Flower Hub: A Reproducible Benchmarking Platform for Federated Learning in Simulation and Deployment

Yan Gao[1,2,*], Mohammad Naseri[1,*], Javier Fernandez-Marques[1,2], Dimitris Stripelis[1], Lorenzo Sani[1,2], Davide Eynard[3], Fan Zhang[2], Hong Jia[4], Ting Dang[5], D. B. Emerson[6], Fatemeh Tavakoli[6], Ole Werger[7], Lars Wulfert[7], Petros Demetrakopoulos[8], Sofia Tsekeridou[8], InSeo Song[9], KangYoon Lee[9], Honghao Li[10], Lingjuan Lyu[11], John P Dickerson[3], Daniel Janes Beutel[1], Nicholas D. Lane[1,2]

[1]Flower Labs, [2]University of Cambridge, [3]Mozilla.ai, [4]University of Auckland, [5]University of Melbourne, [6]Vector Institute, [7]Fraunhofer IMS, [8]NetCompany, [9]Gachon University, [10]Owkin, [11]Sony AI

Abstract

Federated learning (FL) has emerged as a key approach for training models across decentralized data, yet benchmarking in FL remains difficult to reproduce, compare, and extend. Existing evaluations are often tied to custom infrastructure, released as incomplete research code, and conducted primarily in simulation, which limits its portability and practical relevance. We present Flower Hub, a platform for publishing, discovering, and executing decentralized and federated applications. We show how it enables reproducible benchmarking by packaging benchmarks as executable, versioned applications with standardized metadata, pinned dependencies, and explicit evaluation workflows. We instantiate this approach with a multi-domain benchmark suite spanning cross-silo and cross-device settings, and including tasks in medical imaging, financial tabular learning, legal instruction tuning, phishing URL detection, and audio tagging. We further demonstrate that the same benchmarking application can run across both simulation and deployment runtimes without changing the application code, enabling unified evaluation across varying learning environments. Beyond model quality, our benchmark design supports system-aware reporting, including runtime and communication metrics. This work advances benchmarking in FL settings from ad hoc code artifacts towards portable, executable, and reusable benchmark applications.

1 Introduction

Many modern machine learning applications rely on distributed data that cannot be centralized due to privacy, regulatory, or bandwidth constraints [10, 39, 45, 49]. This occurs in domains such as healthcare, edge devices, finance, and legal services [16, 39, 45, 50]. Federated learning (FL) [25, 28, 29, 49] has emerged as a key paradigm for training models across decentralized datasets without requiring raw data to be moved to a central location. In recent years, FL research has grown rapidly, with new algorithms, optimization methods, privacy mechanisms, and system architectures proposed at an increasing pace [17, 49]. As the number of FL approaches continues to expand, fair comparison and validation under unified benchmark settings have become increasingly important.

Despite this progress, benchmarking FL systems is inherently challenging. Unlike centralized ML experiments, FL studies typically require infrastructure for collaborative training across multiple parties. As a result, researchers must often manage not only the algorithms being evaluated but also the

*Equal contribution

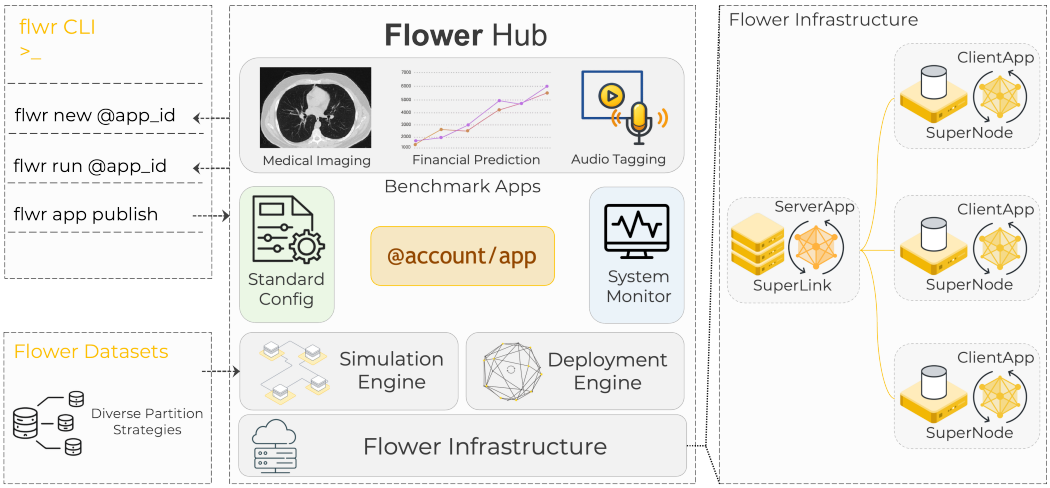


Figure 1: **Overview of Flower Hub.** Flower Hub is an open-source benchmarking platform for federated and decentralized learning that decouples application logic from infrastructure, enabling lightweight, portable and shareable benchmark applications. This design allows researchers to focus on algorithmic development and evaluation. Apps can be executed smoothly across both simulation and real-world deployment settings without code modifications, thereby accelerating the transition from development to deployment. Each published benchmarking app follows a standardized package structure, enhancing reproducibility and facilitating sharing, reuse, and extensibility. An integrated system monitor provides real-time tracking of key performance metrics. Together, these features position Flower Hub as a foundation for a collaborative federated benchmarking ecosystem.

underlying distributed systems logic. Many existing FL benchmarks are isolated projects that tightly couple infrastructure with application code [4, 20, 26, 32, 46]. Consequently, sharing a benchmark with another team often requires sharing the entire infrastructure stack rather than only the model, dataset, and algorithmic components. This coupling makes benchmarks difficult to reuse, extend, and reproduce. Reproducing FL results across teams, therefore, demands substantial engineering effort and deep system expertise, which undermines reproducibility, a central requirement of any benchmark.

Another limitation is that FL evaluation is still dominated by simulation [5, 18]. While simulation is useful for preliminary algorithmic assessment, transitioning from simulation to real-world federated deployment often requires substantial code changes and major engineering overhead. More importantly, FL lacks a standardized benchmark packaging format. Existing benchmarks are commonly released as research repositories, custom scripts, and partially documented pipelines, with no unified way to package training code, evaluation protocols, distributed configurations, and runtime environments. As a result, each new benchmark often becomes another isolated project, thereby limiting the scalability and long-term usability of FL benchmarking efforts. Finally, many FL papers continue to evaluate their methods on centralized datasets such as CIFAR-10 and MNIST [23, 24, 35, 51]. Although these datasets are useful for initial experimentation, they do not adequately capture the properties of real-world FL deployments, such as statistical heterogeneity, decentralized data ownership, domain-specific constraints, and practical system limitations.

To address these challenges, we propose Flower Hub, an executable and reproducible benchmarking platform for federated and decentralized AI. Flower Hub decouples infrastructure logic from application logic by leveraging the underlying Flower FL infrastructure [3]. This design enables researchers to focus on the essential components of their work, such as model training, algorithm design, and evaluation, while making benchmarks lightweight to share because no custom infrastructure code is required. Flower Hub provides portability by command across simulation and deployment settings without code changes. It also introduces a standardized benchmark package that specifies training, evaluation, system configuration, and versioning, thereby improving reproducibility and reusability across benchmarks. In addition, Flower Hub provides users with tools to measure system-level metrics, such as communication cost and latency. Together, these features establish Flower Hub as an open-source community ecosystem in which researchers can publish executable federated benchmarks that others can run, evaluate, and build upon. In this sense, Flower Hub represents an "*App Store moment*" for federated and decentralized benchmarking.

To demonstrate the utility of Flower Hub, we introduce five realistic FL benchmarks spanning five domains: healthcare, finance, security, automotive, and legal services. Our selected benchmarks also cover various data modalities, including text, tabular data, imaging, and audio. Using these benchmarks, we evaluate six widely used FL optimization algorithms in both simulation and deployment settings. The resulting analysis provides insights into the feasibility and effectiveness of deploying FL algorithms in realistic settings. More broadly, these benchmarks can help accelerate the development of inclusive, privacy-preserving, and domain-specialized models for real-world applications.

2 Flower Hub

Flower Hub[2] is an open-source benchmarking platform for federated learning. It supports the discovery, distribution, and execution of benchmarks across both simulation and deployment environments. In Flower Hub, each benchmark is represented as an executable application, allowing users to publish, download, and run benchmarks through a unified interface. This design shifts FL benchmarking from isolated, one-off research efforts toward a collaborative ecosystem that promotes reproducibility and reusability (Figure 1). Further details on using Flower Hub are provided in Appendix C.

