State of the Union: Towards Reproducible Performance Evaluations in Federated Learning

Ningxin Su*, Baochun Li†, Bo Li‡
*Hong Kong University of Science and Technology (Guangzhou)
†University of Toronto
‡Hong Kong University of Science and Technology

***Abstract*—Federated learning (FL) is a privacy-motivated paradigm for distributed training of deep learning models, as it allows a large number of clients to collaboratively train a shared global model without centralizing their private data. Since its debut with Federated Averaging as the first server aggregation algorithm, many FL algorithms have been proposed to improve performance. Yet, these algorithms are rarely benchmarked and compared under the same open-source framework and controlled configurations, and their performance claims can be difficult to substantiate in fair and reproducible studies.**

In this paper, we evaluate a curated and representative collection of FL algorithms in the same open-source benchmarking framework so that they can be compared fairly at scale in a reproducible fashion. To achieve this objective, we present *Plato*, an open-source FL research framework that we have designed and implemented from scratch. With *Plato*, we evaluate and compare algorithms spanning (1) server aggregation; (2) client training customization; (3) client selection, in both synchronous and asynchronous settings; (4) personalized federated learning; and (5) communication efficiency and payload processing. Across diverse experimental scenarios (tasks, client populations, and data distributions), we report findings and practical insights, including pitfalls and confounding factors that can lead to misleading conclusions if not reported carefully. Under our unified experimental settings and time model, Federated Averaging with random client selection remains a strong baseline and is often competitive with more complex alternatives.

***Index Terms*—Federated Learning, Framework, Reproducibility, Comparison Studies**

I. Introduction

Federated learning is a privacy-motivated distributed machine learning paradigm that has attracted substantial research attention since its debut [1]. The essential advantage of federated learning is that clients — ranging from enterprises to edge devices such as mobile phones — can keep their private data locally, while still using such data for collaborative training of a shared global model. To achieve this objective, clients conduct local training with their private data and send their local updates to a server, which then aggregates these updates to generate an improved global model. The training process progresses iteratively in *communication rounds*, until the convergence of the global model. While federated learning reduces the need to centralize raw training data, additional protections (e.g., differential privacy or secure aggregation) are typically required to provide formal privacy guarantees; in this paper, we focus on performance-oriented evaluations and treat privacy/security as important complementary considerations.

In the general context of federated learning, researchers have ventured into a multitude of research directions in the literature, but most of which can roughly be separated into two tracks. The first questioned whether keeping private data locally can indeed preserve data privacy, by proposing a variety of attacks under various assumptions and threat models [2], along with their corresponding defences using, as examples, differential privacy [3] and homomorphic encryption [4] techniques. The second track of research focused on improving the performance of federated learning, by proposing new algorithms that can improve the training performance of federated learning. In this paper, we are mainly concerned with the latter track with a focus on performance, which can be best evaluated with the time it takes for the global model to converge, and the validation accuracy that the global model converged to.

Though performance-oriented algorithms in the literature are numerous and cover a broad range of topics, the following five categories are representative and have been studied extensively:

- ▷ *Server aggregation* algorithms were proposed for the server to adjust its weights when aggregating client updates to produce a shared global model. The very first server aggregation algorithm was Federated Averaging (`FedAvg`), where the weight used to aggregate an update corresponded to the number of local samples used for its training. Improved server aggregation algorithms sought to revise these aggregation weights to better accommodate non-i.i.d. data distributions across clients, typically called *data heterogeneity*.
- ▷ *Client training loop customization* algorithms were designed to calibrate local updates on the clients during local training, with the hope that they could contribute more to improving the global model.
- ▷ *Client selection* algorithms attempted to improve training performance by selecting clients with higher data quality in each round, rather than randomly.
- ▷ *Personalized federated learning* algorithms sought to train models that are *personalized* to produce high validation accuracies on the specific data at each client, while still benefiting from collaborative training over a shared global model.
- ▷ *Communication-efficient federated learning* algorithms aimed to reduce end-to-end training cost by decreasing transmitted bytes (e.g., compression, sparsification, quan-

tization, and structured updates), which can improve time-to-accuracy when communication becomes a bottleneck.

Most works in the literature claimed substantial performance improvements over Federated Averaging (FedAvg). But are these claims trustworthy and transferable across workloads and experimental settings? Two factors come into play when we attempt to answer this question.

First, which performance metric should be best used to evaluate the training performance in federated learning? Existing solutions typically use the number of communication rounds or the number of transmitted local updates, but we are strongly convinced that these are far from ideal when evaluating the performance of training sessions. The time it takes in each communication round to train depends on a wide variety of factors, such as the number of epochs used for local training, the number of data samples used in each epoch, as well as how the training loop is designed. Instead of the number of rounds or updates, the *wall-clock time* elapsed during training before convergence is reached is a simple and intuitive metric, that we believe should be used as the performance metric to make fair comparisons. In addition, the *validation accuracy that the global model converged to* may also be of interest, as it is the ultimate goal of model training.

Second, how can we evaluate the performance of existing algorithms and compare them fairly with one another? Existing solutions may not have corresponding open-source implementations; and even if they do, a wide variety of factors — including the choice of datasets, models, data distributions across clients, varying training speeds across clients (called *system heterogeneity* in the literature), and even random seeds — affect the results of these performance evaluations. In the literature, each of the proposed algorithms was evaluated in its own proprietary simulation or emulation testbed, which may be biased in its own favor. When we seek to compare existing algorithms in the literature, *reproducible* experiments with an open-source software framework and the same configuration settings are desirable, even necessary, for *fair* performance comparisons.

In this paper, we first present Plato, our open-source federated learning research framework, designed and implemented from scratch over the recent years. Plato is, first and foremost, designed to be a *reproducible* benchmarking utility, so that researchers can use it to reproduce the results of existing algorithms in the literature, and compare them fairly with one another. Maximizing fairness while making these comparisons is also a primary design goal, where we have made every effort to ensure that the same configuration settings, including datasets, models, data distributions, even random number generators, are used for all algorithms under comparison.

With Plato, we present a curated collection of comparison studies of representative algorithms across these categories. Under our unified experimental settings and time model, many algorithms do not consistently outperform FedAvg with random client selection [1]; in some cases we observe underperformance. We report our findings and insights from running these experiments, identify several practical pitfalls, and discuss root causes and confounding factors (e.g., settings differences and reproduction uncertainty) that can lead to misleading conclusions if not reported carefully.

Scope and limits. This paper focuses on classic FL benchmark workloads (small/medium models and standard datasets, plus a lightweight language modeling workload) under controlled and reproducible configurations. Although Plato supports LLM model training and adversarial attack algorithms, they are not considered within the scope of this paper. instead, we aim to provide a reproducible framework and methodology, and to document where (and why) results can differ from prior reports.

Highlights of our original contributions in this paper are as follows. *First*, we present Plato, our open-source reproducible federated learning research framework, designed and implemented from scratch to emulate real-world scenarios. *Second*, we extensively evaluate state-of-the-art federated learning algorithms, and make fair comparisons with respect to their performance. *Finally*, based on our experimental results, we look into the reasons behind varying levels of performance, and present our own insights on improving the performance of federated learning, concluding with a cautionary note in favour of federated averaging and random client selection.

II. Preliminaries and Related Work

Preliminaries. Federated Averaging (FedAvg) is a canonical baseline in federated learning (FL): in each round, the server selects a subset of clients, broadcasts the current global model, and aggregates client updates to produce the next global model. Compared to centralized training, FL faces data heterogeneity (non-i.i.d. client data) [5], [6], system heterogeneity (stragglers), and communication bottlenecks; these factors motivate algorithm families that change aggregation, local training, participation/scheduling, and/or personalization. Across this space, the same algorithmic idea can lead to different *practical* outcomes depending on (i) how many clients participate, (ii) how much local work is done per round, and (iii) whether training is synchronous or asynchronous. These choices directly affect both statistical performance and the cost of reaching a target quality under realistic hardware constraints.

Representative algorithm families. We focus on widely used, representative algorithms: server aggregation beyond FedAvg (e.g., FedAdp [7], FedAtt [8], and distillation-based fusion [9]); client training-loop customization (e.g., FedProx [10], SCAFFOLD [11]); client selection/scheduling (e.g., Oort [12], and Pisces for asynchronous settings [13]); and personalized FL (e.g., FedRep [14], FedBABU [15], LG-FedAvg [16], Ditto [17], and Per-FedAvg [18]).

Conceptually, aggregation methods adjust how updates are fused to stabilize training under heterogeneity; training-loop methods modify the local objective/updates to reduce client drift; selection/scheduling methods prioritize which clients contribute under resource constraints; and personalized FL targets client-specific performance (often changing what is shared and when personalization happens). While these algorithms are often introduced as "orthogonal," in practice they frequently interact: for example, changing the training loop can

also change communication volume, and asynchronous selection/aggregation policies can change the effective optimization dynamics via staleness. Our evaluation suite (Section IV) therefore structures results around these algorithm families and reports both statistical and cost-to-target outcomes under a unified protocol.

Recent directions and practical considerations. Recent works explore collaboration graphs for personalization (pFedGraph [19]), contrastive/self-supervised objectives (e.g., MOON [20]), adaptive local aggregation methods (e.g., FedALA [21]), and federated active learning / data-selection methods (e.g., Active FL [22] and CHASe [23]), as well as application-driven settings such as federated recommendation (e.g., LightFR [24]) and security issues in that setting (e.g., popular-item embedding attacks in federated recommendation [25]). Along the privacy-preservation track, privacy leakage in federated language model training (e.g., FILM [26]) highlight privacy and workload constraints that can confound comparisons. These trends also reinforce our claim that benchmark results should be interpreted alongside what is being counted as "cost" (e.g., communication rounds, wall-clock time, or bytes transferred), and whether the evaluation protocol is aligned with the original papers' settings or with a unified, controlled configuration for apples-to-apples comparison.

Benchmarking toolkits and positioning. Existing open-source initiatives include FedScale for model-and-system benchmarking at scale with trace-driven time-to-accuracy reporting [27], FedML as an end-to-end FL library/platform [28], FedEval as an evaluation framework [29], PFLlib and FL-bench as curated algorithm collections and runnable benchmark codebases [30], [31], pFL-Bench for standardized, containerized personalized-FL evaluation (including system overhead) [32], and HtFLlib for heterogeneity-focused workloads [33].

Plato complements these efforts by emphasizing ① reproducible apples-to-apples comparisons under unified configurations, ② explicit simulated wall-clock time support (sync/async), and ③ a strategy-based composition API that isolates extension points for clients, servers, trainers, and payload processors. Table I is a positioning aid rather than an exhaustive benchmark survey: its markers are coarse presence indicators for the capabilities we rely on in this paper. In this paper, we use these design choices to ① separate *framework support* from *paper-evaluated* algorithms/workloads, and ② make time-to-target comparisons feasible even when many "logical" clients must be emulated on limited physical hardware.

