FedML: A Research Library and Benchmark for Federated Machine Learning

Chaoyang He∗, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang
USC, Stanford, USC, MSU, MSU
Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu
USC, UW-Madison, MIT, MIT, USC
Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu
Ping An Tech., UIUC, Tencent, WeBank
Ramesh Raskar, Qiang Yang, Murali Annavaram, Salman Avestimehr∗
MIT, HKUST, USC, USC

Abstract

Federated learning (FL) is a rapidly growing research field in machine learning. However, existing FL libraries cannot adequately support diverse algorithmic development needs of FL, such as topology customization, supporting varied aggregation schemes, and so on. In this work, we introduce FedML, an open research library, and FL benchmarks to facilitate algorithm development and fair performance comparison across computing design choices. FedML supports three computing paradigms: on-device training using a federation of edge devices, distributed training in the cloud that supports exchanging of auxiliary information beyond just gradients, and single-machine simulation of a federated learning algorithm. FedML also promotes diverse algorithmic research with flexible and generic API design and comprehensive reference baseline implementations (optimizer, models, and datasets). We believe that FedML provides an efficient and reproducible means for developing and evaluating FL algorithms that would benefit the FL community. We maintain the source code, documents, and user community at https://fedml.ai.

1 Introduction

Federated learning (FL) is a distributed learning paradigm that aims to train machine learning models from scattered and isolated data [1]. FL differs from cloud-based distributed training in three major aspects: 1) statistical heterogeneity, where different datasets may have different distributions, 2) system constraints, where some devices may have limited compute and memory capacity, and 3) trustworthiness, where some nodes may participate with malicious intent. Solving these unique challenges calls for research across a variety of fields, including machine learning, wireless communication, mobile computing, distributed systems, and information security, making federated learning a truly interdisciplinary research field.

In the past few years, several efforts have been made to address these challenges. To tackle the challenge of statistical heterogeneity, distributed optimization methods such as Adaptive Federated Optimizer [2], FedNova [3], FedProx [4], and FedMA [5] have been proposed. These techniques aim to achieve model convergence even in the presence of dataset heterogeneity. To tackle the challenge of system constraints, researchers apply sparsification and quantization techniques to reduce the communication overheads and computation costs during the training process [6, 7, 8, 9, 10, 11, 12, 13, 14]. To tackle the challenge of trustworthiness, existing research focuses on developing new defense techniques for adversarial attacks to make FL robust [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 24, 27, 28], and proposing methods such as differential privacy (DP) and secure multiparty computation (SMPC) to protect privacy [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39].

∗Corresponding authors. Email: chaoyang.he@usc.edu, avestime@usc.edu

Table 1: Comparison between FedML and existing federated learning libraries and benchmarks.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>standalone simulation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>distributed computing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>on-device training (Mobile, IoT)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>topology customization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>flexible message flow</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>exchange message customization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FedAvg</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>decentralized FL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FedNAS (beyond gradient/model)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VFL (vertical federated learning)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SplitNN (split learning)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>linear models (e.g., Logistic Regression)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>shallow NN (e.g., Bi-LSTM)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Model DNN (e.g., ResNet)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>vertical FL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Although a lot of progress has been made, existing efforts are confronted with a number of limitations that we argue are critical to FL research:

**Lack of support of diverse FL computing paradigms.** Distributed training libraries in PyTorch [40], TensorFlow [41], MXNet [42], and distributed training-specialized libraries such as Horovod [43] and BytePS [44] are designed for distributed training in data centers. Although simulation-oriented FL libraries such as TensorFlow-Federated (TFF) [45], PySyft [32], and LEAF [46] are developed, they only support centralized topology-based FL algorithms like FedAvg [47] or FedProx [4]. Furthermore, they only provide low-level communication APIs (e.g., TFF) or simulate a federation of nodes using a single machine, making them unsuitable or difficult to develop FL algorithms that require the exchange of complex auxiliary information and customized training procedures. Production-oriented libraries such as FATE [48] and PaddleFL [49] are released by industry. However, they are not designed as flexible frameworks that allow topology changes or allow drop-in replacement of different kinds of averaging algorithms. These features are necessary to support algorithmic innovation for open FL problems.

**Lack of support of diverse FL configurations.** We argue that practical FL systems will have diverse network topology, require varied formats to exchange information beyond just gradients, and different training procedures. In terms of network topology, a variety of network topologies such as vertical FL [50, 51, 52, 53, 54, 55, 56], split learning [57, 58], decentralized FL [59, 60, 61, 62], and hierarchical FL [63, 64, 65, 66, 67, 68] have been proposed. In terms of exchanged information, besides exchanging gradients and models, recent FL algorithms propose to exchange information such as pseudo labels in semi-supervised FL [69] and architecture parameters in neural architecture search-based FL [70, 71, 72]. In terms of training procedures, the training procedures in FedGKT [73], federated GAN [74, 75], meta FL [76, 77, 78], and transfer learning-based FL [79, 80, 81, 82, 83] are very different from the vanilla FedAvg algorithm [47]. Unfortunately, such diversity in network topology, exchanged information, and training procedures are not supported in existing FL libraries.

