data_type, task_type)
```

We design a general paradigm to ease the customization of various benchmarks. In addition, we provide templates of benchmark plugins for different categories of input data and different types of tasks. By using the templates, developers can integrate their customized benchmarks with only one line of code as shown in Code.1.

Paritioner for Data Heterogeneity

To support studies in FL with different types of data heterogeneity categorized by [Kairouz *et al.*, 2019], we create reusable data partitioners to simulate real-world local data distributions. In this way, developers can arbitrarily combine the benchmark plugins and partitioners together to design the data heterogeneity. We also provide APIs for customizing new partitioners. We list the data heterogeneity currently supported by FLGo in Table.3.

3.3 Simulator Module

Existing frameworks [Xie *et al.*, 2023; Lai *et al.*, 2022] provided realizations of particular system heterogeneity as benchmarks, but they fail to support APIs for customization on the system heterogeneity. Since both different types and degrees of system heterogeneity have non-trivial impacts on the training effectiveness [Lai *et al.*, 2022; Wang *et al.*, 2022], it's essential to allow developers to customize the system heterogeneity. To this end, we first conclude four types of system heterogeneity that were investigated in previous FL studies:

- **Availability.** If a client is unavailable, the server cannot select it to join the model training [Wang *et al.*, 2022].
- **Responsiveness.** Responsiveness describes the length of the period for the server to wait for the responses of participants [Chai *et al.*, 2020].
- **Completeness.** The client model updates may be incomplete[Li *et al.*, 2020; Wang *et al.*, 2020].
- **Connectivity.** Some participants may lose connection for a long period [Jiang *et al.*, 2023] even if they had been selected to join.

According to the aforementioned four types of system heterogeneity, we design 2 simulators, where one is based on synthetic data and another one is based on real-world data [Lai *et al.*, 2022]. We first develop a client-state machine and a global virtual clock to simulate heterogeneous systems. For example in Fig.3, it takes client i 3 units of time to finish local training at the first round, and the length of time units

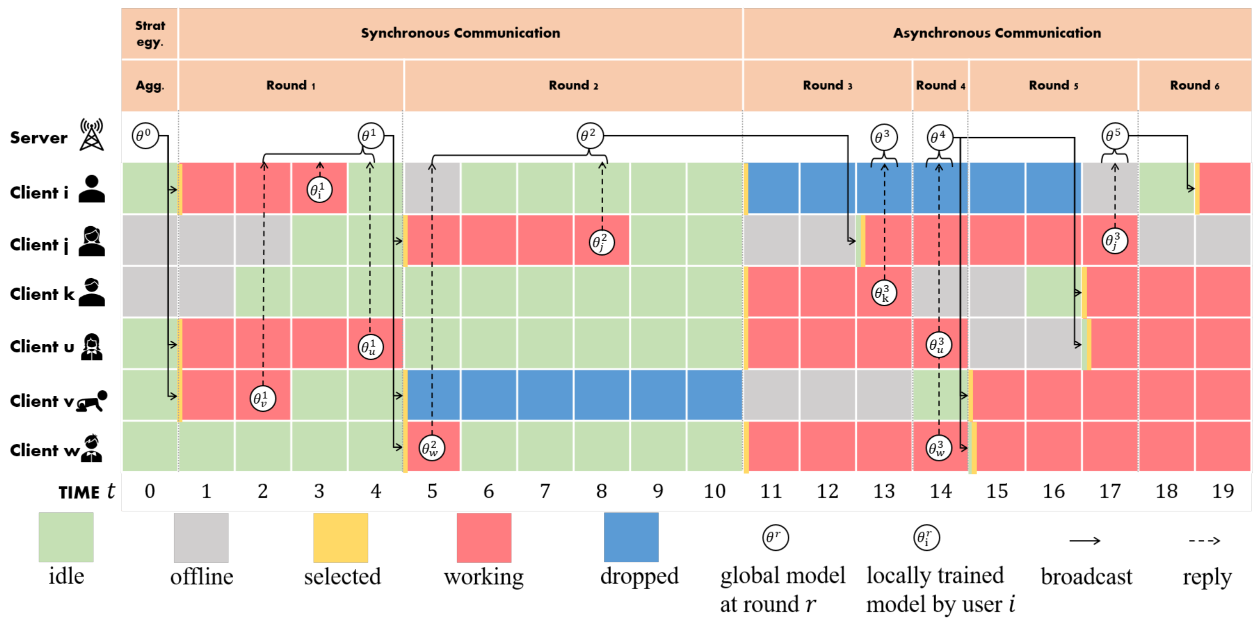


Figure 3: FLGo simulates synchronous and asynchronous scenarios by using the global clock and the client-state machine, where each client's state can shift either as time passes or when a particular condition is reached (e.g., being selected by the center).