2.1 Decoupling Application Logic from Infrastructure

Flower Hub is built on top of the Flower infrastructure, in which a long-running *SuperLink* process represents the central server, while multiple long-running *SuperNode* processes represent federated clients. Further details of the Flower infrastructure are provided in the Appendix A. By relying on this shared infrastructure layer, Flower Hub allows researchers to focus exclusively on application logic. Specifically, the *ServerApp* defines the overall FL workflow, including aggregation strategies and, when applicable, centralized evaluation. The *ClientApp* defines local training, local evaluation, and the corresponding data pipeline. Benchmark apps on Flower Hub therefore need only to include a *ServerApp* and a *ClientApp*. This design makes apps lightweight to distribute and allows researchers to focus on algorithmic and experimental design without implementing or maintaining infrastructure code.

2.2 One-Command Portability Across Simulation and Deployment

Benchmark apps on Flower Hub can be executed in both simulation and deployment settings without code modifications, using a unified command: *flwr run*. In simulation mode, the Flower Simulation Engine runs the full FL workflow locally by launching a *SuperLink* and multiple *SuperNodes*, enabling efficient experimentation even on a single machine. In deployment mode, the same workflow is executed across distributed environments, where independently launched *SuperNodes* connect to a *SuperLink*. The app is downloaded and executed without changes, significantly reducing the gap between development and real-world deployment.

To support both modes, each benchmark provides dedicated data-loading logic. Simulation uses the Flower Datasets library (Appendix B) for flexible partitioning, while deployment loads local data from disk. Consistent partitioning and preprocessing ensure alignment between simulation and deployment settings.

2.3 Standardized Benchmark Package

Every benchmark app on Flower Hub follows a standardized package structure with a specific schema, explicit configurations, pinned dependencies, and machine-readable metadata. The configuration file includes both task-related and system-related settings. Task-related configurations specify the dataset, model architecture, training hyperparameters, and aggregation strategy. System configurations specify the partitioning scheme, client participation rate, number of training rounds, and client resource requirements for simulation. Defining these components in a single configuration file improves reproducibility and makes benchmarks easier to share, reuse, and extend (see Appendix D for an example).

Flower Hub also provides a flexible version-control mechanism for benchmark apps. Publishers can release updated versions of their benchmarks, while users can download and run specific versions from the Hub. In addition, Flower Hub can automatically identify a compatible app version based on the user's local environment, further improving usability and reproducibility across different systems.

[2] `https://flower.ai/apps`

Table 1: Summary of the federated learning benchmark datasets, including their application domain, data modality, number of samples, number of clients, and FL setting.

Datasets	Domain	Data modality	Total # samples	# clients	FL type
fed-brats[3]	Medical	Image	1.6 K	5	Cross-silo
fed-fraud-paysim-banks[4]	Finance	Tabular	6.4 M	5	Cross-silo
fed-legal[5]	Law	Text	83.6 K	5	Cross-silo
fed-phishing-urls[6]	Security	URL	1.1 M	100	Cross-device
fed-urbansound8k[7]	Sensing	Audio	8.7 K	50	Cross-device

2.4 System and ML Performance Monitoring

Monitoring is essential in FL, both to ensure reliable execution and to understand algorithmic behavior. Flower Hub provides a dedicated monitoring framework for system-level metrics in real time. These metrics include end-to-end round latency, client training time, server aggregation time, communication cost, and client-side CPU and GPU memory utilization. Such measurements enable researchers to assess system health during benchmarking, diagnose performance bottlenecks, and better understand the practical trade-offs of different FL configurations.

At the same time, benchmark apps can report task-specific machine learning metrics through their *ServerApp* and *ClientApp* logic. These metrics can include local and centralized loss or accuracy, convergence statistics, class-wise performance, and other algorithm-specific quantities such as gradient- or pseudo-gradient-related noise measures. This separation preserves Flower Hub's role as a general benchmarking and execution platform while allowing each benchmark to expose the ML observables most relevant to its task and method. Together, system-level and ML-level measurements provide a more complete basis for evaluating federated learning benchmarks.

3 Benchmark Development

To demonstrate the utility of Flower Hub, we introduce five realistic FL benchmark tasks: medical image segmentation, financial fraud detection, legal instruction tuning, phishing URL detection, and on-device audio tagging. These tasks involve sensitive, distributed data across institutions or user devices, making FL a natural framework for collaborative training. Collectively, these benchmarks cover both cross-silo and cross-device FL scenarios, spanning image, tabular, audio, and text modalities. For each task, we establish a complete training and evaluation pipeline and use it to benchmark six widely adopted FL optimization algorithms under standardized experimental settings.

3.1 FL Dataset Construction

We carefully select five realistic FL datasets designed to reflect practical deployment environments. The details of each benchmark are described below, with a summary of the proposed datasets presented in Table 1. A summary of the training and evaluation pipelines for the benchmark tasks is provided in Table 2. The distribution of samples across clients is illustrated in Figure 2, while client-level label distributions are provided in the Appendix F.1.

Medical image segmentation[3]. We use the BraTS-GLI dataset [30], which comprises multimodal glioma MRI scans with four input sequences: T1-native, T1-contrast enhanced, T2-weighted, and T2-FLAIR, together with corresponding tumor segmentation masks. To emulate a realistic federated setting, the data are partitioned according to acquisition site, with each site representing an individual federated client. Within each site, samples are randomly divided into training and held-out evaluation subsets using an approximate 80:20 split, while ensuring at least one training sample for sites containing multiple cases. This site-wise partitioning preserves institutional heterogeneity and avoids artificial mixing of data across centers, thereby better reflecting the non-IID conditions commonly encountered in multi-institutional medical imaging federated learning.

Financial fraud detection[4]. We use a PaySim-style synthetic fraud detection dataset [21] consisting of transaction-level records with binary fraud labels. To emulate a realistic federated banking environment, transactions are partitioned across five simulated banks using account-level assignment

[3] `https://huggingface.co/datasets/flwrlabs/fed-brats`

[4] `https://huggingface.co/datasets/flwrlabs/fed-fraud-paysim-banks`

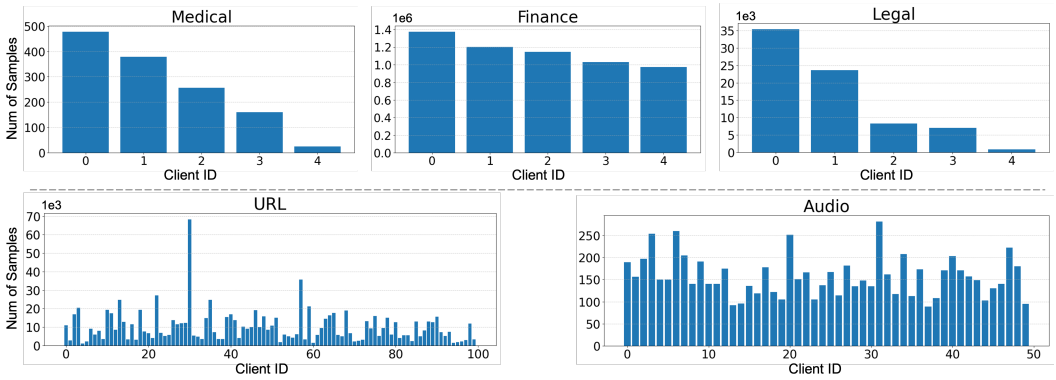


Figure 2: Distribution of sample counts across clients for the five proposed benchmark datasets.

based on the originating account identifier. This ensures that all transactions associated with a given account remain within a single client, preventing account-level leakage across institutions. Client heterogeneity is introduced through predefined quotas that vary bank size, fraud prevalence, and active non-fraud account composition. Within each bank, a stratified train–test split allocates approximately 10% of fraudulent and non-fraudulent transactions to the held-out test set. The resulting benchmark captures key characteristics of cross-institutional financial FL, including client imbalance, heterogeneous fraud distributions, and institution-specific account ownership.

Legal instruction tuning[5]. We construct a federated legal supervised fine-tuning benchmark from five legal NLP sources: LexGLUE LEDGAR, LexGLUE CaseHOLD, LexGLUE Unfair Terms of Service, LexGLUE SCOTUS [6], and merged LegalBench [12] contract natural language inference tasks. To emulate a realistic federated legal setting, each source task is assigned to a separate client silo, such that clients differ not only in data samples but also in task formulation, label space, and legal subdomain. This creates a strongly non-IID partition reflecting institutional specialization across legal organizations. Each client's dataset is converted into a chat-style supervised fine-tuning format and partitioned locally into training, validation, and test subsets using a deterministic 90:5:5 split. Global validation and test sets are then formed by concatenating held-out subsets from all clients, preserving client-level heterogeneity while enabling centralized evaluation.