Benchmarking challenges. Reported results in FL are often sensitive to uncontrolled factors, including client sampling and data partitioning, per-round local work, hardware and batch sizes, and whether slow clients are dropped or waited for. As a result, frameworks that expose these choices explicitly (and record them as artifacts) are important not only for fair comparison but also for understanding *why* a method appears better under one cost model but not another.

III. Plato: Towards Reproducible Performance Evaluations in Federated Learning

We have designed and built Plato[1], an open-source benchmarking framework for federated learning research. Before we started from scratch six years ago, we considered using one of the existing frameworks, such as FedML [28] (most other alternatives in Table I did not exist at the time). Yet, several features that are essential for scalable and reproducible benchmarking on limited hardware were not readily available. In this section, we present what these desirable features are, why they were important enough that we needed to build a new framework entirely from scratch, and how Plato achieved them.

A. Architectural Design

When we first started building Plato in November 2020, we incorporated the following design goals:

- ♢ *Simplicity and ease of use.* Plato should be simple to use, even by inexperienced researchers who are not well versed in federated learning. Existing algorithms in the literature should be easily implementable using its application programming interface (API), which should be well documented.
- ♢ *Availability of datasets, models, and training loops.* It should be able to support a wide variety of datasets and models out of the box across a collection of popular tasks, such as image classification, object detection, and natural language processing. These datasets and models promote fairness when we compare the performance of different algorithms.
- ♢ *Reproducibility.* Though reproducibility in machine learning research is challenging, every effort should be taken — starting from seeding the random number generators — to maximize reproducibility with the same experimental setup.
- ♢ *Support for both real-world implementation and emulation with limited hardware.* While the framework should support real-world network communication between clients and the server, it should ideally also be able to support emulating a realistic number of clients when GPU resources are severely constrained.
- ♢ *Asynchronous federated learning.* In both real-world implementation and emulation with limited hardware, Plato should provide native support for asynchronous federated learning, where the server only needs to wait for a subset of clients before proceeding with its aggregation. This is more performant in practice.

After six years of work, we were able to achieve all of these design goals. Fig. 1 shows an overview of Plato's architectural design. Plato adopts a strategy-based composition API (with callbacks and processor pipelines for cross-cutting concerns) so that algorithms can be implemented by swapping a small number of strategy components while reusing the rest of the stack. Plato's primary machine learning backend is PyTorch

[1]Available at `https://github.com/TL-System/plato`, and named after the great philosopher.

TABLE I
A COMPACT FEATURE MATRIX COMPARING Plato WITH REPRESENTATIVE FL BENCHMARKING TOOLKITS.

Toolkit	Explicit simulated wall-clock time	Async support	Limited-hardware emulation	Strategy-based extensibility	Cross-silo support	Privacy/security hooks	Reproducibility artifacts
Plato	√	√	√	√	√	√	√
FedScale [27]	√	×	×	√	√	×	×
FedML [28]	√	√	√	√	√	×	×
FedEval [29]	×	×	×	√	×	√	×
PFLlib [30] / FL-bench [31]	×	×	×	√	√	×	×
pFL-Bench [32]	×	×	×	√	√	×	√
HtFLlib [33]	×	×	×	√	√	×	×

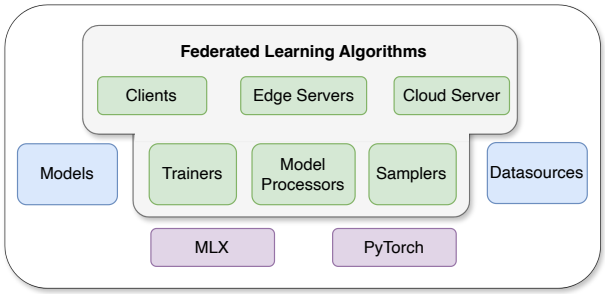


Fig. 1. An overview of Plato's architectural design. As a benchmarking framework, it is designed from the onset to be agnostic to the underlying machine learning backend, with PyTorch as the primary backend and MLX as an alternative backend. Existing algorithms can customize most of the components, including clients, edge or cloud servers, trainers, data distribution samplers, and model processors.

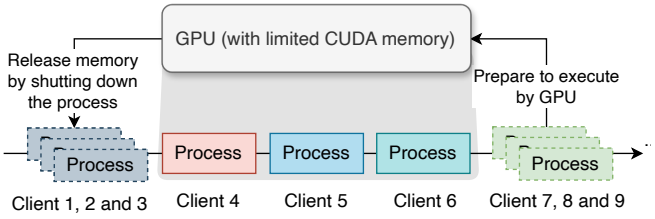


Fig. 2. By running each client's training loop in its own ephemeral process, we may run client trainers in batches. This allows us to select a potentially unlimited number of clients in each round, even with only one GPU.

(with MLX as an alternative backend), and it integrates with common model and dataset ecosystems (e.g., `Torchvision`, `HuggingFace`, and `timm`) to reduce friction when constructing reproducible workloads.

Cross-device and cross-silo. Plato supports both *cross-device* (a large population of clients with partial participation) and *cross-silo* style training (a smaller, more stable set of silos) via configuration and deployment options. In this paper, our primary evaluation focuses on cross-device-style settings; however, cross-silo configuration has been a core part of the framework and reproducibility pathway since its early days. Cross-silo training can be enabled by declaring the number of silos and switching from a cross-device client population to edge-level participation.

B. Scalability while working with Limited Hardware

Before we consider reproducibility in our benchmark experiments, it is important to first discuss how our experiments can best model what occurs in practice. Plato is fully capable of running a real-world server in the cloud, with clients connecting to it using the `WebSocket` two-way communication protocol over HTTPS. That said, for the sake of better reproducibility, it is more likely that benchmarking runs will take place on a physical machine with limited CPU and GPU hardware.

In the literature, most experiments related to federated learning used small-scale experiments, selecting only around 10-20 clients in each round. This is mostly due to the lack of GPU memory on consumer-grade GPUs: an NVIDIA RTX A4500 GPU, for example, only has 24 GB of GPU memory, accommodating no more than 7 clients training a `ResNet-18` model over the `CIFAR-10` dataset, a typical benchmark in the literature.

In contrast, Plato employs a delicate algorithm that spawns a new ephemeral process for every client to run its training loop, called a `trainer` in the framework. This ensures that after each client process finishes training on the GPU, its occupied GPU memory will be immediately and automatically released by the operating system when the ephemeral process is terminated, regardless of how the trainer is implemented. This way, we can run client trainers *a batch at a time*, as we show in Fig. 2. In addition, Plato is able to ① take full advantage of all the GPUs in a physical machine, maximizing GPU utilization; and ② guarantee that the number of client processes spawned is exactly the same as the number of trainers concurrently running, minimizing the demand for *physical memory*. As an example of how Plato efficiently utilized limited hardware to run experiments with larger scales, in our benchmarking experiments using `ResNet-18` and `CIFAR-10`, a federated learning round involving 100 selected clients finished within 10 minutes and 15 batches, with only one NVIDIA RTX A4500 GPU.

C. Extensibility via Composable Strategies and Callbacks

Strategy-based composition. Plato makes extensibility primarily strategy-driven rather than relying on deeply nested subclasses. Clients, servers, and trainers are assembled from small, interchangeable strategy objects that share state via lightweight context containers. Concretely, each client owns a `ComposableClient` that orchestrates five strategies (lifecycle, payload, training, reporting, and communication), while servers accept `AggregationStrategy` and `ClientSelectionStrategy` components, and trainers use `ComposableTrainer` with interchangeable loss/optimizer/step/scheduler/model-update/dataloader/testing strategies. The default strategy stacks reproduce legacy behavior, so implementing an algorithm typically swaps only one or two strategies while reusing the rest.

Callbacks and processors. For cross-cutting concerns (logging, metrics, tracing, early stopping), Plato can be extended with callbacks, which are triggered from strategies via the callback handler. Separately, inbound/outbound processor pipelines can transform payloads (e.g., compression or encryption) without entangling the learning algorithm.

Algorithm families and extension points. Table II summarizes the major algorithm families supported by Plato and their primary extension points, distinguishing between core strategies/APIs and reference example workloads.

How to implement a new method in Plato**.** From an FL researcher's perspective, implementing a new method in Plato typically consists of identifying *what changes* relative to a baseline (e.g., `FedAvg`) and then swapping the smallest corresponding strategy component(s):

- ▷ If you change **aggregation**, implement an `AggregationStrategy`.
- ▷ If you change **who participates / scheduling**, implement a `ClientSelectionStrategy` (often together with a client `ReportingStrategy` to export utility/staleness signals).
- ▷ If you change the **local objective or training loop**, implement trainer strategies (e.g., `LossCriterionStrategy`, `TrainingStepStrategy`, `ModelUpdateStrategy`) and/or a client `TrainingStrategy`.
- ▷ If you change **payloads / communication / privacy**, use `PayloadStrategy`/`CommunicationStrategy` and processor pipelines (e.g., compression/encryption), with callbacks for logging and instrumentation.

In practice, new methods often touch only one or two of these extension points; the remaining strategy stack can remain unchanged, making it easier to implement, validate, and compare methods under consistent settings.

Mapping to recent "hot topics." The above extension points also cover several recent directions raised in the reviews. Federated distillation/fusion methods (e.g., `FedDF`) primarily change what is communicated (e.g., logits, teacher outputs) and therefore map to payload strategies, processor pipelines, and trainer strategies (with aggregation changes when required). This is the design path we use for the `FedDF` overlays in Fig. 3. Federated large-model training and parameter-efficient tuning (e.g., adapter-style updates) primarily map to trainer strategies and payload strategies that redefine which parameters are trained and transmitted. Finally, "agentic" client loops (multi-step local decision-making and richer client reporting) map naturally to customized training and reporting strategies. These are supported as framework extension points, even when a given direction is not evaluated as a workload in this paper.

D. Reproducibility

Reproducibility involving machine learning is hardly straightforward to achieve: a plethora of utilities and frameworks are built to achieve reproducible solutions, and yet, many still believe that it is at best elusive to attain in practice. This is due to the nature of training deep neural network models, where a variety of random number generators from different sources are utilized to produce many essential elements in the training process, from initial model weights to parameterizing model layers such as dropout layers. This is further exacerbated by federated learning, where even more randomness is incorporated, such as in client selection algorithms.