**Lack of standardized FL algorithm implementations and benchmarks.** The diversity of libraries used for algorithm implementation in existing work makes it difficult to fairly compare their performance. The diversity of benchmarks used in existing work also makes it difficult to fairly compare their performance. The non-I.I.D. characteristic of FL makes such comparison even more challenging [84]: training the same DNN on the same dataset with different non-I.I.D. distributions produces varying model accuracies. Table 8 summarizes the datasets and models used in recently published work. We observe that the experimental settings of these work differ in terms of datasets, non-I.I.D. distributions, models, and the number of clients involved in each round. Any difference in these settings could affect the results.

In this work, we present FedML, an open research library, and the benchmark to address the aforementioned limitations and facilitate FL research. FedML provides an end-to-end toolkit to facilitate FL algorithm development and fair performance comparison under diverse computing paradigms and configurations. Table 1 summarizes the key differences between FedML and existing FL libraries and benchmarks. The highlights of FedML are summarized below:
(i) Support of diverse FL computing paradigms. FedML supports three diverse computing paradigms: 1) on-device training for edge devices such as smartphones and Internet of Things (IoTs), 2) distributed computing in the cloud enabling diverse templates of data communication, and 3) single-machine simulation to meet algorithmic and system-level research requirements under different system deployment scenarios.

(ii) Support of diverse FL configurations. FedML introduces a worker/client-oriented programming interface to enable diverse network topologies, flexible information exchange among workers/clients, and various training procedures.

(iii) Standardized FL algorithm implementations. FedML includes standardized implementations of well known FL algorithms. These implementations not only help users to familiarize themselves with FedML APIs but also can be used as baselines for comparisons with newly developed FL algorithms.

(iv) Standardized FL benchmarks. FedML provides standardized benchmarks with well-defined evaluation metrics, multiple synthetic and real-world non-I.I.D. datasets, as well as verified baseline results to facilitate fair performance comparison.

(v) Fully open and evolving. FL is a research field that evolves at a considerably fast pace. This requires FedML to adapt at the same pace. We will continuously expand FedML to optimize three computing paradigms and support more algorithms (distributed optimizer) and benchmarks (models and datasets) for newly explored usage scenarios. FedML is fully open and welcomes contributions from the FL research community as well. We hope researchers in diverse FL applications could contribute more valuable models and realistic datasets to our community. Promising application domains include, but are not limited to, computer vision [85, 86], natural language processing [87, 88, 89, 90, 91], finance [48, 51, 92], transportation [93, 94, 95, 96, 97, 98, 99, 100, 101, 102], digital health [103, 104, 105, 106, 107, 108, 109], recommendation [110, 111, 112, 113, 114, 115], robotics [116, 117], and smart cities [118, 119].

Figure 1: Overview of FedML library.
2 FedML Library: Architecture Design and Programming Interface

Figure 1 provides an overview of FedML library. The FedML library has two key components: FedML-API and FedML-core, which represents high-level API and low-level API, respectively.

FedML-core separates distributed communication and model training into two separate modules. The distributed communication module is responsible for low-level communication among different workers/clients. The communication backend is based on MPI (message passing interface)\(^2\) and MQTT (the most popular IoT protocol). We will also add more backends as necessary, such as RPC (remote procedure call). Inside the distributed communication module, a TopologyManager supports a variety of network topologies that can be used in many existing FL algorithms [59, 60, 61, 62]. In addition, security/privacy-related functions are also supported. The model training module is currently built upon PyTorch, but users can also implement their own trainer with TensorFlow and insert it into our framework.

FedML-API is built upon FedML-core. With the help of FedML-core, new algorithms in distributed version can be easily implemented by adopting the client-oriented programming interface, which is a novel design pattern for flexible distributed computing (Appendix B). Such a distributed computing paradigm is essential for scenarios in which large DNN training cannot be handled by standalone simulation due to GPU memory and training time constraints. FedML-API also suggests a machine learning system practice that separates the implementations of models, datasets, and algorithms. This practice enables code reuse and fair comparison, avoiding statistical or system-level gaps among algorithms led by non-trivial implementation differences. Another benefit is that FL applications can develop more models and submit more realistic datasets without the need to understand the details of different distributed optimization algorithms.