Phishing URL detection[6]. Phishing detection in real-world cybersecurity systems is inherently decentralized, with URL data distributed across users, organizations, and network endpoints, each exhibiting distinct traffic patterns and threat exposure. This leads to strong heterogeneity in both feature distributions and phishing prevalence across clients, making it a natural and challenging setting for FL. To reflect these characteristics, we construct a federated phishing URL detection benchmark by merging two public Hugging Face datasets containing URL strings with binary phishing labels [2, 19]. Prior to partitioning, URLs are canonicalized and deduplicated. We then allocate samples across 100 simulated clients to mimic decentralized data ownership. Client heterogeneity is introduced through imbalanced client sizes, varying phishing rates sampled from a Beta distribution, and feature-level skew based on URL properties such as suspicious domains, URL shorteners, IP-based hosts, and complex query patterns. Each client is subsequently split into local training and test sets using a 90:10 partition. This design captures both label imbalance and feature heterogeneity observed in real-world phishing detection, providing a realistic and challenging benchmark for FL.

On-device audio tagging[7]. We construct a federated version of the UrbanSound8K audio classification dataset [40]. To emulate a realistic federated acoustic sensing scenario, samples are distributed across 50 clients while preserving fsID groups, such that all clips associated with the same source identifier remain within a single client. This prevents leakage of related recordings across clients and reflects settings in which data originate from distinct devices or environments. Client heterogeneity is introduced through imbalanced log-normal client sizes and client-specific label preferences sampled from a Dirichlet distribution, with assignment performed at the fsID-group level. Each client is subsequently partitioned into local training and test subsets using an approximate 90:10 split, with

[5] https://huggingface.co/datasets/flwrlabs/fed-legal
[6] https://huggingface.co/datasets/flwrlabs/fed-phishing-urls
[7] https://huggingface.co/datasets/flwrlabs/fed-urbansound8k

Table 2: Summary of the training and evaluation pipelines for the benchmark tasks. "CLS" denotes classification, and "Trainable" refers to the number of trainable model parameters.

Tasks	Task type	Model	Model size	Trainable	Evaluation metrics
Medical imaging	Segmentation	3D U-Net	1.77 M	1.77 M	Dice score
Financial fraud detection	Binary CLS	MLP	10.75 K	10.75 K	PR-AUC
Legal instruction tuning	Instruction tuning	LLM	3.08 B	7.67 M	F1 score
Phishing URL detection	Binary CLS	1D CNN	0.57 M	0.57 M	ROC-AUC
On-device audio tagging	CLS	CNN	24.40 K	24.40 K	Accuracy

label-stratified allocation where feasible. The resulting benchmark preserves source-level grouping, client imbalance, and label-distribution heterogeneity, providing a realistic testbed for cross-device federated audio classification.

3.2 Training and Evaluation Pipeline

To validate the proposed federated datasets, we establish lightweight training pipelines for each task together with task-appropriate evaluation metrics. All benchmark applications have been published on Flower Hub[2] for public access and reproducible evaluation (Appendix E).

Federated training. For medical image segmentation, we employ a 3D U-Net [42] trained with cross-entropy loss using the Adam optimizer. For financial fraud detection, we use a multi-layer perceptron (MLP) with engineered tabular features, incorporating weighted sampling and loss weighting to address severe class imbalance [31]. For legal instruction tuning, we adopt SmolLM3-3B as the base model and perform parameter-efficient fine-tuning using quantized LoRA, where only adapter weights are communicated between the server and clients [47]. In these cross-silo settings, all five clients participate in each training round; we run 10 rounds for the legal instruction tuning task and 20 rounds for the medical and financial tasks.

For cross-device settings, phishing URL detection is implemented using a one-dimensional CNN trained with the AdamW optimizer [22], with 10 out of 100 clients sampled per round over 20 rounds. For on-device audio tagging, raw audio signals are converted into log-mel spectrograms and processed using a compact CNN classifier [8]; 15 out of 50 clients are sampled per round, and training is conducted for 100 rounds. Across all tasks, we employ a cosine annealing learning-rate schedule. Detailed hyperparameter configurations are provided in the Appendix F.2.

Evaluation metrics. For medical image segmentation, we report the Dice score as the primary evaluation metric. For financial fraud detection, we use PR-AUC to account for extreme class imbalance. For legal instruction tuning, performance is evaluated using token-level F1 score. For phishing URL detection, we report ROC-AUC, and for audio tagging, we use classification accuracy.

Federated optimization strategies. We benchmark six widely used aggregation methods across all tasks. **FedAvg** [29] serves as the baseline, while **FedProx** [25] mitigates client drift under non-IID data. **FedAvgM** [14] introduces server-side momentum for improved convergence. The FedOpt family, **FedAdam**, **FedAdagrad**, and **FedYogi** [34], applies adaptive optimization on the server side with coordinate-wise learning rates and improved stability mechanisms. For fair comparison, we use a shared set of hyperparameters and avoid extensive task-specific tuning.

System performance evaluation. In addition to model performance, we record system-level metrics to characterize computational cost, communication overhead, memory usage, and runtime. For each client, we measure peak CPU and GPU memory consumption during local training to estimate hardware requirements. We also track client training time per round, capturing the computational cost of model updates. To assess overall efficiency, we record the end-to-end wall-clock duration of each federated round and decompose it into training and non-training time, where the latter includes communication, synchronization, and orchestration overheads. Finally, we measure the total communication volume exchanged between the server and clients, including model broadcasts and update aggregation. Together, these metrics provide a comprehensive view of the system-level characteristics of each benchmark.

Deployment evaluation. Deployment evaluation verifies that each benchmark can run beyond simulation using Flower's deployment workflow. In this setting, clients are launched as independent SuperNodes with local data partitions, reflecting real-world federated environments. We conduct a full deployment run using the prescribed training rounds and client participation settings, exercising the

Table 3: Comparison of aggregation strategies across five tasks. The evaluation metrics are Dice score, PR-AUC, token-level F1 score, ROC-AUC, and accuracy for medical image segmentation, financial fraud detection, legal instruction tuning, phishing URL detection, and audio tagging, respectively. All experiments are conducted over three runs with different random seeds, reporting the mean performance with standard deviation in parentheses. The best results are highlighted in **bold**, and the second-best are underlined.

	Medical (%)	Finance (%)	Legal (%)	URL (%)	Audio (%)
FedAvg	78.57 (0.09)	**56.40 (0.45)**	72.15 (0.63)	98.45 (0.05)	48.70 (1.03)
FedProx	**78.58 (0.28)**	55.21 (1.39)	72.29 (1.44)	**98.48 (0.01)**	**52.56 (3.71)**
FedAvgM	78.31 (0.39)	54.86 (0.56)	71.96 (1.19)	98.38 (0.02)	47.44 (2.15)
FedAdam	73.57 (3.76)	14.71 (17.02)	70.02 (1.85)	93.64 (0.20)	25.78 (5.68)
FedAdagrad	64.02 (10.29)	44.80 (2.37)	**72.97 (1.50)**	91.02 (0.47)	22.54 (2.66)
FedYogi	75.18 (1.85)	2.88 (0.10)	70.19 (1.41)	94.80 (0.18)	27.80 (5.70)

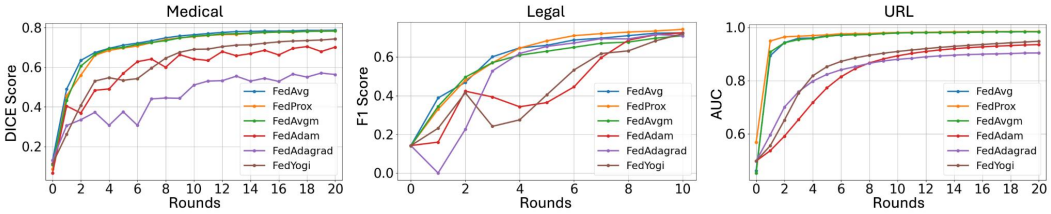


Figure 3: Changes in evaluation metrics over FL rounds for the medical image segmentation, legal instruction tuning, and phishing URL detection tasks.

complete distributed workflow, including server and client applications, data loading, local training, aggregation, communication, and metric reporting. A run is considered successful if the configured training procedure completes and participating clients return valid updates. Where possible, we compare deployment results with simulation under the same configuration to ensure consistency in partitioning, initialization, and metric reporting. Overall, this evaluation validates both reproducibility in simulation and executability under realistic deployment conditions.

4 Experimental Evaluation of Benchmarks

We evaluate each benchmark along three dimensions: model performance, system efficiency, and deployment behavior. These respectively assess predictive quality, computational and communication costs, and the ability to run reliably beyond simulation using independently deployed Flower SuperNodes without code modifications.

4.1 Model Performance Evaluation

Our primary objective is to demonstrate Flower Hub as a benchmarking platform rather than to exhaustively optimize model performance. Accordingly, we report baseline results without extensive hyperparameter tuning; all benchmarks are publicly available for further community exploration. Table 3 summarizes the evaluation results across the five benchmark tasks.