Beyond random number generation, we wish to use the wall-clock time to evaluate the performance of federated learning algorithms; but the elapsed wall-clock time can vary quite substantially across runs, as the training epochs at each client need to run on actual GPUs. While statistical measures such as confidence intervals can in theory be used in experimental results, they are inconvenient since each run involving even only a modest number of clients can take many hours to complete on limited hardware. For these reasons, we are at least obliged to attempt maximizing reproducibility when designing `Plato` as a benchmarking framework, so that different algorithms can be compared with one another fairly.

Consistent configurations. Towards maximizing reproducibility, we begin by making sure that all evaluations use the same dataset, model, training loops, and hyperparameters. This sounds like a trivial exercise; but many existing works have failed to pay attention to some of the details. A case in point is the parameters used in the training loop, which at least include the optimizer, the learning rate scheduler, and the loss criterion, each of which is related to several influential parameters. In addition, as a variety of data samplers are used for simulating the data distribution across clients, the parameters used in these samplers must be kept consistent as well. In `Plato`, all the parameters that may affect reproducibility are centrally specified in **TOML** configuration files (with lightweight include semantics), and command-line flags take precedence over file settings. To mitigate human error preparing these configurations, similar to header files in programming, different configurations can include shared segments of configurations as well so that these shared segments are guaranteed to be consistent across-the-board.

Random seeds. As we stated, FL training sessions involves a substantial amount of randomization: the server randomly selects a number of clients, each client randomly samples the dataset with an i.i.d. or non-i.i.d. probability distribution,

TABLE II
MAJOR ALGORITHM FAMILIES SUPPORTED BY Plato AND THEIR PRIMARY EXTENSION POINTS.

Algorithm family	Primary extension points	Core (built-in strategies/APIs)	Examples (templates)
Server aggregation (sync/async)	`AggregationStrategy` (`aggregate_*`)	FedAvg, FedAsync, FedBuff, FedNova	FedAdp; FedAtt; FedDF; FedAvgGAN; FedUnlearning; Knot
Client selection/scheduling	`ClientSelectionStrategy` + `ReportingStrategy`	RandomSelection, SplitLearningSequential, PersonalizedRatio	Oort/Pisces/AFL (reporting + selection)
Local training-loop customization	Trainer strategies + client `TrainingStrategy`	`ComposableTrainer`; FedProx/FedDyn	MOON; SCAFFOLD (control variates); FedMoS; gradient clipping/pruning
Personalization	Phase-aware aggregation + trainer `ModelUpdateStrategy`	Personalized FedAvg; ratio controls	FedALA/Ditto/FedRep/FedBABU/LG-FedAvg/Per-FedAvg
Communication efficiency and payload handling	`PayloadStrategy` `CommunicationStrategy` processor pipelines	Default payload + transport strategies; processor API	customized processor; split learning; model pruning
Privacy/security algorithms	Encrypted/hybrid aggregation + processors (encrypt/decrypt)	FedAvgHE; MPC runtime support	secure_aggregation; Maskcrypt
Attacks/robustness debugging	Trainer/client strategies + server aggregation strategies + callbacks	Strategy hooks + callback pipeline	gradient_leakage_attacks (DLG); attack_adaptive

and in asynchronous FL (which we shall discuss next), the simulated client speed needs to be randomly sampled from a distribution as well. In Plato, we place an emphasis on how random seeds for random number generators (RNG) can be initially specified, and how they can be protected against influences from third-party frameworks, which tend to use the same generators. Before functions in third-party frameworks are called, we use `random.getstate()` to save the current state of the RNG, and immediately after returning from these functions, we use `random.setstate()` to restore the previous state of the RNG. This is similar in design of saving and restoring states during context switching across operating system kernel threads.

Reproducible mode. With the basic algorithms above in place, Plato is able to (optionally) enable its *reproducible* mode, which enables fully reproducible experiments where the same set of clients and data samples are selected across runs. This ensures that a global model's convergence trajectory remains consistent across runs as much as possible.

Drawing an analogy from photography, photographers compare lenses from different brands using the same camera body to discern differences. In a similar vein, with Plato as a benchmarking tool and a focus on reproducibility, fair and more convincing conclusions can be drawn when comparing different algorithms later in this paper.

E. Simulating Wall-Clock Time

Existing performance evaluations in the FL literature often report the number of rounds (*e.g.* [34]) or the number of updates (*e.g.* [35]) until reaching a target accuracy. These metrics are easy to collect, but can be misleading when different algorithms change the amount of work per round (e.g., extra local epochs, auxiliary state, or larger payloads). We therefore focus on the *wall-clock time to reach a target* as a practical metric. In real deployments, this is a direct measurement. In Plato's *emulation* mode (running many logical clients on limited hardware by batching trainers, Section III-B), we must *simulate* wall-clock time.

What is included. For each client update, we model its end-to-end completion time as the sum of ① client-side computation time (local training and any local processing), ② network transfer time for inbound/outbound payloads, and ③ server-side aggregation overhead. We exclude one-time setup costs (e.g., dataset download, initial model fetch) unless otherwise stated, so that the metric reflects steady-state training progress.

Synchronous FL. Let $\mathcal{S}_t$ be the selected clients in round t, and let $T_i(t)$ denote the end-to-end time for client i's update in that round (client computation + communication under a chosen network model). In synchronous FL, the round duration is dominated by the slowest selected client:

$$T_{\text{round}}(t) = \max_{i \in \mathcal{S}_t} T_i(t) + T_{\text{server}}(t), \tag{1}$$

where $T_{\text{server}}(t)$ accounts for aggregation and bookkeeping. The simulated elapsed time is the cumulative sum of $T_{\text{round}}(t)$ until convergence/target accuracy.

Asynchronous FL (event-driven emulation). In asynchronous FL, the server aggregates after receiving only the first $n_{\min}$ updates (possibly with a staleness bound), without waiting for all selected clients. In real deployments, update arrival order is defined by actual completion times. In emulation, however, trainers are executed in GPU batches, so the server cannot observe arrival order *online* while the batch is still running. Plato therefore attaches a *simulated finish time* to each report and maintains a priority queue of completed reports, sorted by finish time. Once enough reports are available, the server pops the earliest $n_{\min}$, advances simulated time to their completion boundary, aggregates, and immediately schedules new clients. Slower reports remain queued and may be aggregated later if they satisfy staleness rules.

Algorithm 1 summarizes the emulation logic, where $T^{\text{comp}}(r)$ denotes measured client-side compute/processing

Algorithm 1 Emulated async wall-clock time with batching

$t \leftarrow 0$, $PQ \leftarrow \emptyset$
while not reached target **do**
  $\mathcal{S} \leftarrow$ ChooseClients(n_s); $\mathcal{R} \leftarrow$ RunTrainersInBatches$(\mathcal{S})$
  for all $r \in \mathcal{R}$ **do**
    $f \leftarrow t + T^{\text{comp}}(r) + T^{\text{net}}(r)$
    Push$(PQ, (f, r))$
  end for
  $\mathcal{U} \leftarrow$ PopEarliest$(PQ, n_{\min}, K)$
  $t \leftarrow \max\{f : (f, \cdot) \in \mathcal{U}\}$
  Aggregate$(\{r : (\cdot, r) \in \mathcal{U}\})$
end while

time for report r and $T^{\text{net}}(r)$ denotes the network-model transfer time.

Worked micro-example. Suppose $n_s = 3$ and $n_{\min} = 2$ at simulated time $t = 0$, with client end-to-end times $T_1 = 5$s, $T_2 = 8$s, $T_3 = 12$s (including compute and communication). In synchronous FL, the round completes at 12s. In asynchronous FL, the server aggregates after the first two arrivals at $t = \max(5, 8) = 8$s, and the third report remains queued (finish time 12s) for possible use in a later aggregation subject to the staleness rule. This separation between *reported* updates (physically produced by batched trainers) and *simulated* finish events is what allows Plato to emulate asynchronous progress using limited hardware.

Limitations. Simulated time depends on the chosen network model and on measured/profiling of per-client training time; absolute wall-clock values may differ from a real multi-device deployment. We use it primarily for *within-setting* comparisons and explicitly report the assumptions used (client fraction, batching policy, bandwidth model, and staleness parameters).

IV. Evaluating Federated Learning Algorithms

We are now ready to evaluate the performance of representative categories of state-of-the-art FL algorithms in the literature, which has been one of the design goals of Plato as a reproducible evaluation framework. Our evaluation focuses on ① server aggregation, ② client training-loop customization, ③ client selection (including asynchronous settings), and ④ personalized federated learning; we additionally discuss communication efficiency and payload processing as complementary dimensions for reporting and extensibility.

A. Methodology: Validation, Metrics, and Artifacts

Reproducibility artifacts. For the sake of fair comparisons and reproducibility, a reproducibility artifact archive will be released as open-source, which includes ① the pinned framework version, ② the exact configuration bundle used to generate the reported figures, and ③ minimal scripts to reproduce at least one headline figure end-to-end, together with environment notes (PyTorch/CUDA versions) and fixed seeds.

Validation protocol and claim scoping. A key challenge in benchmarking is ensuring that re-implementations match the intended algorithms. For each algorithm, we follow a best-effort validation protocol: ① align code with equations/pseudo-code and stated hyperparameters; ② where public reference implementations exist, cross-check key update rules; ③ perform sanity checks (e.g., limiting cases that reduce to `FedAvg` when applicable); and ④ scope comparative claims to the settings we evaluate. When details or reference code are unavailable, we explicitly avoid attributing underperformance to the method itself, and instead describe results as "under our unified settings and time model." As a stop rule, if we cannot establish the minimum validation evidence for an algorithm (paper specification + basic invariants), we treat it as best-effort and avoid using it to support headline comparative claims. All anecdotal evidence for alignment with the original papers has been provided in the online Plato documentation.

Primary and secondary metrics. Our primary performance metric is the elapsed wall-clock time to reach a target accuracy (or target perplexity for language modeling), as defined in Section III-E. We also report final validation accuracy/perplexity after a fixed budget when applicable. To address bandwidth concerns raised in the reviews, we additionally report the communication volume in terms of total transmitted bytes. When model updates are exchanged, a first-order estimate is:

$$\text{Bytes}(t) \approx \sum_{i \in \mathcal{S}_t} \Big(\text{Bytes}^{\downarrow}(t) + \text{Bytes}_i^{\uparrow}(t) \Big), \tag{2}$$

where $\text{Bytes}^{\downarrow}(t)$ is the server-to-client payload size (often the model), and $\text{Bytes}_i^{\uparrow}(t)$ is the client-to-server payload size (often the update), including method-specific auxiliary state (e.g., control variates). We treat privacy/security as a complementary dimension: Plato supports secure-aggregation and DP-style workflows via strategies and processors, but this paper does not claim formal privacy guarantees unless such guarantees are explicitly computed and reported.