that are spent by client u is 4. Based on the global clock and the client-state machine, we then design easy-to-use APIs for developers to directly define how the state will shift for each participant at each moment or each round. In this way, the four types of system heterogeneity can be easily customized.

3.4 Algorithm Module

In FLGo, each algorithm plugin is described by parties with their behaviors. We conclude the main features of the algorithm module of FLGo in the following subsections.

Time-based asynchronism

Most of the previous studies in asynchronous FL are based on round-wise asynchronism, which cannot be fairly compared with the synchronous FL methods since the time cost of each round is overlooked [Xie *et al.*, 2019; Nguyen *et al.*, 2022]. To this end, we implement the asynchronous FL algorithms based on the global virtual clock, where the developers should consider the behaviors of parties at each time unit when realizing asynchronous algorithms. For example, the server of fully asynchronous FL in Fig.3 samples participants once they become available (e.g. client j at the 13th time unit) in round 3, and receives the locally trained model from client k at the same time unit. As a comparison, the synchronous FL server in rounds 1 and 2 will wait for the slowest participant at each iteration. In this way, the algorithms in syncFL and asyncFL are comparable under the same metric of time, which enables developers to search for the best suitable strategy under more realistic settings.

Flexible Communication

To enable different communication behaviors of parties, we follow the gRPC protocol to model the communication processes. Every time a party sends a message to a remote one, it will receive a response that is returned by a corresponding remote function. The transferred information is organized as key-value pairs for high flexibility. By registering the message handler to the specific message, developers can arbitrarily customize the communication process. For example in above Code.2, a party can send a specific message 'forward' to ask another remote party for partial activations by registering this message to the action of forward computing.

```
/*Code.2
import flgo.algorithm.vflbase

class ActiveParty(vflbase.ActiveParty):
    def iterate(self):
        ...
        # prepare the sending package
        pkg = self.set_message(\
              mtype='forward', package={})
        # request activations from the party
        acts = self.communicate_with(\
               party.id, pkg)['res']

class PassiveParty(vflbase.PassiveParty):
    def initialize(self):
        # register the action to 'forward'
        self.actions = {'forward':\
                        self.my_forward}

    # define the action
    def my_forward(self, package):
        ...
        return {'res':activations}
```

Cross-Architecture

To ease the development of algorithms with different architectures (i.e. horizontal FL [McMahan *et al.*, 2017], vertical

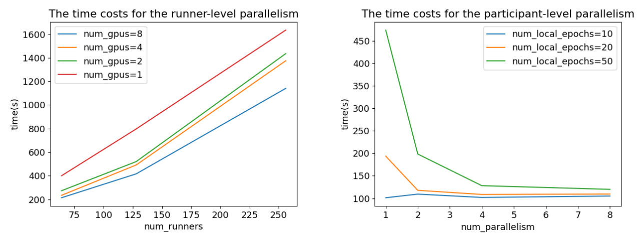


Figure 4: The acceleration by two levels of parallelism: (left) runner-level, and (right) participant-level.

FL [Liu *et al.*, 2022], decentralized FL[Beltrán *et al.*, 2023], and hierarchical FL [Liu *et al.*, 2019]), we respectively conclude general paradigms for each type of architecture based on previous works. By using these paradigms, developers can easily customize their algorithms by only writing a few codes to replace the corresponding parts in the paradigm.