Across aggregation strategies, no single method consistently dominates, reflecting the challenges of non-IID client data. FedProx achieves the best performance in most cases, with FedAvg and FedAvgM performing similarly and showing comparable convergence behavior (Figure 3). In contrast, the FedOpt family (FedAdam, FedAdagrad, and FedYogi) often exhibits suboptimal or unstable performance, except in the legal instruction tuning task, where strong pretrained initialization improves results. This suggests that FedOpt methods are sensitive to hyperparameters (e.g., server learning rate and momentum), particularly when training from scratch [34, 43]. The issue is further exacerbated in tasks like financial fraud detection, where extreme class imbalance increases optimization difficulty (Figure 8).

Performance also varies across tasks. In cross-silo settings, medical image segmentation and legal instruction tuning achieve relatively strong results (generally above 70%), benefiting from full client participation and sufficient aggregate data. In contrast, financial fraud detection performs worse due to severe non-IID characteristics. In cross-device settings, phishing URL detection achieves high perfor-

Table 4: System performance across benchmark tasks in simulation mode, measured using FedAvg. The legal task is executed on an H200 GPU, while all other tasks use an A100 GPU. "Comm." denotes the total communication cost over all training rounds. "CPU Mem." and "GPU Mem." represent the average peak memory usage across rounds. "Training" indicates the cumulative client training time, and "Latency" refers to the total end-to-end time required to complete federated training.

Tasks	Comm. (GB)	CPU Mem. (GB)	GPU Mem. (GB)	Training (s)	Latency (s)
Medical	3.85	36.23	2.45	10.05 K	40.52 K
Finance	0.01	2.78	0.02	363.01	13.25 K
Legal	2.86	9.47	21.24	14.63 K	42.43 K
URL	0.85	1.91	0.31	27.34	244.00
Audio	0.28	7.33	0.10	156.67	2.17 K

mance (above 90% across methods), reflecting the relative simplicity of the task. Audio tagging, however, remains challenging, with all methods achieving only modest accuracy (slightly above 50%), due to both heterogeneous client distributions and the intrinsic difficulty of audio representation learning.

Overall, these results establish baseline performance on the proposed benchmarks, providing a reference point for future research. They also illustrate the ease of conducting federated benchmarking experiments using Flower Hub.

4.2 System Performance Evaluation

Flower Hub includes a built-in system monitor for real-time performance tracking during benchmark execution. Table 4 summarizes system metrics across tasks in simulation mode using FedAvg. Differences across aggregation strategies are negligible, and thus omitted.

Communication cost is primarily determined by the size of the trainable model parameters. For example, the Medical task incurs the highest communication overhead due to the large 3D U-Net model, whereas the Finance task requires minimal communication (on the order of 0.01 GB) owing to its lightweight MLP architecture. CPU and GPU memory utilization reflect both data loading and model training processes, suggesting potential opportunities for optimization through improved hardware utilization and data pipeline design.

Client-side training time is influenced by both dataset size and model complexity. As expected, the legal tuning task, based on LLM fine-tuning, exhibits the longest training time, while the URL task requires comparatively little computation. Notably, training time does not dominate the total end-to-end latency across tasks. Instead, data preprocessing and communication overhead contribute significantly to overall execution time in federated settings. These observations highlight the importance of jointly considering model design, data handling, and system-level efficiency when evaluating federated learning workloads.

4.3 Deployment Evaluation

All benchmark applications on Flower Hub can be executed in a deployment setting without requiring code modifications. To validate this capability, we construct a cross-region deployment topology (Figure 10 & Table 6). Figure 4 presents the evolution of wall-clock time and cumulative communication cost over federated learning rounds. The results illustrate the relative contribution of client-side training time within each round, as well as the overall system behavior during distributed execution. In particular, communication cost increases approximately linearly with the number of rounds, reflecting the use of a fixed model architecture and consistent parameter exchange across iterations. These findings demonstrate the practical deployability of Flower Hub benchmarks and provide insight into the temporal and communication characteristics of real-world federated training.

5 Related Work

Research on federated benchmarking can be broadly grouped into three strands: benchmark suites, frameworks and tooling, and evaluation or reproducibility studies.

Benchmark suites. Early work such as LEAF [4] established a foundation of federated datasets, metrics, and reference implementations. Subsequent efforts have expanded realism and domain coverage, including OARF [15], FLBench [26], FedScale [20], FLamby [32], and pFL-Bench [7]. More recent

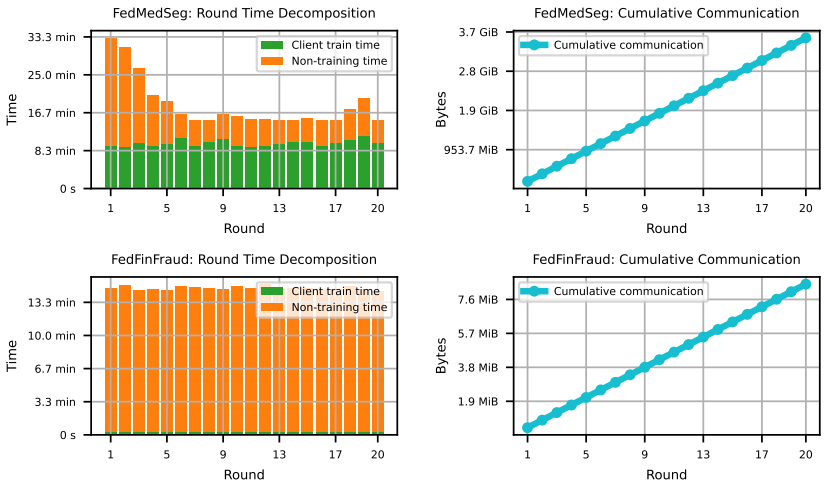


Figure 4: Deployment profiles for FedMedSeg and FedFinFraud under FedAvg. Stacked bars show per-round wall-clock time split into client training and non-training overhead (e.g., preprocessing, communication, and synchronization). Right panels show cumulative communication over FL rounds.

benchmarks target emerging settings such as multimodal learning, IoT data, and federated large language models [9, 1, 46]. While these works improve realism and diversity, they are typically released as standalone repositories with tightly coupled infrastructure, limiting reuse and standardization.

Frameworks and tooling. A large body of work focuses on simplifying the implementation and execution of federated learning, including TensorFlow Federated [41], FedLab [48], FLUTE [11], FedML [13], EasyFL [52], and FederatedScope [44]. Systems such as OpenFL [36], NVFlare [37], and APPFL [38] further emphasize deployment and privacy-preserving workflows. These frameworks reduce engineering overhead but primarily serve as execution substrates rather than defining benchmarks as standardized, portable artifacts.

Evaluation and reproducibility. Prior work has also addressed evaluation methodology and reproducibility. FedEval [5] proposed a comprehensive evaluation framework, while UniFed [27] introduced schema-based comparability across multiple FL systems. MedPerf [18] moves closer to executable benchmarking with standardized packaging and real-world deployment in healthcare. Other studies [33] highlight the gap between simulation and deployment. However, these efforts either focus on evaluation standardization or domain-specific solutions, leaving benchmark packaging and portability largely unaddressed.

Against this background, Flower Hub is best understood not as another benchmark suite or framework, but as a benchmark platform built on top of Flower [3]. It decouples application logic from infrastructure, packages benchmarks as executable and versioned units, and enables seamless execution across simulation and deployment. This complements existing work by addressing the missing systems layer for benchmark exchange, portability, and long-term reusability.

6 Conclusion

We presented Flower Hub, a reproducible benchmarking platform for federated and decentralized learning that packages benchmarks as executable, versioned applications with standardized metadata and evaluation workflows. By decoupling application logic from infrastructure, Flower Hub enables seamless execution across simulation and deployment without code changes. We demonstrated this approach through five realistic benchmarks spanning cross-silo and cross-device settings across multiple domains. Results across six aggregation strategies show strong task-dependent performance variation, while built-in system monitoring provides visibility into communication, memory, runtime, and deployment behavior. These findings highlight Flower Hub's potential as a foundation for reproducible and portable federated benchmarking.

Limitations and future work. Our study emphasizes baseline evaluation rather than extensive optimization. The current benchmarks do not yet cover all FL settings, such as personalization, privacy, robustness, or highly constrained environments, and deployment experiments remain limited in scale. Future work will expand the benchmark suite, incorporate broader evaluations, and support large-scale, long-running deployments.

References

[1] S. Alam, T. Zhang, T. Feng, H. Shen, Z. Cao, D. Zhao, J. Ko, K. Somasundaram, S. S. Narayanan, S. Avestimehr, et al. Fedaiot: A federated learning benchmark for artificial intelligence of things. *arXiv preprint arXiv:2310.00109*, 2023.

[2] E. Alvarado. Phishing Dataset, n.d. URL `https://huggingface.co/datasets/ealvaradob/phishing-dataset`.