Clarifying "elapsed wall-clock time" in our plots. Unless otherwise stated, when figures and tables in this section report "elapsed wall-clock time," the x-axis is *Plato's simulated elapsed time* defined in Section III-E with wall-clock simulation enabled. In our single-machine, batched execution environment, this time model is driven primarily by measured client-side compute/processing time and orchestration overhead; we intentionally avoid claiming that it matches real-world end-to-end runtime under a specific network deployment. To make bandwidth effects visible without committing to a particular network model, we report communication volume (bytes) as a complementary metric and explicitly call out method-specific auxiliary state (e.g., control variates) when relevant.

Uncertainty and reproducibility. To mitigate randomness, Plato supports fixed RNG seeds and an optional reproducible mode that freezes client/sample selection. Unless otherwise stated, the figures report single runs under fixed seeds due to the high wall-clock cost of large-scale emulation; we disclose this limitation and provide a reproducibility pathway (versioned configs and scripts) so that additional seeds can be run by the community.

B. Datasets and Models

Among the diverse array of datasets and models that Plato supports (Section III-C), we have chosen the following 5 configurations of datasets and models for our evaluations in this paper:

TABLE III
VALIDATION EVIDENCE SOURCES USED FOR REPRESENTATIVE ALGORITHMS EVALUATED IN THIS PAPER.

Algorithm	Category	Public ref. code	Evidence used in our implementation/validation
FedAvg	Baseline	Y	Paper specification + sanity checks; acts as limiting case for several methods
FedAdp, FedAtt, FedDF	Server aggregation	N/unknown	Paper equations/pseudo-code + invariants; for FedDF, additionally validate the proxy split and distillation payload path
FedProx, SCAFFOLD, FedDyn, MOON	Training-loop customization	Y	Paper equations + public implementations (when available) + limiting-case and payload-content checks
Oort, Active FL, Pisces	Client selection	Y/unknown	Paper description + reference implementations when available; validate report/utility/staleness signals
APFL, Ditto, FedRep, FedBABU, LG-FedAvg, Per-FedAvg, FedPer	Personalization	Y	Paper description + public implementations (when available); validate phase scheduling and payload content

TABLE IV
MINIMAL UNCERTAINTY REPORTING (REPRESENTATIVE WORKLOAD). WE REPORT MEAN±STD OF THE SIMULATED TIME-TO-ACCURACY ON EMNIST+LeNet-5, COMPUTED FROM $n = 2$ FIXED-SEED RUNS PER METHOD (TARGET ACCURACY: 50%).

Algorithm	Time to 50% accuracy (s)
FedAvg	158.3 ± 4.4
FedProx	149.3 ± 14.9
Oort	153.6 ± 10.6
FedAdp	136.0 ± 3.0
FedAtt	143.5 ± 17.1

TABLE V
UNIFIED EVALUATION SETTINGS (REPRESENTATIVE WORKLOADS).

Workload	Clients	Round	Epochs
EMNIST + LeNet-5	1000	100	5
FEMNIST + LeNet-5	1000	100	5
CIFAR-10 + ResNet-18	1000	100	5
CINIC-10 + VGG-16	1000	100	5
Tiny-Shakespeare + DistilGPT2	50	20	1

- ▷ The Extended MNIST (EMNIST) dataset [36] with the LeNet-5 model. This dataset contains $817,851$ images, each of which is a 28×28 grayscale image in 1 out of 62 classes. Representing handwritten digits and letters from over 500 writers, this dataset is a superset of MNIST, and has been a popular benchmark dataset to test the baseline performance in existing FL literature. The classic LeNet-5 model can easily converge to an excellent accuracy with this dataset.
- ▷ The Federated EMNIST [37] dataset with the LeNet-5 model. Originated from EMNIST, the difference of the Federated EMNIST dataset is that it is already partitioned by the client ID, using the data provider IDs included in the original EMNIST dataset. As a result of this partitioning, there are $3,597$ clients in total, each of which has 227.37 images on average. For each client, 90% data samples are used for training, while the remaining samples are used for testing. This dataset is known for its highly non-i.i.d. nature, which can prove helpful when client selection algorithms are evaluated.
- ▷ The CIFAR-10 [38] dataset with the ResNet-18 model. CIFAR-10 has been a staple benchmark in the existing FL literature, especially when combined with ResNet-18.
- ▷ The CINIC-10 dataset with the VGG-16 model. This combination is harder to converge during training, which we hope can provide a different perspective on the performance of FL algorithms.
- ▷ Beyond image classification tasks, we also choose to include a language modeling task for additional variety, training the Distilled-GPT2 model on the Tiny-Shakespeare dataset. Readily available from the Hugging Face repository, Tiny-Shakespeare contains $40,000$ lines of text from a variety of Shakespeare's plays, and Distilled-GPT2 is a distilled variant of the original Transformer-based GPT2 model. As a language modeling task, this task does not use labeled data for training, and therefore does not exhibit data heterogeneity across clients.

We conduct all our experiments on a Dell Precision 7820 server with three NVIDIA RTX A4500 GPUs, each equipped with 24 GB CUDA memory. PyTorch 2.8, CUDA 13.0, and Plato v1.4.3 have been used. In a total of 1000 clients, 100 are chosen in each round, either randomly or using a client selection algorithm. The non-i.i.d. data distributions in our experiments were obtained using the Dirichlet distribution, with a concentration parameter, denoted as α, set to 1 or 0.1, representing a moderately or highly non-i.i.d. distribution, respectively.

Original-paper settings versus unified settings. Since original papers often use different evaluation workloads, budgets, and hyperparameter choices, readers can reasonably question whether a unified-setting comparison alone explains observed performance gaps. To make this boundary explicit, we provide a compact original-settings summary for representative methods in Tables VI–VII and include the full per-method configurations (including additional datasets/models and ablations) in our reproducibility artifact, together with the configuration bundle used in our unified-setting experiments.

C. Server Aggregation Algorithms

Since Federated Averaging (FedAvg), originally proposed with the inception of federated learning [1], is essentially a simple algorithm for the server to aggregate model updates from the clients, we begin our exploration with two

TABLE VI

ORIGINAL-PAPER HEADLINE EXPERIMENTAL SETTINGS VERSUS THE UNIFIED SETTINGS USED IN OUR Plato EVALUATIONS (PART I: AGGREGATION, TRAINING-LOOP CUSTOMIZATION, AND CLIENT SELECTION). FOR COMPACTNESS WE SHOW REPRESENTATIVE METHODS; FULL PER-METHOD CONFIGURATIONS ARE IN OUR REPRODUCIBILITY ARTIFACT. UNLESS STATED, OUR UNIFIED SETTINGS USE $N/n/E$=1000/100/5 (TABLE V).

Algorithm	Cat.	Original paper: headline workload(s)	Orig. $N/n/E$	Original paper: key settings (as reported)	Notes on setting differences / reproduction status
Server aggregation					
FedAdp [7]	Agg.	MNIST/FashionMNIST + CNN	–/10/1	$\|S\|$=10, E=1, T=300; lr 0.01 (decay 0.995/rnd); non-IID x-class.	Unified protocol/time model; full original configs in artifact.
FedAtt [8]	Agg.	WikiText-2/PTB/Reddit + GRU	100/10/ 1–20	$C \in \{0.1, 0.5\}$; T=50; $E \in \{1, 5, 10, 15, 20\}$.	Unified LM uses $N/n/E$=50/20/1 (Table V); full original configs in artifact.
Client training-loop customization					
FedProx [10]	Train.	MNIST + multinomial logistic regression	1000/10/20	10 selected/rnd; SGD; E=20; proximal strength μ tuned.	Unified protocol/time model; full original configs in artifact.
SCAFFOLD [11]	Train.	EMNIST + logistic regression	100/20/5	20% sampled/rnd; E=5; Option II; control variates (c, c_i).	Auxiliary state increases communication (Table XI); full original configs in artifact.
Client selection / scheduling					
Oort [12]	Select.	OpenImage + MobileNet/ShuffleNet	>14k/100/–	Sample 100/rnd; exploration 0.9 (decay); straggler penalty α=2; window W=20.	Unified protocol/time model; full original configs in artifact.
Pisces [13]	Select.	MNIST + LeNet-5 (also CIFAR-10/FEMNIST/StackOverflow)	200/20/5	Concurrency C=20; E=5; staleness β=0.5; Zipf latency (a=1.2).	Our async eval uses $n_{\min} \in \{20, 50\}$ and staleness bound 10 (Section IV-F).

TABLE VII

ORIGINAL-PAPER HEADLINE EXPERIMENTAL SETTINGS VERSUS THE UNIFIED SETTINGS USED IN OUR Plato EVALUATIONS (PART II: PERSONALIZED FEDERATED LEARNING). FOR COMPACTNESS WE SHOW REPRESENTATIVE METHODS; FULL PER-METHOD CONFIGURATIONS ARE IN OUR REPRODUCIBILITY ARTIFACT. UNLESS STATED, OUR UNIFIED SETTINGS USE $N/n/E$=1000/100/5 (TABLE V).

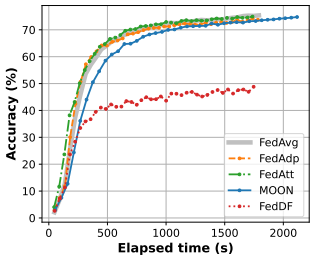
Algorithm	Original paper: headline workload(s)	Orig. $N/n/E$	Original paper: key settings (as reported)	Notes on setting differences / reproduction status
Personalized federated learning				
APFL [39]	MNIST + logistic regression (strongly convex setting)	100/100/–	No client sampling; local steps τ=10; lr 0.1 with linear decay; adaptive mixing α.	Unified protocol/time model; full original configs in artifact.
Ditto [17]	FEMNIST/CelebA (benchmarks; multiple models)	*varies*	Per-device proximal term λ (selected via local validation); personalized model stays local; global trained FedAvg-style.	Unified protocol/time model; full original configs in artifact.
FedRep [14]	CIFAR-10 + CNN (CIFAR-100/FEMNIST/Sent140 also reported)	100/10/ 10+1	r=0.1; alternate head vs representation (e.g., 10+1 epochs); T=100 (CIFAR).	Payload split reduces transmitted parameters; communication implications in Section IV-I.
FedBABU [15]	CIFAR-100 + MobileNet	100/10/10	Client fraction f=0.1; local epochs τ=10; lr schedule with decays; evaluates after fine-tuning stage.	Unified protocol/time model; full original configs in artifact.
LG-FedAvg [16]	MNIST + (local feature extractor + small global head)	100/10/1	C=0.1; E=1; communicates only global head/last layers to reduce communication.	Model split changes payload; communication implications in Section IV-I.
Per-FedAvg [18]	CIFAR-10 + 2-layer MLP (MNIST also reported)	50/10/4	Participation r=0.2; $\tau \in \{4, 10\}$; meta inner/outer (α=0.01, β=0.001); K=1000 rounds.	Meta inner/outer loop adds local computation; we report time-to-target and bytes-to-target under our cost model.

weighted server aggregation algorithms in the literature that were designed to improve FedAvg's performance: FedAdp [7] and FedAtt [8]. Across all four image-classification panels in Fig. 3, we additionally overlay MOON [20] and FedDF [9] to distinguish two adjacent but different directions: MOON modifies the client-side local objective with a contrastive regularizer, while FedDF replaces weighted averaging with server-side distillation/fusion over a shared proxy set.