3.5 Experiment Tools

Parallel Acceleration

We provide two-level parallels in FLGo to accelerate the FL training process in simulations. The first one is the runner-level, where each runner $r = (\mathcal{A}, \theta, G, \mathcal{T}, s)$ denotes using the algorithm $\mathcal{A}$ to optimize the global model θ with the hyperparameters G on the federated task $\mathcal{T}$ under the environment created by the simulator s. The runner-level parallelism enables developers to use multiple GPUs to save efficiency, where devices will be automatically scheduled by the device scheduler module to avoid out-of-memory errors. A runner queue is used to promise that all the runners will successfully finish their tasks, whereas the losers will be automatically put back in the queue again. The second-level parallelism is the participant-level, where each participant denotes a virtual client of each runner. The participant-level parallelism accelerates the training process by reducing the wall-clock time of iterative local training of all the participants. The acceleration results are shown in Fig.4. The left part in Fig.4 shows that increasing the runner-level parallelism can well reduce the time cost as the number of runners grows. The right part in Fig.4 shows that increasing the number of processes can also save the training efficiency, especially when the cost of data/model transmission across different devices is relatively smaller than the local training processes.

Experiment Tracker and Analyzer

We realize two modules to help do experiments. First, we use the logger to track the running-time variables of interest without writing intrusive codes, and we preserve APIs to customize loggers for different experimental purposes. The logger module is also compatible with popular ML logging tools like wandb [Biewald, 2020]. The running-time variables of interest will finally be stored on the disk as records. Second, we develop the analyzer to load the experimental records from the disk and help make further analyses on them. We also provide basic tools to visualize the records and generate tables in a customizable manner.

Parameters Auto-Tuning

We realize automatically tuning hyper-parameters based on the aforementioned parallelism. By using this module, developers can input the grid of hyper-parameters and specify the GPUs to find the group of hyper-parameters that achieves the optimal performance on the validation dataset.

4 Experiments

In this section, we conduct experiments to show how FLGo facilitates FL studies by customization from three aspects: data heterogeneity, system heterogeneity, and asynchronism. The meanings of heterogeneity notions are in the appendix.

4.1 Experimental Settings

Datasets and Models

The datasets and corresponding models are concluded in Tabel.4. For data heterogeneity experiments, we respectively conduct I.I.D, quantity skew, and non-I.I.D scenarios for three datasets. For CIFAR10, we use I.I.D, Imbalance (0.5), and Dirichlet (0.3) data partition. For Cora, we employ IID, Imbalance, and Louvain data partition. For the Shakespeare dataset, we use IID, Imbalance, and Diversity (0.2) data partition. For system heterogeneity and asynchronism experiments, we use I.I.D and Dirichlet (1.0) data partition.

Baselines

For data heterogeneity experiments, the baselines are FedAvg[McMahan *et al.*, 2017], FedProx[Li *et al.*, 2020], and Scaffold[Karimireddy *et al.*, 2020]. For system heterogeneity experiments, the baselines are FedAvg[McMahan *et al.*, 2017], FedNova[Wang *et al.*, 2020], and FedProx[Li *et al.*, 2020]. For asynchronism experiments, the baselines are synchronous algorithms FedAvg[McMahan *et al.*, 2017], FedProx[Li *et al.*, 2020] and the asynchronous algorithm FedAsync[Xie *et al.*, 2019].

Hyparameters

We tune all the methods by grid search. The hyperparameters for the CIFAR10 and Cora datasets: the number of rounds to be 1000, the learning rate to be 0.1, proportion to be 1.0,epoch $E \in \{1, 3, 5\}$, and batch size to be 50. For the Shakespeare dataset, we set rounds to 100, learning rate $\eta \in \{0.1, 0.3, 0.8\}$, proportion to be 0.1, epoch $E \in \{5, 10\}$, and batch size to be 64. For Fashion Mnist and SVHN dataset, the number of rounds to be 1000, the learning rate to be 0.1, epoch to be 5, and batch size to be 50. For Reddit dataset, the number of rounds to be 1000, the learning rate, epoch and batch size to be 1, 1 and 50, respectively.

Implementation

Experiments are run on a 64 GB-RAM Ubuntu 18.04.6 server with Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 4 NVidia(R) 2080Ti GPUs, and PyTorch 1.12.0.

4.2 Data Heterogeneity

We compare the performance of three state-of-the-art FL algorithms toward the non-IID problem in Table.5. The algorithm's performance varies significantly with different datasets and distribution settings. For CIFAR10, Scaffold

	Data Heterogeneity			**System Heterogeneity**		**Asynchronous**
Dataset	Cifar 10	Cora	Shakespeare	Fashion Mnist	Reddit	SVHN
Task	Image Classification	Node Classification	Next-Character Prediction	Image Classification	Next-word Prediction	Image Classification
Model	Two-layer CNN	Two-layer GCN	Stacked LSTM	Two-layer CNN	Stacked LSTM	Two-layer CNN

Table 4: Experimental datasets and corresponding models.