[3] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[4] 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.

[5] D. Chai, L. Wang, L. Yang, J. Zhang, K. Chen, and Q. Yang. Fedeval: A holistic evaluation framework for federated learning. *arXiv preprint arXiv:2011.09655*, 2020.

[6] I. Chalkidis, A. Jana, D. Hartung, M. Bommarito, I. Androutsopoulos, D. Katz, and N. Aletras. Lexglue: A benchmark dataset for legal language understanding in english. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4310–4330, 2022.

[7] D. Chen, D. Gao, W. Kuang, Y. Li, and B. Ding. pfl-bench: A comprehensive benchmark for personalized federated learning. *Advances in Neural Information Processing Systems*, 35: 9344–9360, 2022.

[8] K. Choi, G. Fazekas, and M. Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.

[9] T. Feng, D. Bose, T. Zhang, R. Hebbar, A. Ramakrishna, R. Gupta, M. Zhang, S. Avestimehr, and S. Narayanan. Fedmultimodal: A benchmark for multimodal federated learning. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pages 4035–4045, 2023.

[10] Y. Gao, M. R. Scamarcia, J. Fernandez-Marques, M. Naseri, C. S. Ng, D. Stripelis, Z. Li, T. Shen, J. Bai, D. Chen, et al. Flowertune: A cross-domain benchmark for federated fine-tuning of large language models. *arXiv preprint arXiv:2506.02961*, 2025.

[11] M. H. Garcia, A. Manoel, D. M. Diaz, F. Mireshghallah, R. Sim, and D. Dimitriadis. Flute: A scalable, extensible framework for high-performance federated learning simulations. *arXiv preprint arXiv:2203.13789*, 2022.

[12] N. Guha, J. Nyarko, D. Ho, C. Ré, A. Chilton, A. Chohlas-Wood, A. Peters, B. Waldon, D. Rockmore, D. Zambrano, et al. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in neural information processing systems*, 36:44123–44279, 2023.

[13] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[14] T.-M. H. Hsu, H. Qi, and M. Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

[15] S. Hu, Y. Li, X. Liu, Q. Li, Z. Wu, and B. He. The oarf benchmark suite: Characterization and implications for federated learning systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(4):1–32, 2022.

[16] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini. A survey on federated learning for resource-constrained iot devices. *IEEE Internet of Things Journal*, 9(1):1–24, 2021.

[17] S. Ji, Y. Tan, T. Saravirta, Z. Yang, Y. Liu, L. Vasankari, S. Pan, G. Long, and A. Walid. Emerging trends in federated learning: From model fusion to federated x learning. *International Journal of Machine Learning and Cybernetics*, 15(9):3769–3790, 2024.

[18] A. Karargyris, R. Umeton, M. J. Sheller, A. Aristizabal, J. George, A. Wuest, S. Pati, H. Kassem, M. Zenk, U. Baid, et al. Federated benchmarking of medical artificial intelligence with medperf. *Nature machine intelligence*, 5(7):799–810, 2023.

[19] kmack. Phishing URLs Dataset, 2024. URL `https://huggingface.co/datasets/kmack/Phishing_urls`.

[20] F. Lai, Y. Dai, S. Singapuram, J. Liu, X. Zhu, H. Madhyastha, and M. Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International conference on machine learning*, pages 11814–11827. PMLR, 2022.

[21] P. Lalwani. Synthetic Financial Datasets for Fraud Detection, 2024. URL `https://huggingface.co/datasets/purulalwani/Synthetic-Financial-Datasets-For-Fraud-Detection`.

[22] H. Le, Q. Pham, D. Sahoo, and S. C. Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, 2018.

[23] Q. Li, B. He, and D. Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10713–10722, 2021.

[24] Q. Li, Y. Diao, Q. Chen, and B. He. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th international conference on data engineering (ICDE)*, pages 965–978. IEEE, 2022.

[25] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[26] Y. Liang, Y. Guo, Y. Gong, C. Luo, J. Zhan, and Y. Huang. Flbench: A benchmark suite for federated learning. In *BenchCouncil International Federated Intelligent Computing and Block Chain Conferences*, pages 166–176. Springer, 2020.

[27] X. Liu, T. Shi, C. Xie, Q. Li, K. Hu, H. Kim, X. Xu, T.-A. Vu-Le, Z. Huang, A. Nourian, et al. Unifed: All-in-one federated learning platform to unify open-source frameworks. *arXiv preprint arXiv:2207.10308*, 2022.

[28] P. M. Mammen. Federated learning: Opportunities and challenges. *arXiv preprint arXiv:2101.05428*, 2021.

[29] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[30] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, et al. The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993–2024, 2014.

[31] M. K. Mishra and R. Dash. A comparative study of chebyshev functional link artificial neural network, multi-layer perceptron and decision tree for credit card fraud detection. In *2014 International Conference on Information Technology*, pages 228–233. IEEE, 2014.

[32] J. Ogier du Terrail, S.-S. Ayed, E. Cyffers, F. Grimberg, C. He, R. Loeb, P. Mangold, T. Marchand, O. Marfoq, E. Mushtaq, et al. Flamby: Datasets and benchmarks for cross-silo federated learning in realistic healthcare settings. *Advances in Neural Information Processing Systems*, 35:5315–5334, 2022.

[33] C. Prigent, K. Keahey, A. Costan, L. Cudennec, and G. Antoniu. On the reproducibility challenges of federated learning: Investigating the gap between simulation, emulation and real-world deployments. In *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 185–194. IEEE, 2025.

[34] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[35] Y. A. U. Rehman, Y. Gao, P. P. B. De Gusmao, M. Alibeigi, J. Shen, and N. D. Lane. L-dawa: Layer-wise divergence aware weight aggregation in federated self-supervised visual representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16464–16473, 2023.

[36] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, et al. Openfl: An open-source framework for federated learning. *arXiv preprint arXiv:2105.06413*, 2021.

[37] H. R. Roth, Y. Cheng, Y. Wen, I. Yang, Z. Xu, Y.-T. Hsieh, K. Kersten, A. Harouni, C. Zhao, K. Lu, et al. Nvidia flare: Federated learning from simulation to real-world. *arXiv preprint arXiv:2210.13291*, 2022.

[38] M. Ryu, Y. Kim, K. Kim, and R. K. Madduri. Appfl: open-source software framework for privacy-preserving federated learning. In *2022 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 1074–1083. IEEE, 2022.

[39] K. Saab, T. Tu, W.-H. Weng, R. Tanno, D. Stutz, E. Wulczyn, F. Zhang, T. Strother, C. Park, E. Vedadi, et al. Capabilities of Gemini models in medicine. *arXiv preprint arXiv:2404.18416*, 2024.

[40] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044, 2014.

[41] TensorFlow Federated Developers. Tensorflow federated. Project documentation, 2019. URL `https://www.tensorflow.org/federated`. Framework introduced in 2019; accessed 2026-04-27.

[42] F. Wang, R. Jiang, L. Zheng, C. Meng, and B. Biswal. 3d u-net based brain tumor segmentation and survival days prediction. In *International MICCAI Brainlesion Workshop*, pages 131–141. Springer, 2019.

[43] X. Wu, F. Huang, Z. Hu, and H. Huang. Faster adaptive federated learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 10379–10387, 2023.

[44] Y. Xie, Z. Wang, D. Gao, D. Chen, L. Yao, W. Kuang, Y. Li, B. Ding, and J. Zhou. Federatedscope: A flexible federated learning platform for heterogeneity. *arXiv preprint arXiv:2204.05011*, 2022.

[45] H. Yang, X.-Y. Liu, and C. D. Wang. FinGPT: Open-Source Financial Large Language Models. *FinLLM Symposium at IJCAI 2023*, 2023.

[46] R. Ye, R. Ge, X. Zhu, J. Chai, Y. Du, Y. Liu, Y. Wang, and S. Chen. Fedllm-bench: Realistic benchmarks for federated learning of large language models. *Advances in Neural Information Processing Systems*, 37:111106–111130, 2024.

[47] S. Yue, W. Chen, S. Wang, B. Li, C. Shen, S. Liu, Y. Zhou, Y. Xiao, S. Yun, X. Huang, et al. Disc-lawllm: Fine-tuning large language models for intelligent legal services. *arXiv preprint arXiv:2309.11325*, 2023.

[48] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu. Fedlab: A flexible federated learning framework. *Journal of Machine Learning Research*, 24(100):1–7, 2023.