One key feature of FedML is its support of FL on real-world hardware platforms. Specifically, FedML includes FedML-Mobile and FedML-IoT, which are two on-device FL testbeds built upon real-world hardware platforms. Currently, FedML-Mobile supports Android smartphones and FedML-IoT supports Raspberry Pi 4 and NVIDIA Jetson Nano (see Appendix D for details). With such testbeds built upon real-world hardware platforms, researchers can evaluate realistic system performance, such as training time, communication, and computation cost. To support conducting experiments on those real-world hardware platforms, our FedML architecture design can smoothly transplant the distributed computing code to the FedML-Mobile and FedML-IoT platforms, reusing nearly all algorithmic implementations in the distributed computing paradigm. Moreover, for FedML-IoT, researchers only need to program with Python to customize their research experiments without the need to learn new system frameworks or programming languages (e.g., Java, C/C++)\(^3\).

Due to space limitation, we defer the detailed description of the programming interface design in Appendix B. The implementation of our APIs can be found in the GitHub repository.

3 FedML Benchmark: Algorithms, Models, and Datasets

3.1 Algorithms: Federated Optimizer

![Diagram of FL algorithms](image)

Figure 2: Supported FL algorithms that are diverse in network topology, exchanged information, and training procedures.

As shown in Figure 2, FedML is capable of supporting FL algorithms that are diverse in network topology, exchanged information, and training procedures. These supported algorithms can be used as implementation examples and baselines to help users develop and evaluate their own algorithms. Currently, FedML includes the standard implementations of multiple state of the art FL algorithms:

\(^2\)https://pypi.org/project/mpi4py/
\(^3\)https://github.com/FedML-AI/FedML-IoT
Federated Averaging (FedAvg) [47], FedOpt (server Adam) [2], FedNova (client optimizer) [3], FedProx [4], FedMA [5], as well as some novel algorithms that have diverse training paradigms and network typologies, including FedGKT (efficient edge training) [73], Decentralized FL [59], Vertical Federated Learning (VFL) [120], Split Learning [57, 58], Federated Neural Architecture Search (FedNAS) [70], and Turbo-Aggregate [36]. For more details of these algorithms, please refer to Appendix C.1 and our tutorial. 4

3.2 Models and Datasets

Inconsistent usage of datasets, models, and non-I.I.D. partition methods make it difficult to fairly compare the performance of FL algorithms (in Table 8, we summarize the non-I.I.D. datasets and models used in recently published works). To enforce fair comparison, FedML provides three categories of benchmarks as part of our GitHub distribution, which explicitly specifies the combinations of datasets, models, and non-I.I.D. partition methods to be used as the baseline in future research. The three categories of benchmarks include: 1) linear models (convex optimization), 2) lightweight shallow neural networks (non-convex optimization), and 3) deep neural networks (non-convex optimization).

<table>
<thead>
<tr>
<th>Datasets</th>
<th># of training samples</th>
<th># of testing samples</th>
<th>non-I.I.D. partition method</th>
<th># of clients / devices</th>
<th>baseline model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>60000</td>
<td>10000</td>
<td>power law</td>
<td>1000</td>
<td>logistic regression</td>
</tr>
<tr>
<td>Federated EMNIST</td>
<td>671585</td>
<td>77483</td>
<td>realistic partition</td>
<td>3400</td>
<td>logistic regression</td>
</tr>
<tr>
<td>Synthetic ($\alpha$, $\beta$) [121]</td>
<td>4305</td>
<td>4672</td>
<td>refer to [121]</td>
<td>30</td>
<td>logistic regression</td>
</tr>
</tbody>
</table>

**Federated datasets for linear models (convex optimization).** The linear model category is used for convex optimization experiments such as those presented in prior work [121] and [122]. In this category, we include three datasets (Table 2): MNIST [123], Federated EMNIST [2], and Synthetic ($\alpha$, $\beta$) [121], with the logistic regression as the baseline model.

<table>
<thead>
<tr>
<th>Datasets</th>
<th># of training samples</th>
<th># of testing samples</th>
<th>partition method</th>
<th># of clients / devices</th>
<th>baseline model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Federated EMNIST</td>
<td>671585</td>
<td>77483</td>
<td>realistic partition</td>
<td>3400</td>
<td>CNN (2 Conv + 2 FC) [2]</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>50000</td>
<td>10000</td>
<td>Pachinko Allocation</td>
<td>500</td>
<td>ResNet-18 + group normalization</td>
</tr>
<tr>
<td>Shakespeare</td>
<td>16068</td>
<td>2356</td>
<td>realistic partition</td>
<td>715</td>
<td>RNN (2 LSTM + 1 FC)</td>
</tr>
<tr>
<td>StackOverflow</td>
<td>135818730</td>
<td>16586035</td>
<td>realistic partition</td>
<td>342477</td>
<td>RNN (1 LSTM + 2 FC)</td>
</tr>
</tbody>
</table>

**Federated datasets for lightweight shallow neural networks (non-convex optimization).** Due to resource constraints of edge devices, shallow neural networks are commonly used in existing work for experiments. In this category, we include four datasets (Table 3): Federated EMNIST [46], CIFAR-100 [124], Shakespeare [47], and StackOverflow [125]. These dataset are used to train different CNN and RNN models. Please refer to Appendix C.2 for more details.