	Cifar10 - CNN			**Cora - GCN**			**Shakespeare - LSTM**		
Data Heterogeneity	**IID**	**Imbalance**	**Dir(0.3)**	**IID**	**Imbalance**	**Louv.**	**IID**	**Imbalance**	**Div.**
fedavg	0.8216	0.8303	0.7355	0.9059	0.9022	**0.7693**	**0.4457**	0.4424	**0.3340**
fedprox	0.8206	0.8207	0.7099	0.8911	0.9022	0.7638	0.4443	**0.4432**	0.2934
scaffold	**0.8551**	**0.8474**	**0.7492**	**0.9151**	0.9022	0.7638	0.4304	0.4309	0.2775

Table 5: Comparison on model testing accuracy of different methods across datasets and heterogeneity settings.

achieves the best performance under all the data distributions. For Cora, all the methods achieve similar model performance. Scaffold achieves the best results on the IID distribution, while FedAvg performs the best on the Louvain distribution. For Shakespeare, the best results lie in FedAvg and FedProx, where Scaffold is the worst one across all the data distributions. These results suggest that there is no method that always outperforms other methods across datasets and heterogeneity settings, which is consistent with the observations in [Li *et al.*, 2022]. In addition, the imbalance setting won't have an obvious impact on the results of all these methods. The non-IID settings cause performance reduction for all the methods on different datasets compared to the IID settings.

4.3 System Heterogeneity

We respectively investigate the impact of 4 different types of system heterogeneity on three FL algorithms in FashionMNIST and Reddit datasets: FedAvg[McMahan *et al.*, 2017], FedNova[Wang *et al.*, 2020] and FedProx[Li *et al.*, 2020]. Fig.5(a) suggests that the heterogeneity of client availability has a non-trivial negative impact on the performance of all the methods. Especially, when data heterogeneity meets availability heterogeneity, the performance is further worsened than the IID scenario on both two datasets. Fig.5(b) shows the impact of heterogeneity in client completeness. When clients are not able to complete all the local training epochs, the training efficiency is correspondingly degraded. Although FedNova is claimed to be robust to varying completeness, its performance is slightly worse than FedAvg and FedProx in FashionMNIST, which may be due to the over-enlarging of short model updates. Fig.5(c) shows the impact of heterogeneity in client connectivity. We use the virtual time as the horizontal axis instead of the communication round since the server will wait for another unit of time once there exist dropping-out clients. The results show that less connectivity will increase the time cost of the training process. For FashionMNIST, the connectivity has only a trivial impact on the model performance. For Reddit, less connectivity causes performance degradation in the non-IID setting. Fig.5(d) shows the impact of heterogeneity in client responsiveness. This heterogeneity will only influence the time cost of training since the server will wait for all the clients. The results show that a larger variance of client latencies leads to larger time costs with the same client latency mean. This is consistent with the *Cask Effect* where the slowest client will dominate the time costs for synchronous algorithms.

4.4 Asynchronism

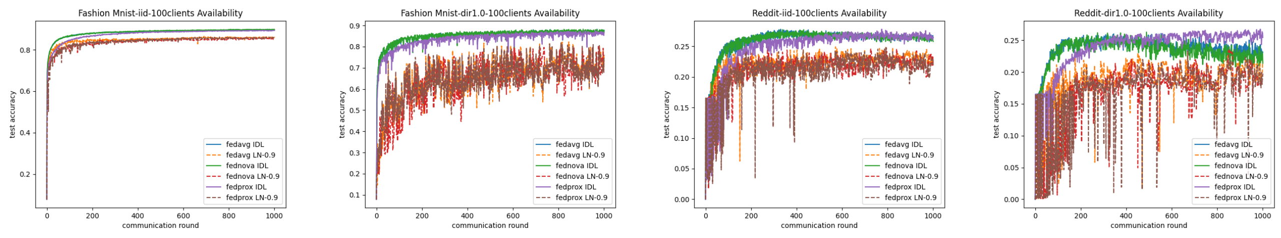
We create a complex combination of different heterogeneity to customize the environment. First, we make the probability of each client follow the distribution $lognormal(0, -log(0.1))$. We use the same completeness heterogeneity setting in [Li *et al.*, 2020]. We make the clients have the same drop probability of 0.5, and we set the response latencies of clients to follow a log-normal distribution with $(mean, var.) = (200, 50)$. The results in Fig.5(e) show that asynchronous strategies can significantly reduce time costs when carrying out the same times of aggregation. Under the IID setting, FedAsync reduces the time cost to achieve the same model performance against other baselines. Under the non-IID setting, the performance of FedAsync is worse than FedAvg and FedProx after the same amounts of aggregations, which suggests that the effectiveness of aggregation for asynchronous strategies can be further improved.