[49] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

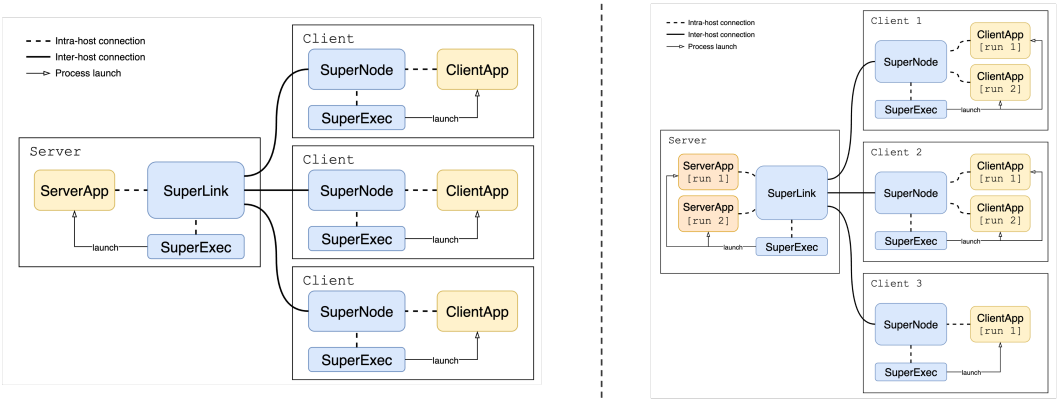
[50] Z. Zhang, X. Hu, J. Zhang, Y. Zhang, H. Wang, L. Qu, and Z. Xu. Fedlegal: The first real-world federated learning benchmark for legal nlp. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3492–3507, 2023.

[51] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[52] W. Zhuang, X. Gan, Y. Wen, and S. Zhang. Easyfl: A low-code federated learning platform for dummies. *IEEE Internet of Things Journal*, 9(15):13740–13754, 2022.

A Flower Infrastructure

Flower's infrastructure separates long-lived system components responsible for networking and coordination from short-lived application components that implement project-specific federated learning logic. In a typical federated learning system, a central server coordinates training while multiple clients execute tasks on local data and return results. Flower adopts a hub-and-spoke architecture in which both server-side and client-side functionalities are divided into infrastructure processes and application processes. This separation enables multiple federated learning applications to share the same federation while using different models, hyperparameters, aggregation strategies, or machine learning frameworks (Figure 5). Further details can be found in the Flower Documentation[8].



(a) The basic Flower architecture for federated learning

(b) Multi-run federated learning architecture with Flower

Figure 5: Overview of Flower infrastructure.

SuperLink. On the server side, the core infrastructure component is the *SuperLink*, a long-running process that acts as the communication hub of the federation. It forwards task instructions from the server-side application to connected client nodes and collects their results. Users interact with the SuperLink via the Flower command-line interface, while client-side SuperNodes connect to it through the Fleet API. The SuperLink is responsible for coordination and communication, rather than implementing learning logic.

ServerApp. The server-side learning logic is encapsulated in the *ServerApp*, a short-lived process that defines the behavior of a federated learning run. This includes client selection, configuration, aggregation of client updates, and the implementation of federated optimization strategies (e.g., FedAvg). The ServerApp communicates with the SuperLink through the ServerAppIO API, allowing it to exchange messages with clients without managing low-level networking details.

SuperNode. On the client side, the corresponding infrastructure component is the *SuperNode*, a long-running process that connects to the SuperLink, retrieves tasks, executes them, and returns results. Each SuperNode represents a participating client environment, such as a device, machine, or institutional data silo. Importantly, SuperNodes initiate outbound connections to the SuperLink and do not require inbound connectivity, making them suitable for deployment in restricted or firewall-protected environments.

ClientApp. Client-side computation is implemented in the *ClientApp*, a short-lived process containing application-specific logic such as local training, evaluation, preprocessing, and access to local data. The ClientApp is executed on demand when a SuperNode is selected to participate in a round. It communicates with the SuperNode through the ClientAppIO API, while the SuperNode manages all network communication with the SuperLink. This separation allows the ClientApp to focus solely on local computation and data handling.

SuperExec. Flower also provides *SuperExec*, a long-running process responsible for scheduling, launching, and managing application processes such as ServerApp and ClientApp. On the server side, SuperExec manages ServerApp instances in coordination with the SuperLink; on the client

[8] `https://flower.ai/docs/framework/index.html`

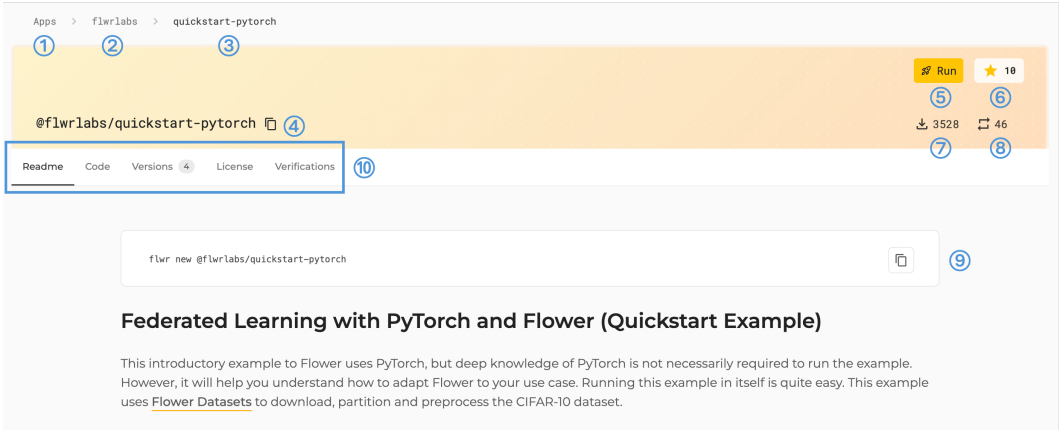


Figure 6: Example of an application detail page on Flower Hub. ① indicates the navigation link to the main applications page. ② denotes the publisher account name, which links to the publisher's profile page. ③ displays the application name, and ④ shows the full application identifier in the `@account/app` format. ⑤ provides a button to execute the application on SuperGrid. ⑥ reports the number of stars received by the application, while ⑦ and ⑧ indicate the total number of downloads and executions, respectively. ⑨ presents the command for fetching the application to a local environment. ⑩ corresponds to the information panel, including the README, code details, versioning, license, and verification metadata. The interface shown reflects the design at the time of publication and may evolve in future versions.

side, it manages ClientApp instances in coordination with the SuperNode. By default, SuperExec is launched automatically, although it can also be managed as a separate component if needed.

During a deployed run, the workflow proceeds as follows. A federation is first established by starting a SuperLink and connecting one or more SuperNodes. When a user invokes `flwr run`, the run is submitted through the SuperLink. The ServerApp is then launched to coordinate the training process, while selected SuperNodes execute the corresponding ClientApp locally. Communication between server and clients occurs indirectly via the SuperLink and SuperNodes, and results are returned to the ServerApp for aggregation.

Simulation and Deployment. Flower distinguishes between two execution modes: Simulation Runtime and Deployment Runtime. In the Simulation Runtime, `flwr run` executes the workflow locally using a managed SuperLink and multiple simulated clients, typically as worker processes on a single machine. This mode supports rapid prototyping, debugging, and algorithm validation. In the Deployment Runtime, SuperLink and SuperNodes operate as independent processes across distributed environments, communicate via secure (TLS-enabled) gRPC, and interact with real client-side data stored on devices or local systems. Crucially, the same ServerApp and ClientApp code can be used in both modes, with the runtime selected through configuration.

Overall, Flower's infrastructure follows a layered design: SuperLink and SuperNodes provide persistent communication and coordination, SuperExec manages application lifecycles, and ServerApp and ClientApp implement task-specific learning logic. This separation decouples reusable infrastructure from application code, enabling multiple federated learning applications—including those distributed via Flower Hub—to share the same federation while flexibly selecting different subsets of clients for each run.

B Flower Datasets

Flower Datasets, provided through the `flwr-datasets` library, supports data preparation for federated learning, federated analytics, and federated evaluation. It enables centralized datasets to be transformed into client-specific partitions, facilitating the simulation of federated environments with either independent and identically distributed (IID) or heterogeneous non-IID data distributions. Further details are available in the Flower Datasets Documentation[9].

[9] `https://flower.ai/docs/datasets/`

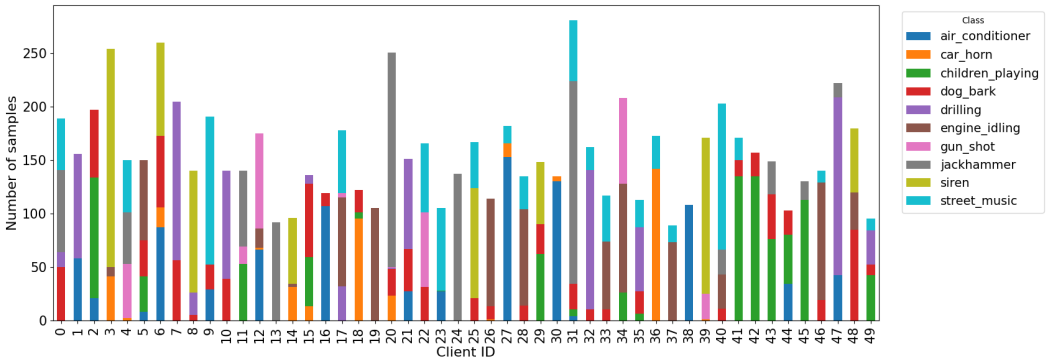


Figure 7: Number of samples and class distribution across clients in the proposed federated audio tagging dataset.