<table>
<thead>
<tr>
<th>Datasets</th>
<th># of training samples</th>
<th># of testing samples</th>
<th>partition method</th>
<th># of clients / devices</th>
<th>baseline model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>50000</td>
<td>10000</td>
<td>Latent Dirichlet Allocation</td>
<td>10</td>
<td>ResNet-56, MobileNet</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>50000</td>
<td>10000</td>
<td>Latent Dirichlet Allocation</td>
<td>10</td>
<td>ResNet-56, MobileNet</td>
</tr>
<tr>
<td>CINIC-10</td>
<td>90000</td>
<td>90000</td>
<td>Latent Dirichlet Allocation</td>
<td>10</td>
<td>ResNet-56, MobileNet</td>
</tr>
<tr>
<td>StackOverflow</td>
<td>135818730</td>
<td>10,586,035</td>
<td>realistic partition</td>
<td>342477 (10)</td>
<td>RNN (2 LSTM + 1 FC)</td>
</tr>
</tbody>
</table>

**Federated datasets for deep neural networks (non-convex optimization).** Given the resource constraints of edge devices, large DNN models are usually trained under the cross-organization FL (also called cross-silo FL) setting. For example, [70] has studied large DNN models for cross-silo FL in the hospital scenario. However, large DNN models dominate the accuracy in most learning tasks. Pushing FL of large DNN models on edge devices is challenging but a meaningful endeavor, which motivates us to make this benchmark category. For example, [73] has proposed an efficient training algorithm for large CNN models on edge devices. Table 4 shows datasets and models we include in this category. Please refer to Appendix C.2 for more details.

<table>
<thead>
<tr>
<th>Datasets</th>
<th># of training samples</th>
<th># of testing samples</th>
<th>partition method</th>
<th># of clients / devices</th>
<th>baseline model</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>50,000</td>
<td>10,000</td>
<td>Latent Dirichlet Allocation</td>
<td>10</td>
<td>ResNet-56, MobileNet</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>50,000</td>
<td>10,000</td>
<td>Latent Dirichlet Allocation</td>
<td>10</td>
<td>ResNet-56, MobileNet</td>
</tr>
<tr>
<td>CINIC-10</td>
<td>90,000</td>
<td>90,000</td>
<td>Latent Dirichlet Allocation</td>
<td>10</td>
<td>ResNet-56, MobileNet</td>
</tr>
<tr>
<td>StackOverflow</td>
<td>135,818,730</td>
<td>10,586,035</td>
<td>realistic partition</td>
<td>342477 (10)</td>
<td>RNN (2 LSTM + 1 FC)</td>
</tr>
</tbody>
</table>

**Remark:** FedML also supports advanced CV and NLP models and datasets (e.g., Google Landmark [85]; Transformer/BERT models and related datasets), please following this link for the latest update: https://github.com/FedML-AI/FedML/README.md.

4http://doc.fedml.ai/#/algorithm-reference-implementation
4 Experiments

FedML provides benchmark experimental results as references for newly developed FL algorithms. To ensure real-time updates, we maintain benchmark experimental results using Weight and Bias\(^5\). The web link to view all the benchmark experiment results can be found at our GitHub repository\(^6\).

![Graphs showing test accuracy of ResNet-56 during training.](image)

Figure 3: Test accuracy of ResNet-56 during training.

Table 5: Experimental results of training modern CNNs.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Non-I.I.D. Partition Method</th>
<th>Model</th>
<th>Number of Workers</th>
<th>Algorithm</th>
<th>Acc. on I.I.D</th>
<th>Acc. on non-I.I.D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MobileNet</td>
<td></td>
<td>FedAvg</td>
<td>91.12</td>
<td>86.32 (↓ 4.80)</td>
<td></td>
</tr>
<tr>
<td>CIFAR-100 Latent Dirichlet Allocation</td>
<td>ResNet-56</td>
<td>10</td>
<td>FedAvg</td>
<td>68.91</td>
<td>64.70 (↓ 4.21)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MobileNet</td>
<td></td>
<td>FedAvg</td>
<td>55.12</td>
<td>53.54 (↓ 1.58)</td>
<td></td>
</tr>
<tr>
<td>CINIC-10 Latent Dirichlet Allocation</td>
<td>ResNet-56</td>
<td>10</td>
<td>FedAvg</td>
<td>82.57</td>
<td>73.49 (↓ 9.08)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MobileNet</td>
<td></td>
<td>FedAvg</td>
<td>79.95</td>
<td>71.23 (↓ 8.72)</td>
<td></td>
</tr>
</tbody>
</table>

*Note: to reproduce the result, please use the same random seeds we set in the library.