5 Conclusion

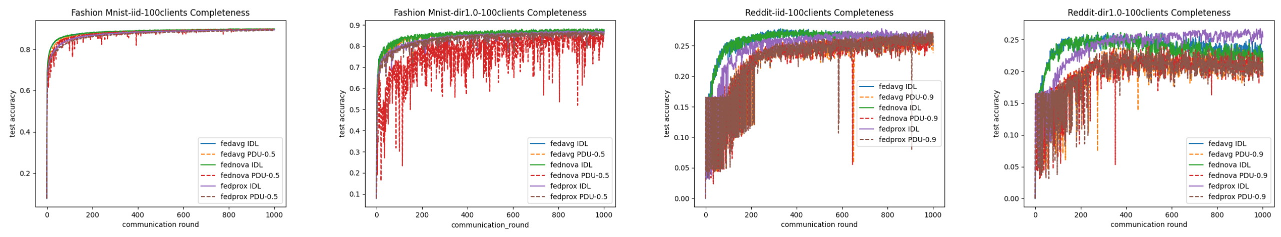
In this work, we presented a novel lightweight FL platform, FLGo, to facilitate cross-application FL studies with a high ability of shareability . FLGo offers 40+ benchmarks, 20+ algorithms, and 2 system simulators as out-of-the-box plugins. We also developed a range of experimental tools for various experiment purposes. Comprehensive experiments were conducted to verify the ability of customization of FLGo, under various data and system heterogeneity settings.

Acknowledgments

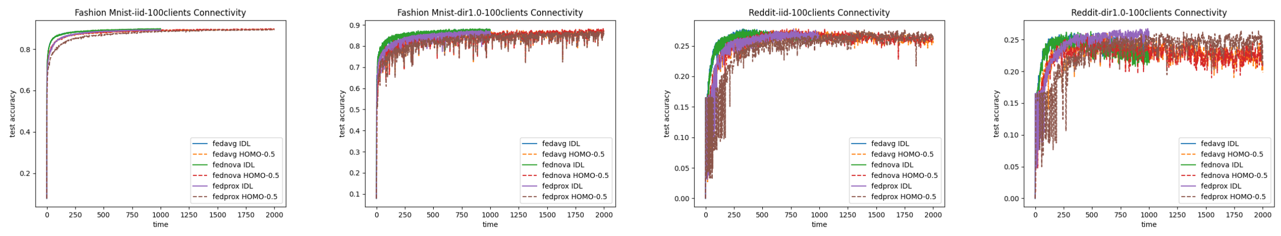
The research was supported by Natural Science Foundation of China (62272403, 61872306).