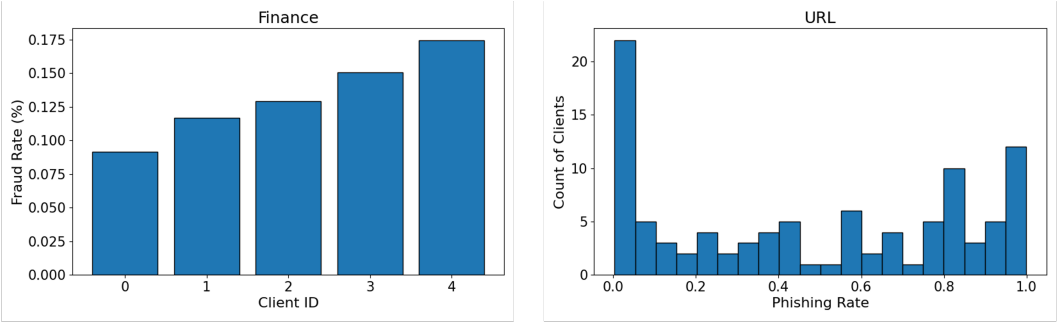


Figure 8: Distribution of fraud rates and phishing rates across clients for the financial fraud detection and phishing URL detection benchmarks.

The core abstraction is the `FederatedDataset`, which integrates dataset loading, preprocessing, and partitioning. It supports direct loading from the Hugging Face Datasets Hub and partitions selected dataset splits into multiple client subsets using configurable partitioning strategies. For instance, `IidPartitioner` enables IID partitioning, while `DirichletPartitioner` and `PathologicalPartitioner` are commonly used to simulate non-IID client distributions.

Flower Datasets also supports centralized evaluation by allowing the training split to be partitioned while keeping the test split unpartitioned. The resulting partitions are represented as Hugging Face `Dataset` objects, ensuring compatibility with widely used machine learning frameworks such as PyTorch, TensorFlow, NumPy, Pandas, and JAX.

Beyond public datasets, Flower Datasets can operate on local data sources, including CSV, JSON, image, audio, and in-memory formats. For deployment-oriented scenarios, the `flwr-datasets create` command can generate pre-partitioned datasets on disk, enabling individual SuperNodes to access their assigned local data partitions efficiently.

Overall, Flower Datasets provides a reproducible and flexible data preparation layer for Flower-based experiments, bridging public and local datasets with configurable partitioning strategies to support both controlled benchmarking and realistic federated learning scenarios.

C How-to Guide for Flower Hub

This section provides a practical guide to interacting with Flower Hub, including how to fetch, run, and publish benchmark applications. For additional details, we refer readers to the Flower Hub Documentation[10].

[10] `https://flower.ai/docs/hub/index.html`

Table 5: Selected hyperparameter configurations for the five benchmark tasks.

Tasks	num-rounds	fraction-train	local-epochs	batch-size	lr-max	lr-min	optimizer
Medical	20	1.0	3	4	1e-4	1e-5	Adam
Finance	20	1.0	2	1024	1e-4	1e-5	Adam
Legal	10	1.0	1	16	5e-5	1e-5	AdamW
URL	20	0.1	1	128	1e-3	1e-4	AdamW
Audio	100	0.3	2	32	1e-2	1e-4	Adam

C.1 Fetching an Application from Flower Hub

To obtain an application from Flower Hub, users must first install the Flower command-line interface, `flwr`, within their local Python environment. Once installed, an application can be fetched using the command `flwr new @account/app`. For example, `flwr new @flwrlabs/quickstart-pytorch` downloads the specified application and initializes it locally.

Applications on Flower Hub may include metadata specifying the required Flower App Bundle (FAB) format version and compatible Flower versions. As a result, the fetching process may fail prior to download if the local Flower installation is incompatible with the selected application. Users are therefore advised to verify version compatibility before fetching.

C.2 Running an Application from Flower Hub

Applications hosted on Flower Hub can be executed in either the Simulation Runtime or the Deployment Runtime without modifying the application source code, using the command `flwr run @account/app`.

For deployment, users must configure and launch one or more SuperNodes. Each SuperNode requires the address of the corresponding SuperLink and, when necessary, a path to local datasets specified via the `-node-config` option. Configuration details may vary across applications; therefore, users should consult the application's README for task-specific parameters and setup instructions.

C.3 Publishing an Application on Flower Hub

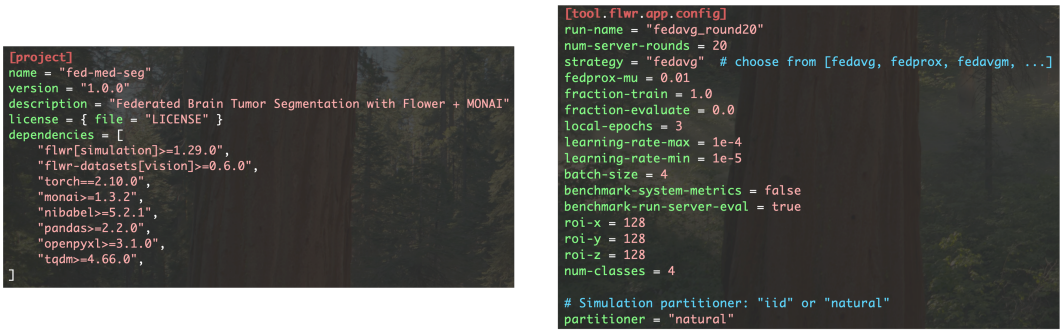
To publish an application on Flower Hub, it must be structured as a complete Flower project, including source code, documentation, and metadata. Applications should be designed to run in both Simulation and Deployment runtimes without modification to core logic. This is typically achieved by separating runtime-specific data-loading components from shared training logic. The accompanying README should document required configurations for both execution modes, including simulation settings and deployment data requirements.

Before publication, metadata must be defined in the `pyproject.toml` file. The `[project]` section should specify the application name, version, description, and license, while the `[tool.flwr.app]` section defines the publisher name and, if applicable, the FAB format and compatible Flower versions. The publisher field must match the Flower account used for publication. Notably, the application name is publicly visible and cannot be changed after initial release.

Flower Hub publishes source files rather than prebuilt artifacts. When executing `flwr app publish`, Flower recursively collects files from the project directory, applies filtering rules (e.g., file types and `.gitignore`), and validates the file set prior to upload. Validation includes checks on encoding, file count, individual file size, and total package size. If validation fails, the process terminates before upload.

To publish, users must first authenticate via `flwr login supergrid`, which initiates a browser-based login linked to their Flower account. The application can then be published using `flwr app publish <your-app-path>`. Upon successful publication, the application becomes accessible on Flower Hub at a URL of the form `https://flower.ai/apps/<account>/<app>/`.

Subsequent updates require modifying the source code, incrementing the `version` field in `pyproject.toml`, and re-running the publication command. Flower Hub maintains all published versions, enabling users to access and execute specific releases as needed.



```toml
[project]
name = "fed-med-seg"
version = "1.0.0"
description = "Federated Brain Tumor Segmentation with Flower + MONAI"
license = { file = "LICENSE" }
dependencies = [
    "flwr[simulation]>=1.29.0",
    "flwr-datasets[vision]>=0.6.0",
    "torch==2.10.0",
    "monai>=1.3.2",
    "nibabel>=5.2.1",
    "pandas>=2.2.0",
    "openpyxl>=3.1.0",
    "tqdm>=4.66.0",
]
```

```toml
[tool.flwr.app.config]
run-name = "fedavg_round20"
num-server-rounds = 20
strategy = "fedavg"  # choose from [fedavg, fedprox, fedavgm, ...]
fedprox-mu = 0.01
fraction-train = 1.0
fraction-evaluate = 0.0
local-epochs = 3
learning-rate-max = 1e-4
learning-rate-min = 1e-5
batch-size = 4
benchmark-system-metrics = false
benchmark-run-server-eval = true
roi-x = 128
roi-y = 128
roi-z = 128
num-classes = 4

# Simulation partitioner: "iid" or "natural"
partitioner = "natural"
```

Figure 9: Configuration example for the `fed-med-seg` application. (Left) The configuration block specifies metadata, including the application name, version, description, license, and pinned dependencies. (Right) The configuration block defines the hyperparameters used for federated training and evaluation.