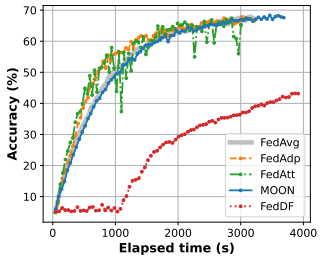
FedAdp [7] sought to accelerate model convergence with the presence of non-i.i.d. data distributions across different clients, by assigning weights to each client based on its "contribution," evaluated based on the moving average of the cosine similarity between its local gradients and the global gradients. The intuition was that clients with local gradients that were more similar to the global gradients should be given more weight, as their updates were more likely to improve the global model.

Though FedAtt [8] proposed a different weighting algorithm for server aggregation, its overarching objective was the same: to produce a global model that is closely aligned with local updates. Instead of using cosine similarity, FedAtt computes the aggregation weights based on layer-wise "attention scores," which is computed layer-by-layer by calculating the norm of the difference between the local and global parameter matrices in each layer.

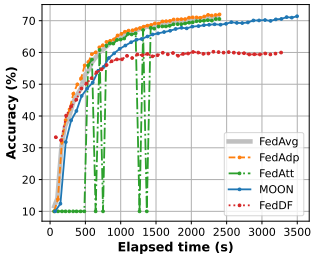
We now begin our evaluations of FedAdp and FedAtt, both state-of-the-art weighted server aggregation algorithms, against FedAvg, our baseline. Before we discuss our observations, it is worth noting that substantial performance gains were reported in the literature; *e.g.*, a reduction of communication rounds of up to 54% using FedAdp; and with the same number of rounds, an validation accuracy improvement of up to 27% using FedAtt. Our results over four combinations of image classification datasets and models, shown in Fig. 3, painted a remarkably different picture. In three of the four



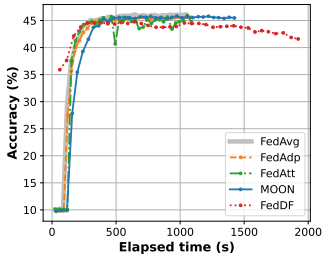
(a) `LeNet-5` on `EMNIST` (Dirichlet distribution with $\alpha = 1$)



(b) `LeNet-5` on `FEMNIST` (with inherent non-i.i.d. data partitioning)



(c) `ResNet-18` on `CIFAR-10` (Dirichlet distribution with $\alpha = 1$)



(d) `VGG-16` on `CINIC-10` (Dirichlet distribution with $\alpha = 1$)

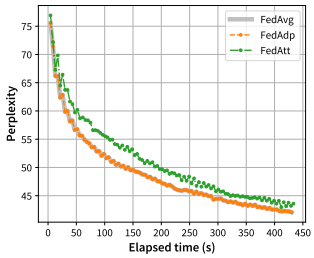
Fig. 3. Evaluating weighted server aggregation algorithms (`FedAvg`, `FedAdp`, and `FedAtt`) and their adjacent algorithm designs (`MOON`, and `FedDF`), using the elapsed wall-clock time as the performance metric, and with the presence of non-i.i.d. data distributions.

tasks, `FedAvg` slightly outperformed both `FedAdp` and `FedAtt` in terms of the elapsed wall-clock time, and with `ResNet-18` on `CIFAR-10`, `FedAvg`'s performance is a close second. If one must select a "winner," it would be `FedAvg`, as `FedAtt` was not as stable during convergence, and `FedAdp` severely underperformed with `VGG-16` on `CINIC-10`.

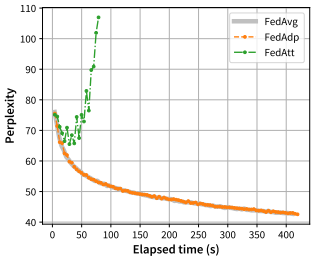
Across all four image-classification panels in Fig. 3, we additionally include `MOON` and `FedDF` as reference overlays because they test a different design hypothesis from weighted averaging. `MOON` keeps the server aggregation rule simple but changes the local training loop with a contrastive objective, whereas `FedDF` fuses client models through server-side distillation on a deterministic proxy split reserved from the corresponding training distribution. We therefore interpret these two families of curves as a scoped comparison point for adjacent design choices under the same unified time model, rather than as evidence that the entire server-aggregation landscape can be reduced to weighted averaging alone.

As no complete source code is made available from these original papers, and because their experimental protocols (workloads, budgets, and reported metrics) differ from ours, we do not attribute the observed gaps to the algorithms themselves. Instead, we report these results as *"under our unified settings and time model"* and treat them as a reproducible comparison point. Plausible sources of discrepancy include differences in ① client participation rates (n/N) and heterogeneity models, ② what is communicated (gradients vs. weight deltas) and whether additional randomization/noise is enabled, and ③ evaluation budgets (rounds vs. time-to-target).

In addition to image classification tasks, we are also interested in how server aggregation algorithms perform on



(a) Synchronous training



(b) Asynchronous training

Fig. 4. Evaluating server aggregation algorithms with the `Distilled-GPT2` model and the `Tiny-Shakespeare` dataset, and with both synchronous and asynchronous modes of training. As a language modeling task, the quality of a model is evaluated using its perplexity (lower values are better).

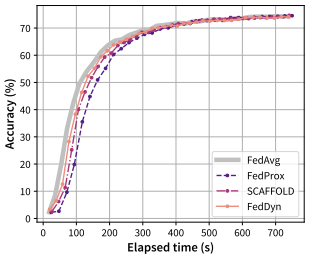
language modeling tasks. Fig. 4(a) and Fig. 4(b) show their performance training `DistilGPT2` on `Tiny-Shakespeare`, with the synchronous and asynchronous modes of training, respectively. Under this workload and our time model, `FedAdp` is comparable to `FedAvg` while `FedAtt` does not reach the target within our asynchronous budget. Since this task does not exhibit label-induced non-i.i.d. heterogeneity, we view these results as a reminder that re-weighting rules tuned for heterogeneous supervised settings may not transfer uniformly across workloads; we therefore avoid making general claims beyond the evaluated settings.

Beyond the results we presented so far, we have also conducted a large number of additional experiments with alternative algorithms of assigning weights to different clients during server aggregation. We observed no statistically significant advantage over `FedAvg` by varying the assigned weights. Conceptually, it is reasonable to assign more weights to clients with data of higher quality; but in practice, it is difficult to accurately assess the quality of data on each client, especially if we consider all potential datasets and models.

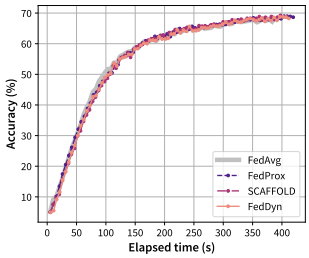
D. Client Training Loop Customization Algorithms

Rather than assigning different weights to clients during server aggregation, it was also proposed in the literature that the training loops on the clients can be customized instead. In a similar spirit as server aggregation, when a client's local training deviates from the global model substantially, it may suffer from low data quality, and should be adjusted with a regularizer. We evaluate three such algorithms in this category — `FedProx` [10], `SCAFFOLD` [11], and `FedDyn` [40] — against `FedAvg`, our baseline.

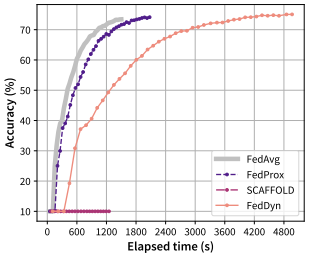
Reported gains for training-loop customization methods are typically under paper-specific workloads and budgets and are often summarized in rounds. Under our unified settings and simulated time-to-target metric (Section III-E), we do not observe consistent improvements over `FedAvg` across the four workloads in Fig. 5. In particular, `SCAFFOLD` does not reach the target within the budget when training `ResNet-18` on `CIFAR-10`, and both `FedProx` and `FedDyn` can lag behind `FedAvg` on larger models, even when they are competitive on simpler workloads.



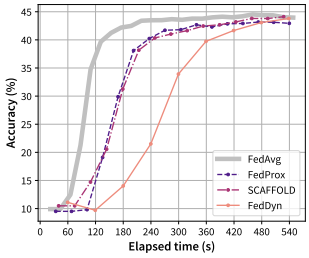
(a) `LeNet-5` on `EMNIST` (Dirichlet distribution with $\alpha = 1$)



(b) `LeNet-5` on `FEMNIST` (with inherent non-i.i.d. data partitioning)

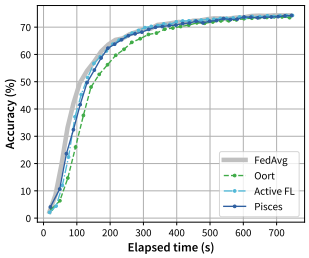


(c) `ResNet-18` on `CIFAR-10` (Dirichlet distribution with $\alpha = 1$)

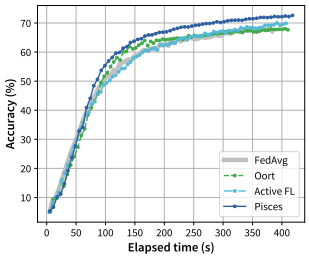


(d) `VGG-16` on `CINIC-10` (Dirichlet distribution with $\alpha = 1$)

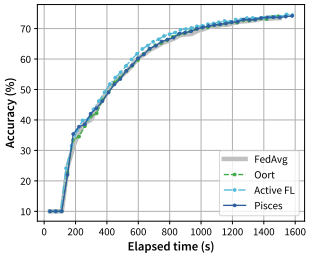
Fig. 5. Evaluating three client training loop customization algorithms — `FedProx`, `SCAFFOLD`, and `FedDyn` — using the elapsed wall-clock time as the performance metric, and with the presence of non-i.i.d. data distributions.