To demonstrate the capability of FedML, we ran experiments in a real distributed computing environment. We trained two CNNs (ResNet-56 and MobileNet) using the standard FedAvg algorithm. Table 5 shows the experimental results, and Figure 3 shows the corresponding test accuracy during training. As shown, the accuracy of the non-I.I.D. setting is lower than that of the IID. setting, which is consistent with findings reported in prior work [84].

Table 6: Training time with FedAvg on modern CNNs (Hardware: 8 x NVIDIA Quadro RTX 5000 GPU (16GB/GPU); RAM: 512G; CPU: Intel Xeon Gold 5220R 2.20GHz).

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of workers</th>
<th>Single-GPU standalone simulation (wall clock time)</th>
<th>Multi-GPU distributed training (wall clock time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-56</td>
<td>10</td>
<td>&gt; 4 days</td>
<td>11 hours</td>
</tr>
<tr>
<td>MobileNet</td>
<td>10</td>
<td>&gt; 3 days</td>
<td>7 hours</td>
</tr>
</tbody>
</table>

*Note that the number of workers can be larger than the number of GPUs because FedML supports multiple processing training in a single GPU.

We also compared the training time of distributed computing with that of standalone simulation. The result in Table 6 reveals that when training large CNNs, the standalone simulation is about 8 times slower than distributed computing with 10 parallel workers. Therefore, when training large DNNs, we suggest using FedML’s distributed computing paradigm, which is not supported by LEAF [46]. Moreover, FedML supports multiprocessing in a single GPU which enables FedML to run a large number of training workers by using only a few GPUs. For example, when training ResNet on CIFAR-10, FedML can run 112 workers in a server with 8 GPUs.

5 Conclusion

FedML is a federated learning library and benchmark that can be used by researchers as well industry practitioners. Our goal is to empower researchers and engineers with an end-to-end toolkit to facilitate developing FL algorithms and provide a platform to fairly compare different design choices and the tradeoffs. We welcome feedback from the readers, and will continuously update FedML to support the research of the federated learning community.

---

\(^5\)https://www.wandb.com/

\(^6\)https://github.com/FedML-AI/FedML/tree/master/benchmark
References


A The Taxonomy of Research Areas and a Comprehensive Publication List

Table 7: The taxonomy of research areas in federated learning and related publication statistics

<table>
<thead>
<tr>
<th>Research Areas</th>
<th>Approaches or Sub-problems (# of Papers)</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Challenges</td>
<td>Distributed Optimization (56), Non-IID and Model Personalization (49), Vertical FL (8), Decentralized FL (3), Hierarchical FL (7), Neural Architecture Search (4), Transfer Learning (11), Semi-Supervised Learning (3), Meta Learning (3)</td>
<td>144</td>
</tr>
<tr>
<td>Trustworthiness</td>
<td>Preserving Privacy (35), Adversarial Attack (43), Fairness (4), Incentive Mechanism (5)</td>
<td>87</td>
</tr>
<tr>
<td>System Challenges</td>
<td>Communication-Efficiency (27), Computation Efficiency (17), Wireless Communication and Cloud Computing (71), FL System Design (19)</td>
<td>134</td>
</tr>
<tr>
<td>Models and Applications</td>
<td>Models (22), Natural language Processing (15), Computer Vision (3), Healthcare (27), Transportation (13), Other (21)</td>
<td>101</td>
</tr>
<tr>
<td>Common</td>
<td>Benchmark and Dataset (20), Survey (7)</td>
<td>27</td>
</tr>
</tbody>
</table>

From a comprehensive FL publication list: https://anonymous

B Programming Interface

The goal of the FedML programming interface is to provide a simple user experience and allows users to build distributed training applications (e.g. to design customized message flow and topology definitions) by only focusing on algorithmic implementations while ignoring the low-level communication backend details.

(a) Training procedure-oriented programming

```python
import torch.distributed as dist
... def train():
    model = Net()
    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
    num_batches = ceil(len(train_set) / batch_size)
    for epoch in range(epoch_num):
        for data, target in train_set:
            optimizer.zero_grad()
            output = model(data)
            loss = F.nll_loss(output, target)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
    return epoch_loss / num_batches
```