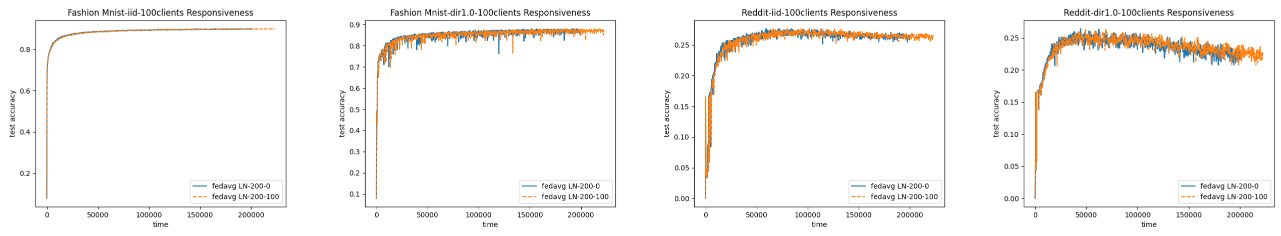
(a) The impact of client availability



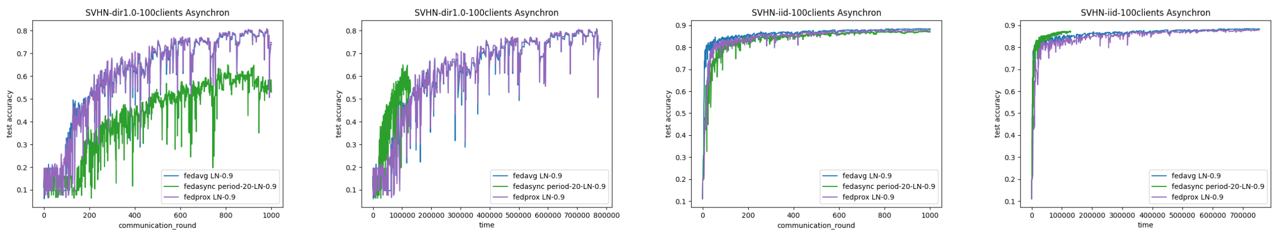
(b) The impact of client completeness



(c) The impact of client connectivity



(d) The impact of client responsiveness



(e) Asynchronism

Figure 5: The subfigures in the first 4 rows conduct the impact of system heterogeneity, where each row's heterogeneity is of: (a) availability; (b) completeness; (c) connectivity; and (d) responsiveness. For each row, in the order from left to right: IID-Fashion, Dirichlet-Fashion, IID-Reddit, and Dirichlet-Reddit. (e) compares synchronous and asynchronous strategies in the simulation of the complex combinations of different heterogeneity.

References

[Beltrán *et al.*, 2023] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state-of-the-art, frameworks, trends, and challenges, 2023.

[Beutel *et al.*, 2020] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[Biewald, 2020] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

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

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

[Chai *et al.*, 2020] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. *CoRR*, abs/2001.09249, 2020.

[He *et al.*, 2020] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[Hsu *et al.*, 2019] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, abs/1909.06335, 2019.

[Jiang *et al.*, 2023] Zhifeng Jiang, Wei Wang, and Ruichuan Chen. Taming client dropout and improving efficiency for distributed differential privacy in federated learning, 2023.

[Kairouz *et al.*, 2019] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, and et al. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.

[Karimireddy *et al.*, 2020] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[Lai *et al.*, 2022] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*, pages 11814–11827. PMLR, 2022.

[Li *et al.*, 2020] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[Li *et al.*, 2022] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *IEEE International Conference on Data Engineering*, 2022.

[Liu *et al.*, 2019] Lumin Liu, Jun Zhang, S. H. Song, and Khaled Ben Letaief. Client-edge-cloud hierarchical federated learning. *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019.

[Liu *et al.*, 2021a] Quande Liu, Cheng Chen, Jing Qin, Qi Dou, and Pheng-Ann Heng. Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space, 2021.

[Liu *et al.*, 2021b] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. Fate: An industrial grade platform for collaborative learning with data protection. *The Journal of Machine Learning Research*, 22(1):10320–10325, 2021.

[Liu *et al.*, 2022] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Qian Yang. Vertical federated learning. *ArXiv*, abs/2211.12814, 2022.

[Long *et al.*, 2021] Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking, 2021.

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

[Nguyen *et al.*, 2021] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[Nguyen *et al.*, 2022] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pages 3581–3607. PMLR, 2022.

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

[Wang *et al.*, 2020] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623, 2020.

[Wang *et al.*, 2021] Zheng Wang, Xiaoliang Fan, Jianzhong Qi, Chenglu Wen, Cheng Wang, and Rongshan Yu. Federated learning with fair averaging, 2021.

[Wang *et al.*, 2022] Zheng Wang, Xiaoliang Fan, Jianzhong Qi, Haibing Jin, Peizhen Yang, Siqi Shen, and Cheng Wang. Fedgs: Federated graph-based sampling with arbitrary client availability, 2022.

[Xie *et al.*, 2019] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *ArXiv*, abs/1903.03934, 2019.

[Xie *et al.*, 2023] 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. *Proceedings of the VLDB Endowment*, 16(5):1059–1072, 2023.

[Zhang *et al.*, 2021] Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu-Ming Yiu. Subgraph federated learning with missing neighbor generation. *CoRR*, abs/2106.13430, 2021.