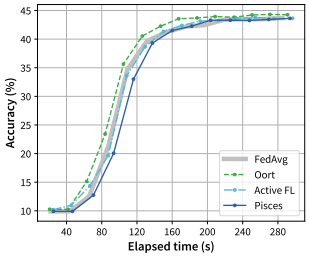
(a) `LeNet-5` on `EMNIST` (Dirichlet distribution with $\alpha = 1$)



(b) `LeNet-5` on `FEMNIST` (with inherent non-i.i.d. data partitioning)



(c) `ResNet-18` on `CIFAR-10` (Dirichlet distribution with $\alpha = 1$)



(d) `VGG-16` on `CINIC-10` (Dirichlet distribution with $\alpha = 1$)

Fig. 6. Evaluating three client selection algorithms — `Oort`, `Active FL`, and `Pisces` — using the elapsed wall-clock time as the performance metric, and with the presence of non-i.i.d. data distributions.

Because several papers do not provide end-to-end reference code and the exact hyperparameter schedules, we do not claim to reproduce their convergence trajectories. Our experiments also highlight a practical trade-off that can be obscured when comparisons are reported only in rounds: training-loop customization can increase per-round compute and/or communication (e.g., `SCAFFOLD` communicates model-sized control variates), so improvements in rounds do not necessarily translate to improvements in elapsed time or bytes-to-target. We therefore report communication-volume estimates as a complementary metric in Section IV-I (Table XI).

Last but not the least, it is worth noting that customizing client training loops is not feasible in most scenarios where the training loop is supplied by the underlying framework, rather than manually implemented (as is typical in simple PyTorch tutorials). For example, training loops for models in the Hugging Face repository are supplied by the Hugging Face training stack, and manual customization requires both access to the underlying code path and in-depth understanding of its execution. In this paper, we therefore do not evaluate *training-loop customization algorithms* on the HuggingFace-based `Distilled-GPT2` workload; however, we do evaluate *server aggregation* on `Distilled-GPT2+Tiny-Shakespeare` in Fig. 4.

E. Client Selection Algorithms

Client selection algorithms addressed the challenge of non-i.i.d. client data distributions using a substantially different approach: they proposed to select clients with high-quality data to begin with in each round. In this category, we choose to evaluate the following three state-of-the-art algorithms, using different measures to select clients from the server's perspective:

- ▷ `Oort` [12] used a measure calculated during the local training process, called *utility*, to estimate the quality of data on each client. The utility of a client is simply computed based on its training loss.
- ▷ `Active FL` [22] is a federated active learning method that uses a measure called *value function* to evaluate the quality of data on each client, which, similar to `Oort`, is also computed from training-loss signals. The reviewer-suggested CHASe method [23] is technically relevant here because it makes client heterogeneity explicit in the data-selection criterion rather than treating client choice as a purely utility-driven ranking.
- ▷ Different from `Oort` and `Active FL`, `Pisces` [13] was proposed to improve performance in *asynchronous* FL scenarios. In `Pisces`, clients were selected the same statistical utility computed with the training loss as in `Oort`, combined with the degree of staleness on slower clients due to asynchronous aggregation. Clients with lower utilities and higher degrees of staleness were less likely to be selected.

These client-selection methods are adjacent to personalized FL, but they solve a different problem: selection changes who contributes to training, whereas personalization changes what each client ultimately keeps. We cite CHASe [23] here as literature context for heterogeneity-aware selection, but we do not treat it as an additional personalized-FL baseline in our evaluation suite.

Our experimental results have been shown in Fig. 6, using the same combinations of datasets and models. In our results,

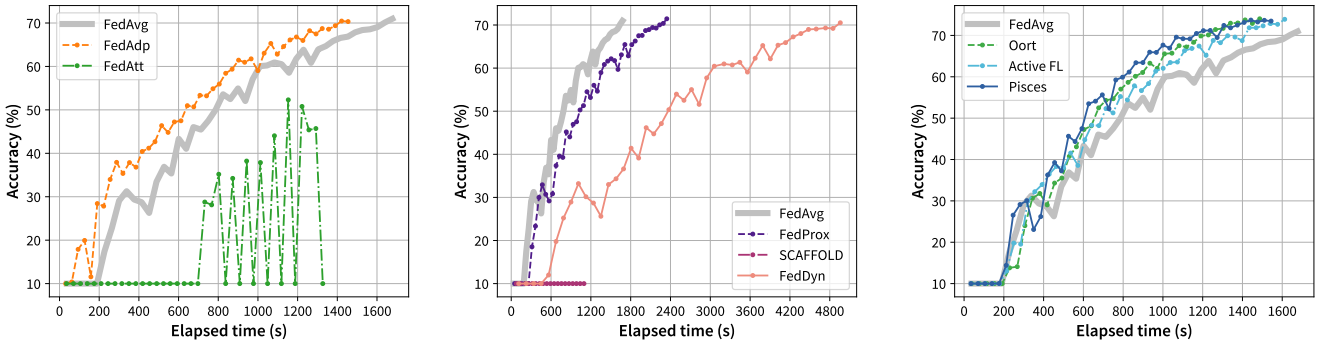


Fig. 7. Evaluating (1) server aggregation; (2) client training loop customization; and (3) client selection algorithms with an extreme non-i.i.d. Dirichlet distribution, where $\alpha = 0.1$. The `ResNet-18` model is trained on the `CIFAR-10` dataset, and 100 out of 1000 clients are selected in each round.

there is no clear "winner" across the board: `Oort` performed best on the `CINIC-10` dataset, `Active FL` performed the best on `CIFAR-10`, `Pisces` performed the best on `FEMNIST`, and `FedAvg` performed best on `EMNIST`. In all our results except `FEMNIST`, the differences are marginal. With `FEMNIST`, the performance gap is approximately 4% (72% vs. 68%) in the model accuracy after convergence, comparing `Pisces` with `FedAvg` at around 380s. Given that the `FEMNIST` dataset is handcrafted to represent an extreme non-i.i.d. distribution, such a performance gain in accuracy — and no performance gains in the speed of convergence — is quite marginal.

In this extreme non-i.i.d. setting, we observe that biased client selection algorithms can concentrate participation on a subset of clients that appear "high utility" under their scoring rules. This can reduce time-to-target in some cases, but it also raises a familiar exploitation–exploration tension: aggressive exploitation may reduce coverage of the population distribution and can increase overfitting risk if the selected subset is not representative. We therefore treat these algorithms as workload- and heterogeneity-dependent, and we avoid presenting them as uniformly superior to random selection.

F. Extreme Data Heterogeneity

The observation that `Pisces` outperformed `FedAvg` with an extreme non-i.i.d. data distribution, represented by the `FEMNIST` dataset, was quite intriguing, and motivated us to perform more experiments in more extreme settings of data heterogeneity across all three categories: server aggregation, customizing client training loops, and client selection. As existing algorithms in all these categories were proposed to address the challenge of data heterogeneity, we wonder if more extreme data heterogeneity conditions may widen the performance advantages over `FedAvg`.

We show our results in Fig. 7, where client data follows an extreme non-i.i.d. Dirichlet distribution with $\alpha = 0.1$. Under this setting and our time model, training-loop customization methods are not consistently competitive: `SCAFFOLD` does not converge within the budget, and both `FedProx` and `FedDyn` can be slower than `FedAvg`. For server aggregation, `FedAdp` converges faster than `FedAvg` in this particular setting, while `FedAtt` does not converge within the budget, suggesting that the choice of re-weighting rule can substantially affect stability. For client selection, biased policies can improve time-to-target and/or final accuracy in this extreme heterogeneity regime, but we emphasize that these gains are settings-dependent and may not generalize beyond the tested scenario.

At this point, it is worth noting that such extreme non-i.i.d. data distributions may be a rare occurrence in practice. Consider a handwriting recognition task that the `EMNIST` dataset embodies: it would have been inconceivable that each user would always write just several of the digits and characters, and none of the remaining ones.

To summarize, Table VIII presents the time required for each algorithm to reach the target accuracy listed in the last row (bold indicates the best time-to-target for each workload). Overall, there is no single method that dominates across workloads under our unified settings and time model. `FedAvg` remains a strong baseline and is often close to the best performer; client selection can help in some heterogeneous regimes but does not uniformly improve results; and for training-loop customization methods, the additional computation/communication overhead can offset their intended benefits in time-to-target comparisons.

G. Asynchronous Federated Learning

In addition to synchronous FL settings, we are also intrigued about how these client selection algorithms perform in asynchronous FL settings. As we presented in Section III-E, a substantial amount of effort has been made to allow Plato to support asynchronous FL settings at a larger scale, with simulated wall-clock times and limited GPU hardware. Such an investment of development effort is largely due to the fact that asynchronous FL is not only more performant than it synchronous counterpart, but more practical as well. Indeed, a more recent client selection algorithm, Pisces [13], is specifically designed for asynchronous training.

With a total of 1000 clients and 100 of them selected the first round, our experimental results are shown in Fig. 8. $n_{\min}$, the minimum number of client updates that the server should wait for before aggregating, is set to 20 and 50 in our `EMNIST` and `CIFAR-10` tasks, respectively. The staleness bound is 10, implying that if a client is more than 10 rounds behind the current global model, the server would need to wait for the client to finish. As one can observe in our results, the

TABLE VIII
THE ELAPSED TIME TO CONVERGE TO A TARGET ACCURACY: A COMPARISON ACROSS SERVER AGGREGATION, CLIENT TRAINING LOOP CUSTOMIZATION, AND CLIENT SELECTION ALGORITHMS.

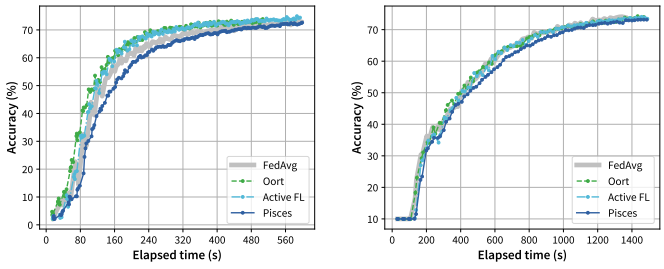
Categories	Algorithms	EMNIST + LeNet-5	FEMNIST+ LeNet-5	CIFAR-10 + ResNet-18	CINIC-10 + VGG-16	CIFAR-10 + ResNet-18 Dirichlet $\alpha = 0.1$
Server Aggregation	FedAvg	632s	368s	1391s	219s	1640s
	FedAdp	953s	388s	**1089s**	286s	1421s
	FedAtt	**616s**	362s	failed to reach	449s	failed to converge
Client Training Loop Customization	FedProx	659s	351s	1710s	473s	2237s
	Scaffold	689s	341s	failed to converge	424s	failed to converge
	FedDyn	653s	345s	3786s	479s	4959s
Client Selection	Oort	801s	405s	1380s	**167s**	1228s
	Active FL	672s	330s	1292s	192s	1434s
	Pisces	725s	**222s**	1382s	204s	**1169s**
Model Target Accuracy		74%	68%	73%	43%	70%

TABLE IX
THE ACCURACY OF THE MODEL CONVERGING AFTER THE EXACT NUMBER OF ROUNDS WE RAN: A COMPARISON ACROSS SERVER AGGREGATION, CLIENT TRAINING LOOP CUSTOMIZATION, AND CLIENT SELECTION ALGORITHMS.