(b) Worker-oriented programming

```python
class FedAvgClientManager(WorkerManager):
    def __init__(self, topology_manager, trainer):
        self.topology_manager = topology_manager
        self.trainer = trainer
        self.register_message_receive_handlers(self):
            self.send_message(C2S_message_A(self, msg_params))
            def send_message_to_client(receive_id, params_B):
                dist.all_reduce(param.grad.data, op=dist.reduce_op.SUM)
            def handle_C2S_message_A(receive_id, params_B):
                for param in model.parameters():
                    dist.all_reduce(param.grad.data, op=dist.reduce_op.SUM)
                    param = float(param.grad.data / dist.get_world_size())
            def __train__(self):
                for receive_id in self.topology_manager.get_neighbor_ids():
                    self.send_message_to_client(receive_id, params_B)
```

Figure 4: A worker-oriented programming design pattern of FedML

**Worker-oriented programming.** As shown in Figure 4(b), FedML provides the worker-oriented programming design pattern, which can be used to program the worker behavior when participating in training or coordination in the FL algorithm. We describe it as worker-oriented because its counterpart, the standard distributed training library (as the torch.distributed example shown in Figure 4(a)), normally completes distributed training programming by describing the entire training procedure rather than focusing on the behavior of each worker.

With the worker-oriented programming design pattern, the user can customize its own worker in FL network by inheriting the WorkerManager class and utilizing its predefined APIs `register_message_receive_handler` and `send_message` to define the receiving and sending messages without considering the underlying communication mechanism (as shown in the highlighted blue box in Figure 4(b)). Conversely, existing distributed

3More details can be found at https://pytorch.org/tutorials/intermediate/dist_tuto.html
training frameworks do not have such flexibility for algorithm innovation. In order to make the comparison clearer, we use the most popular machine learning framework PyTorch as an example. Figure 4(a) illustrates a complete training procedure (distributed synchronous SGD) and aggregates gradients from all other workers with the all_reduce messaging passing interface. Although it supports multiprocessing training, it cannot flexibly customize different messaging flows in any network topology. In PyTorch, another distributed training API, torch.nn.parallel.paraDistributedDataParallel, also has such inflexibility. Note that torch.distributed.rpc is a low-level communication back API that can finish any communication theoretically, but it is not user-friendly for federated learning researchers.

**Message definition beyond gradient and model.** FedML also considers supporting message exchange beyond the gradient or model from the perspective of message flow. This type of auxiliary information may be due to either the need for algorithm design or the need for system-wide configuration delivery. Each worker defines the message type from the perspective of sending. Thus, in the above introduced worker-oriented programming, the WorkerManager should handle messages defined by other trainers and also send messages defined by itself. The sending message is normally executed after handling the received message. As shown in Figure 4(b), in the yellow background highlighted code snippet, workers can send any message type and related message parameters during the `train()` function.

![Figure 5: Various Topology Definitions in Federated Learning](image)

**Topology management.** As demonstrated in Figure 5, FL has various topology definitions, such as vertical FL [50, 51, 52, 53, 54, 55, 56], split learning [57, 58], decentralized FL [59, 60, 61, 62], and Hierarchical FL [63, 64, 65, 66, 67, 68]. In order to meet such diverse requirements, FedML provides TopologyManager to manage the topology and allow users to send messages to arbitrary neighbors during training. Specifically, after the initial setting of TopologyManager is completed, for each trainer in the network, the neighborhood worker ID can be queried through the TopologyManager. In line 26 of Figure 4(b), we see that the trainer can query its neighbor nodes through the TopologyManager before sending its message.

**Trainer and coordinator.** We also need the coordinator to complete the training (e.g., in the FedAvg algorithm, the central worker is the coordinator while the others are trainers). For the trainer and coordinator, FedML does not over-design. Rather, it gives the implementation completely to the developers, reflecting the flexibility of our framework. The implementation of the trainer and coordinator is similar to the process in Figure 4(a), which is completely consistent with the training implementation of a standalone version training. We provide some reference implementations of different trainers and coordinators in our source code (Section 3.1).

**Privacy, security, and robustness.** While the FL framework facilitates data privacy [97] by keeping data locally available to the users and only requiring communication for model updates, users may still be concerned about partial leakage of their data which may be inferred from the communicated model (see, e.g., [126]). Aside from protecting the privacy of users’ data, another critical security requirement for the FL platform, especially when operating over mobile devices, is the robustness towards user dropouts. Specifically, to accomplish the aforementioned goals of achieving security, privacy, and robustness for FL, various cryptography and coding-theoretic approaches have been proposed to manipulate intermediate model data (see [127, 36]).

To facilitate rapid implementation and evaluation of data manipulation techniques to enhance security, privacy, and robustness, we include low-level APIs that implement common cryptographic primitives such as secrete

---

8It is recommended to use `torch.nn.parallel.paraDistributedDataParallel` instead of `torch.nn.DataParallel`. For more details, please refer to [https://pytorch.org/tutorials/intermediate/ddp_tutorial.html](https://pytorch.org/tutorials/intermediate/ddp_tutorial.html) and [https://pytorch.org/docs/master/notes/cuda.html#cuda-nn-ddp-instead](https://pytorch.org/docs/master/notes/cuda.html#cuda-nn-ddp-instead)

9[https://pytorch.org/tutorials/intermediate/rpc_tutorial.html](https://pytorch.org/tutorials/intermediate/rpc_tutorial.html)
sharing, key agreement, digital signature, and public key infrastructure. We also plan to include an implementation of Lagrange Coded Computing (LCC) [128]. LCC is a recently developed coding technique on data that achieves optimal resiliency, security (against adversarial nodes), and privacy for any polynomial evaluations on the data. Finally, we plan to provide a sample implementation of the secure aggregation algorithm [127], using the above APIs.

In standard FL settings, it is assumed that there is no single central authority that owns or verifies the training data or user hardware, and it has been argued by many recent studies that FL lends itself to new adversarial attacks during decentralized model training [22, 25, 20, 129, 130]. Several robust aggregation methods have been proposed to enhance the robustness of FL against adversaries [25, 131, 132].

To accelerate generating benchmark results on new types of adversarial attacks in FL, we include the latest robust aggregation methods presented in literature i.e. (i) norm difference clipping [25]; weak differential private (DP) [25]; (ii) RFA (geometric median) [131]; (iii) KRUM and (iv) MULTI-KRUM [132]. Our APIs are easily extendable to support newly developed types of robust aggregation methods. On the attack end, we observe that most of the existing attacks are highly task-specific. Thus, it is challenging to provide general adversarial attack APIs. Our APIs support the backdoor with model replacement attack presented in [22] and the edge-case backdoor attack presented in [130] to provide a reference for researchers to develop new attacks.

C Benchmark

C.1 Details of Supported Algorithms

**Federated Averaging (FedAvg).** FedAvg [47] is a standard federated learning algorithm that is normally used as a baseline for advanced algorithm comparison. We summarize the algorithm message flow in Figure 2(a).

Each worker trains its local model for several epochs, then updates its local model to the server. The server aggregates the uploaded client models into a global model by weighted coordinate-wise averaging (the weights are determined by the number of data points on each worker locally) and then synchronizes the global model back to all workers. In our FedML library, based on the worker-oriented programming, we can implement this algorithm in a distributed computing manner. We suggest that users start from FedAvg to learn using FedML.

**Decentralized FL.** We use [59], a central server free FL algorithm, to demonstrate how FedML supports decentralized topology with directed communication. As Figure 2(b) shows, such an algorithm uses a decentralized topology, and more specifically, some workers do not send messages (model) to all of their neighbors. The worker-oriented programming interface can easily meet this requirement since it allows users to define any behavior for each worker.

**Vertical Federated Learning (VFL).** VFL or feature-partitioned FL [120] is applicable to the cases where all participating parties share the same sample space but differ in the feature space. As illustrated in Figure 2(c), VFL is the process of aggregating different features and computing the training loss and gradients in a privacy-preserving manner to build a model with data from all parties collaboratively [133, 51, 134, 135]. The FedML library currently supports the logistic regression model with customizable local feature extractors in the vertical FL setting, and it provides NUS-WIDE [136] and lending club loan [137] datasets for the experiments.

**Split Learning.** Split Learning is a computing and memory-efficient variant of FL introduced in [57, 58] where the model is split at a layer, and the parts of the model preceding and succeeding this layer are shared across the worker and server, respectively. Only the activations and gradients from a single layer are communicated to the worker and server, respectively. Only the activations and gradients from a single layer are communicated to the server. The server aggregates the uploaded client models into a global model by weighted coordinate-wise averaging (the weights are determined by the number of data points on each worker locally) and then synchronizes the global model back to all workers. In our FedML library, based on the worker-oriented programming, we can implement this algorithm in a distributed computing manner. We suggest that users start from FedAvg to learn using FedML.

**Federated Neural Architecture Search (FedNAS).** FedNAS [70] is a federated neural architecture search algorithm [142] that enables scattered clients to collaboratively search for a neural architecture. FedNAS differs from other FL algorithms in that it exchanges information beyond gradient even though it has a centralized topology similar to FedAvg.

C.2 Details of Datasets

**Federated EMNIST:** EMNIST [143] consists of images of digits and upper and lower case English characters, with 62 total classes. The federated version of EMNIST [46] partitions the digits by their author. The dataset has natural heterogeneity stemming from the writing style of each person.

**CIFAR-100:** Google introduced a federated version of CIFAR-100 [124] by randomly partitioning the training data among 500 clients, with each client receiving 100 examples [2]. The partition method is Pachinko Allocation Method (PAM) [144].
Shakespeare: [47] first introduced this dataset to FL community. It is a dataset built from *The Complete Works of William Shakespeare*. Each speaking role in each play is considered a different device.