Categories	Algorithms	EMNIST + LeNet-5	FEMNIST+ LeNet-5	CIFAR-10 + ResNet-18	CINIC-10 + VGG-16	CIFAR-10 + ResNet-18 Dirichlet $\alpha = 0.1$
Server Aggregation	FedAvg	74.73%	68.2%	73.42%	43.4%	67.96%
	FedAdp	73.79%	68.33%	**74.24%**	42.62%	67.49%
	FedAtt	75.24%	69.08%	failed to reach	42.85%	failed to converge
Client Training Loop Customization	FedProx	**75.5%**	68.18%	73.72%	42.68%	68.93%
	Scaffold	74.66%	68.86%	failed to converge	42.46%	failed to converge
	FedDyn	74.73%	68.58%	72.26%	43.8%	69.07%
Client Selection	Oort	74.72%	67.76%	74.04%	**43.94%**	**74.02%**
	Active FL	74.59%	69.23%	73.59%	43.22%	71.85%
	Pisces	74.63%	**73.13%**	73.65%	43.3%	72.49%
Model Target Running Rounds		40	75	40	10	40

performance of these client selection algorithms (including random selection in FedAvg) is broadly comparable under our asynchronous configuration. Oort and Active FL slightly outperform FedAvg in our EMNIST task, whereas Pisces is slightly weaker than FedAvg in both tasks.

To further drive home the performance difference between these client selection algorithms, we show in Table X the elapsed wall-clock time, as simulated by Plato, required for them to converge to a specific target accuracy. The results indicate that FedAvg is competitive in time-to-target under our asynchronous configuration, and it is the fastest in some settings (e.g., CIFAR-10). These results suggest that custom-tailored client selection algorithms that rely on utility estimates may not improve performance uniformly across tasks, reinforcing the need to report results under clearly specified settings and metrics.



(a) Training the LeNet-5 model on the EMNIST dataset, with $n_{\min} = 20$.

(b) Training the ResNet-18 model on the CIFAR-10 dataset, with $n_{\min} = 50$.

Fig. 8. Comparing client selection algorithms in asynchronous FL training with an extreme non-i.i.d. Dirichlet distribution, where $\alpha = 0.1$.

TABLE X
COMPARING THE ELAPSED TIME TO CONVERGE TO A TARGET ACCURACY.

Tasks	Model Target Accuracy	Algorithms			
		FedAvg	Oort	Active FL	Pisces
EMNIST + LeNet-5	73%	619s	**464s**	563s	696s
CIFAR-10 + ResNet-18	74%	**1327s**	1433s	1456s	1626s

H. Personalized Federated Learning

Personalized federated learning has recently emerged as one of the most actively investigated topics in the general area of federated learning. The overall objective of proposed personalized FL algorithms in the literature was to train models with excellent validation accuracies when tested with local data on each client. The fundamental assumption was that,

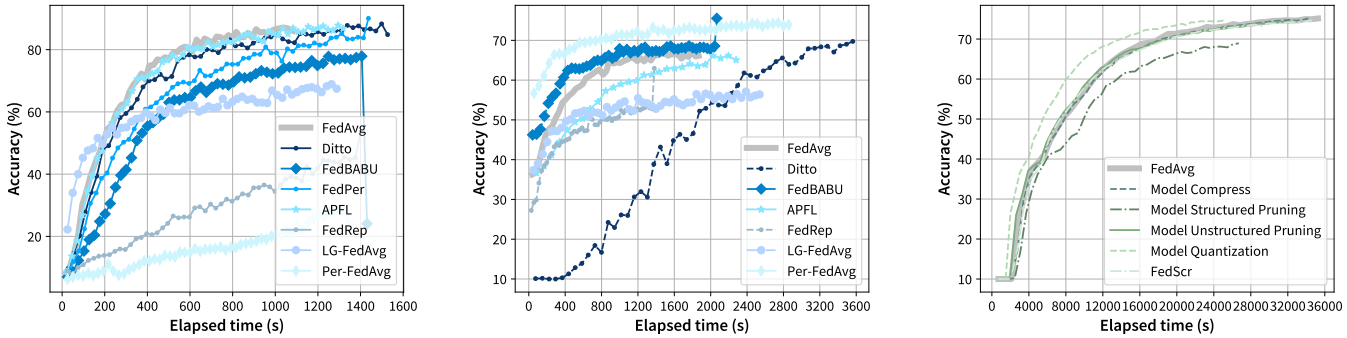


Fig. 9. Validations of personalized federated learning algorithms with training the (1) `LeNet-5` model on the `FEMNIST` dataset; (2) training the `ResNet-18` model on the `CIFAR-10` dataset. And (3) an example of evaluating payload processors (e.g., compression/quantization/pruning) as modular components that alter communication cost.

by collaborating with other clients using federated learning, a shared global model could be obtained to further improve local validation accuracies, as compared to using local data only for local training. This assumption holds due to data heterogeneity across different clients.

Before we start to examine personalized FL algorithms in the literature, it is worth pointing out that an intuitive baseline is to first use `FedAvg` as the server aggregation algorithm to train a global model, and after a target accuracy or round is reached, then apply local fine-tuning to the converged global model using local data on each client. Such an intuitive baseline, simply called `FedAvg` in our experiments, has also been mentioned in the literature. For example, Adaptive Personalized Federated Learning (APFL) [39] included this baseline algorithm in its comparison study, calling it *Localized FedAvg*.

As we began to examine the literature, however, we made a few rather surprising discoveries, with both good and bad news to share about evaluating personalized FL algorithms. On the one hand, the good news is that most related works in the literature share the same performance objective of maximizing the average local validation accuracy across all clients. Though some existing works propose to guarantee some degree of fairness, maximizing the average local validation accuracy remains to be a key performance metric.

On the other hand, the bad news is that existing designs are substantially different, making it challenging to compare them fairly across-the-board. For example, some related works, such as Ditto [17], APFL [39], LG-FedAvg [16] and Per-FedAvg [18], perform their personalization steps during the FL training process, while others, such as FedBABU [15] and Localized FedAvg, start their personalization steps only after the training process concludes and the shared global model converges.

Even for personalized FL algorithms that fall into the same category and perform their personalization during the training process, it is technically challenging to compare them fairly across-the-board. If we carry out an in-depth investigation, some algorithms, such as Ditto [17] and LG-FedAvg [16], perform twice the number of epochs in each round of local training. It is obviously not fair to compare the local validation accuracies of algorithm `A` with its alternative algorithm `B` that uses twice the number of epochs every round, without doubling the number of epochs for `A`.

Despite all the technical difficulties against fair comparisons, we have meticulously implemented *eight* different personalized FL algorithms in Plato: `APFL` [39], `Ditto` [17], `FedPer` [41], LG-FedAvg [16], Per-FedAvg [18], FedBABU [15], FedRep [14], and last but not the least, Localized `FedAvg` with a personalization phase after the model converges.

Taken together, these methods are not interchangeable implementations of a single idea: some personalize by changing the local objective during training, while others defer personalization until after the shared model converges. That distinction matters in our benchmark because the personalization phase changes both the computation budget and the payload exchanged during training, so we keep the methods separate rather than collapsing them into one aggregate "personalization" score.

When we train the `LeNet-5` model with the `FEMNIST` dataset, shown in Section IV-H, it turns out that Localized `FedAvg`, shown as `FedAvg` in the figure, demonstrated the best performance among all 8 personalized FL algorithms, when performance is evaluated using the average local validation accuracies across all clients. Though some alternative algorithms such as `APFL` and `Ditto` showed similar performance as `FedAvg`, others performed poorly in this simple benchmark. `LG-FedAvg` shows rapid convergence initially, but its average local accuracy after convergence is mediocre at best. `FedRep` and `Per-FedAvg` showed the worst performance in the pack, showing abysmally slow convergence over time.

If we switch to a more challenging training workload that trains a `ResNet-18` model on the `CIFAR-10` dataset, shown in Section IV-H, `Per-FedAvg` and `FedBABU`, surprisingly, took the top two spots in the performance ranking, matching the convergence speed of `FedAvg`. `Ditto`, which showed competent performance with the `LeNet-5` model, performed poorly this time, due to its high communication overhead that took twice as long in wall-clock time to converge.

By comparing the performance of the same algorithm across both benchmarks, we make a surprising observation that some algorithms, such as `Per-FedAvg` and `Ditto`, showed remarkably inconsistent performance. They often show leading

TABLE XI
APPROXIMATE COMMUNICATION VOLUME PER ROUND UNDER OUR UNIFIED SETTINGS (TABLE V), ASSUMING FULL-PRECISION MODEL EXCHANGE (DOWNLOAD + UPLOAD) WITH NO COMPRESSION AND IGNORING METHOD-SPECIFIC AUXILIARY STATE (E.G., CONTROL VARIATES).

Workload	Model size (MB)	Volume/round
EMNIST + LeNet-5	0.24	48 MB
CIFAR-10 + ResNet-18	42.70	8.54 GB
CINIC-10 + VGG-16	56.25	11.25 GB

performance in one benchmark, while offering very poor performance in another. The most consistent option, as we can easily observe, is Localized FedAvg, which outperformed all other algorithms in the LeNet-5 benchmark, and among the top 3 in the ResNet-18 benchmark. If one were asked to choose a personalized FL algorithm in practice, localized FedAvg is, without a doubt, more likely to be chosen.

I. Communication Efficiency and Payload Processing

Communication-efficient FL algorithms target the bandwidth and latency bottlenecks of federated training by reducing transmitted bytes or by changing what is communicated (e.g., compressed model updates, quantized weights, sparse/structured updates, and method-specific auxiliary state). In Plato, such functionality is exposed through payload and communication strategies and processor pipelines, which can transform outbound payloads before transmission and reconstruct them on receipt (e.g., compression/decompression, encryption/decryption, and safetensor serialization). This design allows communication-efficient algorithms to be benchmarked alongside algorithmic changes (aggregation, client selection, and training-loop customization) under a consistent metric suite, including wall-clock time and communication volume.