StackOverflow [125]: Google TensorFlow Federated (TFF) team maintains this federated dataset, which is derived from the Stack Overflow Data hosted by kaggle.com. We integrate this dataset into our benchmark.

CIFAR-10 and CIFAR-100. CIFAR-10 and CIFAR-100 [124] both consists of $32 \times 32$ color images. CIFAR-10 has 10 classes, while CIFAR-100 has 100 classes. Following [145] and [5], we use latent Dirichlet allocation (LDA) to partition the dataset according to the number of workers involved in training in each round.

CINIC-10. CINIC-10 [146] has 4.5 times as many images as that of CIFAR-10. It is constructed from two different sources: ImageNet and CIFAR-10. It is not guaranteed that the constituent elements are drawn from the same distribution. This characteristic fits for federated learning because we can evaluate how well models cope with samples drawn from similar but not identical distributions.

### C.3 Lack of Fair Comparison: Diverse Non-I.I.D. Datasets and Models

<table>
<thead>
<tr>
<th>Conference</th>
<th>Paper Title</th>
<th>dataset</th>
<th>partition method</th>
<th>model</th>
<th>worker/device number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICML 2019</td>
<td>Agnostic Federated Learning [147]</td>
<td>UCI Adult Census dataset</td>
<td>-</td>
<td>fully connected neural network</td>
<td>10</td>
</tr>
<tr>
<td>ICML 2019</td>
<td>Bayesian Nonparametric Federated Learning of Neural Networks [145]</td>
<td>MNIST</td>
<td>Dir(0.5)</td>
<td>1 hidden layer neural networks</td>
<td>10</td>
</tr>
<tr>
<td>ICML 2019</td>
<td>Federated Learning with Only Positive Labels [149]</td>
<td>CIFAR-10/100</td>
<td>1 class / 1 client</td>
<td>ResNet-9</td>
<td>-</td>
</tr>
<tr>
<td>ICML 2020</td>
<td>Federated Learning with Matched Averaging [13]</td>
<td>EMNIST</td>
<td>1 class / 1 client</td>
<td>Fully connected network</td>
<td>-</td>
</tr>
<tr>
<td>ICML 2020</td>
<td>SCAFFOLD: Stochastic Controlled Averaging for Federated Learning[10]</td>
<td>EMNIST</td>
<td>1 class / 1 client</td>
<td>Fully connected network</td>
<td>-</td>
</tr>
<tr>
<td>ICML 2020</td>
<td>From Local SGD to Local Fixed-Point Methods for Federated Learning [11]</td>
<td>CIFAR-10</td>
<td>1 class / client</td>
<td>ResNet-32</td>
<td>-</td>
</tr>
<tr>
<td>ICML 2020</td>
<td>Acceleration for Compressed Gradient Descent in Distributed and Federated Optimization[52]</td>
<td>CIFAR-10</td>
<td>1 class / client</td>
<td>Fully Connected Net</td>
<td>-</td>
</tr>
<tr>
<td>ICLR 2020</td>
<td>Federated Learning with Matched Averaging [13]</td>
<td>CIFAR-10</td>
<td>1 layer LSTM</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ICLR 2020</td>
<td>Fair Resource Allocation in Federated Learning [153]</td>
<td>Synthetic dataset use LR</td>
<td>natural non-IID</td>
<td>multinomial logistic regression</td>
<td>10</td>
</tr>
<tr>
<td>ICLR 2020</td>
<td>DBA: Distributed Backdoor Attacks against Federated Learning[120]</td>
<td>DBA: Distributed Backdoor Attacks against Federated Learning</td>
<td>MNIST</td>
<td>2 conv and 2 fc</td>
<td>10</td>
</tr>
</tbody>
</table>

*Note: we will update this list once new publications are released.*
D IoT Devices

Currently, we support two IoT devices: Raspberry Pi 4 (Edge CPU Computing) and NVIDIA Jetson Nano (Edge GPU Computing).

D.1 Raspberry Pi 4 (Edge CPU Computing - ARMv7l)

Raspberry Pi 4 Desktop kit is supplied with:

- Raspberry Pi 4 Model B (2GB, 4GB or 8GB version)
- Raspberry Pi Keyboard and Mouse
- 2 micro HDMI to Standard HDMI (A/M) 1m Cables
- Raspberry Pi 15.3W USB-C Power Supply
- 16GB NOOBS with Raspberry Pi OS microSD card

For more details, please check this link: https://www.raspberrypi.org/products/raspberry-pi-4-desktop-kit.

D.2 NVIDIA Jetson Nano (Edge GPU Computing)

NVIDIA® Jetson Nano™ Developer Kit is a small, powerful computer that lets you run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. All in an easy-to-use platform that runs in as little as 5 watts.

For more details, please check this link: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.