V. CONCLUDING REMARKS

We designed and built Plato, an open-source benchmarking framework for federated learning research, with a primary focus on reproducible benchmark comparisons across a large number of federated learning algorithms at scale, yet with limited GPU hardware. In this paper, we presented the design and implementation highlights in Plato; but more importantly, we reported findings from evaluating representative algorithm families under unified settings and a consistent wall-clock time model.

Overall, across the classic benchmark workloads we evaluated under our unified configurations and simulated wall-clock time model, FedAvg with random client selection remains a remarkably strong and stable baseline, often matching—and sometimes exceeding—the time-to-target performance of more complex alternatives once their added computation, communication, and tuning overheads are accounted for. Importantly, we do not interpret these results as a universal statement about the intrinsic merit of any algorithm; rather, they highlight that reported gains can be highly workload- and setting-dependent, and that no single method consistently dominates across tasks, models, and heterogeneity regimes. We observed the clearest improvements over FedAvg primarily in client selection under extreme data heterogeneity, suggesting that modest scheduling bias can be beneficial in some regimes, but also that such gains should be reported alongside their potential exploration-exploitation tradeoffs and communication cost. We hope Plato and the accompanying reproducibility artifacts encourage future studies to make these tradeoffs explicit — by validating implementations, pinning configurations, and reporting time-to-target (and, when relevant, communication volume) — so that performance claims become easier to substantiate, compare, and transfer to new deployments.

ACKNOWLEDGMENT

This work was supported in part by the NSERC Discovery Research Program, and by an NSFC grant 62432008, a RGC RIF grant R6021-20, a RGC TRS grant T43-513/23N-2, RGC CRF grants C7004-22G, C1029-22G and C6015-23G, an NSFC/RGC grant CRS_HKUST601/24, as well as RGC GRF grants 16207922, 16207423 and 16203824.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

[2] Y. Wen, J. Geiping, L. Fowl, M. Goldblum, and T. Goldstein, "Fishing for User Data in Large-Batch Federated Learning via Gradient Magnification," in *Proc. International Conference on Machine Learning (ICML)*, 2022.

[3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep Learning with Differential Privacy," in *Proc. SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[4] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proc. Symposium on Theory of Computing (STOC)*, 2009.

[5] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine*, 2020.

[6] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated Learning with Non-IID Data," *arXiv preprint arXiv:1806.00582*, vol. 15, p. 100207, 2018.

[7] H. Wu and P. Wang, "Fast-Convergent Federated Learning with Adaptive Weighting," *IEEE Trans. Cognitive Communications and Networking*, vol. 7, no. 4, pp. 1078–1088, 2021.

[8] S. Ji, S. Pan, G. Long, X. Li, J. Jiang, and Z. Huang, "Learning Private Neural Language Modeling with Attentive Aggregation," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[9] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble Distillation for Robust Model Fusion in Federated Learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[10] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *Proc. 3rd Conference on Machine Learning and Systems (MLSys)*, 2020.

[11] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. International Conference on Machine Learning (ICML)*, vol. 119, 2020, pp. 5132–5143.

[12] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient Federated Learning via Guided Participant Selection," in *Proc. Symposium on Operating Systems Design and Implementation (OSDI)*, 2021, pp. 19–35.

[13] Z. Jiang, W. Wang, B. Li, and B. Li, "Pisces: Efficient Federated Learning via Guided Asynchronous Training," in *Proc. ACM Symposium on Cloud Computing (SoCC)*, 2022.

[14] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting Shared Representations for Personalized Federated Learning," in *Proc. International Conference on Machine Learning (ICML)*, 2021.

[15] J. Oh, S. Kim, and S.-Y. Yun, "FedBABU: Toward Enhanced Representation for Federated Image Classification," in *Proc. The Tenth International Conference on Learning Representations (ICLR)*, 2022.

[16] P. P. Liang, T. Liu, L. Ziyin, R. Salakhutdinov, and L.-P. Morency, "Think Locally, Act Globally: Federated Learning with Local and Global Representations," *arXiv preprint arXiv:2001.01523*, 2020.

[17] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and Robust Federated Learning through Personalization," in *Proc. International Conference on Machine Learning (ICML)*. PMLR, 2021, pp. 6357–6368.

[18] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[19] R. Ye, Z. Ni, F. Wu, K. Chen, and S. Chen, "pFedGraph: Personalized Federated Learning with Inferred Collaboration Graphs," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 39 801–39 817.

[20] Q. Li, B. He, and D. Song, "MOON: Model-Contrastive Federated Learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 10 713–10 722.

[21] J. Zhang, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and H. Guan, "FedALA: Adaptive Local Aggregation for Personalized Federated Learning," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[22] J. Goetz, K. Malik, D. Bui, S. Moon, H. Liu, and A. Kumar, "Active Federated Learning," *arXiv preprint arXiv:1909.12641*, 2019.

[23] J. Zhang, J. Wang, H. Li, Z. Xie, K. Chen, and L. Shou, "CHASe: Client Heterogeneity-Aware Data Selection for Effective Federated Active Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 6, pp. 3088–3102, 2025. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TKDE.2025.3547423

[24] H. Zhang, F. Luo, J. Wu, and F. Chen, "LightFR: Lightweight Federated Recommendation with Privacy-Preserving Matrix Factorization," *ACM Transactions on Information Systems*, vol. 41, no. 4, 2023.

[25] J. Zhang, H. Li, D. Rong, Y. Zhao, K. Chen, and L. Shou, "Preventing the Popular Item Embedding Based Attack in Federated Recommendations," in *Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 2179–2191.

[26] S. Gupta, Y. Huang, Z. Zhong, T. Gao, X. Liu, X. Lin, and M. Bansal, "FILM: Recovering Private Text in Federated Learning of Language Models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 8130–8143.

[27] F. Lai, Y. Dai, S. S. Singapuram, J. Liu, X. Liu, H. Zhu, H. V. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking Model and System Performance of Federated Learning at Scale," in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 162. PMLR, 2022, pp. 11 814–11 827.

[28] C. He and *et al.*, "FedML: A Research Library and Benchmark for Federated Machine Learning," in *Proc. NeurIPS Federated Learning Workshop*, 2020.

[29] D. Chai, L. Wang, L. Yang, J. Zhang, K. Chen, and Q. Yang, "A Survey for Federated Learning Evaluations: Goals and Measures," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 10, pp. 5007–5024, 2024.

[30] Y. Liu, J. Jiang, S. Wu, Y. Li, S. Zhou, W. Li, and T. Liu, "PFLlib: A Beginner-Friendly and Comprehensive Personalized Federated Learning Library and Benchmark," *Journal of Machine Learning Research*, vol. 26, no. 50, pp. 1–10, 2025.

[31] KarhouTam and contributors, "FL-bench: A Federated Learning Benchmark," https://github.com/KarhouTam/FL-bench, 2023, accessed: 2026-01-09.

[32] D. Chen, J. W. Wong, Q. Wang, S. Xu, Z. Feng, J. Chen, Q. Fan, P. Varshney, and Y. You, "pFL-Bench: A Comprehensive Benchmark for Personalized Federated Learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 9344–9360.

[33] J. Zhang, X. Wu, Y. Zhou, Y. Li, S. Zheng, Z. Chai, and F. Du, "HtFLlib: A Comprehensive Heterogeneous Federated Learning Library and Benchmark," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2025.

[34] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices with Non-IID Data," in *Proc. IEEE International Conference on Big Data*, 2020, pp. 15–24.

[35] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated Learning with Buffered Asynchronous Aggregation," in *Proc. International Conference on Machine Learning (ICML)*, 2021.

[36] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "EMNIST: Extending MNIST to Handwritten Letters," in *Proc. IEEE International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921–2926.

[37] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A Benchmark for Federated Settings," *arXiv preprint arXiv:1812.01097*, 2018.

[38] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[39] Y. Deng, M. M. Kamani, and M. Mahdavi, "Adaptive Personalized Federated Learning," *arXiv preprint arXiv:2003.13461*, 2020.

[40] D. A. E. Acar, Y. Zhao, R. M. Navarro, M. Mattina, P. N. Whatmough, and V. Saligrama, "Federated Learning Based on Dynamic Regularization," in *Proc. The Tenth International Conference on Learning Representations (ICLR)*, 2021.

[41] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated Learning with Personalization Layers," *CoRR*, 2019.

Ningxin Su received her M.E. and B.E. degrees from the University of Sheffield and Beijing University of Posts and Telecommunications in 2020 and 2019, respectively, and her Ph.D. degree from the University of Toronto in 2025. She is currently a tenure-track Assistant Professor at the Internet of Things Thrust, Information Hub, Hong Kong University of Science and Technology, Guangzhou. She was a recipient of the Best Paper Award from the IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom) in 2023. Her research areas include the distributed machine learning, federated learning, metaverse and networking.

Baochun Li (Fellow, IEEE) received his B.Eng. degree from Tsinghua University in 1995 and his M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign in 1997 and 2000, respectively. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. Since August 2005, he holds the Bell Canada Endowed Chair in Computer Engineering. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000, the Multimedia Communications Best Paper Award from the IEEE Communications Society in 2009, the University of Toronto McLean Award in 2009, the Best Paper Award from IEEE INFOCOM in 2023, and the IEEE INFOCOM Achievement Award in 2024. He is a Fellow of the Canadian Academy of Engineering, the Engineering Institute of Canada, and IEEE. His current research interests include cloud computing, security and privacy, distributed machine learning, federated learning, and networking.

Bo Li (Fellow, IEEE) received the BEng (summa cum laude) degree in computer science from Tsinghua University, Beijing, China, and the PhD degree in electrical and computer engineering from the University of Massachusetts at Amherst. He is a chair professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He held a Cheung Kong visiting chair professor with Shanghai Jiao Tong University between 2010 and 2016. He was an adjunct researcher with the Microsoft Research Asia (MSRA) (1999-2006) and with the Microsoft Advanced Technology Center (2007–2009). He made pioneering contributions in multimedia communications and the Internet video broadcast, in particular the Coolstreaming system, which was credited as the first large-scale Peer-to-Peer live video streaming system in the world. It attracted significant attention from both industry with substantial VC investment, and academia in receiving the Test-of-Time Best Paper Award from IEEE INFOCOM (2015). He received 6 Best Paper Awards from IEEE including INFOCOM (2021). He has been an editor or a guest editor for more than a two dozen of IEEE and ACM journals and magazines. He was the Co-TPC chair for IEEE INFOCOM 2004.