C.4 Signing an Application from Flower Hub

Flower Hub supports application signing as a mechanism for attaching reviewer-generated verification metadata to published applications. An application must first be available on Flower Hub before it can be signed; however, the reviewer need not be the original publisher. This enables independent users or organizations to review and verify applications published by others. We note that application signing and related verification metadata are currently provided as preview features and may evolve over time.

The signing process is based on a cryptographic public–private key pair. The reviewer generates a key pair, securely stores the private key, and registers the corresponding public key in their Flower account profile. After authentication, the reviewer can invoke the Flower CLI command `flwr app review` to inspect and sign an application. Signing can target either the latest version (e.g., `@account/app`) or a specific version (e.g., `@account/app==x.y.z`).

During the review process, the Flower CLI downloads the FAB, unpacks it for inspection, and prompts the reviewer to confirm the signing operation. The reviewer then provides the path to their private key, and a cryptographic signature is generated over the FAB digest together with a timestamp. This signature is submitted to Flower Hub along with the application identifier and version, ensuring that verification is tied to a specific application artifact.

A key property of this mechanism is its decentralized trust model. Flower Hub does not restrict verification to application publishers or a central authority. Instead, any authenticated user with a registered signing key can review and sign an application, including those published by other accounts. The resulting signatures are displayed on the application page, allowing users to inspect which entities have verified the application and to determine which signers they trust.

This decentralized verification approach is particularly relevant for federated learning, where applications are executed across multiple organizations, devices, or data-owning sites. By enabling independent reviewers to sign the same application version, Flower Hub supports a layered trust model in which institutions can rely on signatures from trusted parties. The verification section thus serves as a transparency mechanism: while it does not guarantee trustworthiness, it provides verifiable evidence that specific reviewers have inspected and cryptographically endorsed a given application version.

D Configuration Example for Flower Hub Applications

Each benchmark application on Flower Hub follows a standardized package structure with a well-defined schema, explicit configurations, pinned dependencies, and machine-readable metadata. Figure 9 illustrates an example configuration for the medical image segmentation application.

The configuration consists of two main blocks. The first defines application metadata, including the version, license, and pinned dependencies, which ensure reproducibility across environments. The second specifies the hyperparameters used for federated training and evaluation, providing flexibility to adapt experimental settings while maintaining a consistent structure.

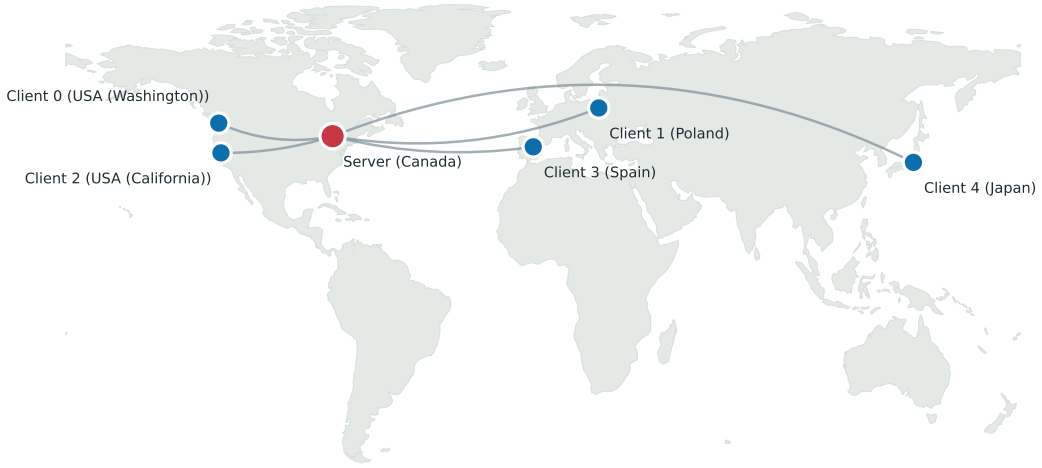


Figure 10: Cross-region deployment topology used in our deployment-mode experiments. The federated server was hosted in Canada, while five clients were deployed on Vast.ai instances in the USA, Poland, Spain, and Japan. Edges indicate client-server communication during federated training.

Table 6: Vast.ai instance configuration used for deployment-mode experiments.

Role	GPU	CPU	VRAM
Server	1× RTX 5090	AMD Ryzen 9 5900XT 16-Core	32 GB
Client 0	1× RTX 5090	AMD EPYC 9J14 96-Core	32 GB
Client 1	1× RTX 5090	AMD Ryzen 7 7700X 8-Core	32 GB
Client 2	1× RTX 5090	AMD EPYC 9654 96-Core	32 GB
Client 3	1× RTX 5090	AMD EPYC 7642 48-Core	32 GB
Client 4	1× RTX 4090	AMD EPYC 7662 64-Core	48 GB

E Application Detail Page on Flower Hub

All benchmark applications used in this paper are publicly available on Flower Hub. The corresponding application identifiers are as follows:

- `@flwrlabs/fed-med-seg`[11]
- `@flwrlabs/fed-fin-fraud`[12]
- `@flwrlabs/fed-legal-llm`[13]
- `@flwrlabs/fed-phish-guard`[14]
- `@flwrlabs/fed-audio-tagging`[15]

Each application on Flower Hub is associated with a dedicated detail page that provides comprehensive information, including metadata, source code, documentation, and usage statistics (Figure 6). This design ensures transparency and accessibility, making all benchmark applications fully open source and facilitating community engagement, reuse, and extension. Such openness contributes to the robustness and sustainability of the Flower Hub ecosystem.

[11] https://flower.ai/apps/flwrlabs/fed-med-seg/
[12] https://flower.ai/apps/flwrlabs/fed-fin-fraud/
[13] https://flower.ai/apps/flwrlabs/fed-legal-llm/
[14] https://flower.ai/apps/flwrlabs/fed-phish-guard/
[15] https://flower.ai/apps/flwrlabs/fed-audio-tagging/

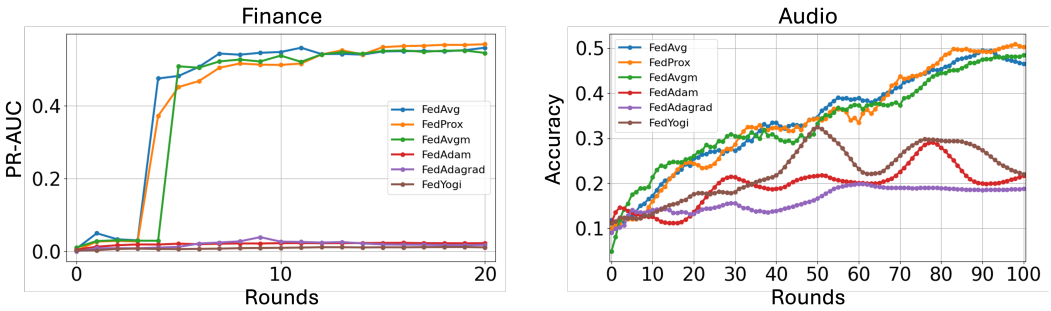


Figure 11: Changes in evaluation metrics over federated learning rounds for the financial fraud detection and audio tagging tasks. FedAdam, FedAdagrad, and FedYogi exhibit suboptimal or unstable performance on these tasks.

Table 7: Client-side memory footprint during deployment-mode FedAvg runs.

Benchmark	Mean CPU	Peak CPU	Mean GPU	Peak GPU
FedMedSeg	23.27 GiB	23.31 GiB	2.45 GiB	2.45 GiB
FedFinFraud	1.73 GiB	1.74 GiB	20.15 MiB	20.15 MiB

F Detailed Benchmark Settings

F.1 Client-level Label Distribution of FL Datasets

Figures 7 and 8 illustrate the class distributions across clients for the audio tagging, financial fraud detection, and phishing URL detection benchmarks. These results highlight the pronounced non-IID characteristics of the proposed federated datasets.

For the audio tagging dataset, no single client contains the full set of classes, increasing the difficulty of learning a globally consistent model. In the financial fraud detection dataset, the class distribution is extremely imbalanced, with all clients exhibiting fraud rates below 0.2%, reflecting realistic real-world conditions. Similarly, in the phishing URL detection dataset, the phishing rate varies substantially across clients, introducing additional heterogeneity. Together, these properties create challenging yet representative settings for evaluating federated learning algorithms.

F.2 Hyperparameter Selection

Table 5 summarizes the common hyperparameter configurations used across the five benchmark tasks. The complete set of hyperparameters for each task is available on the corresponding application detail pages (Section E).

These configurations are chosen based on commonly adopted settings in prior literature, and we do not perform extensive hyperparameter tuning. The reported results should therefore be interpreted as baseline performance, providing a reference for future optimization and exploration.

F.3 Deployment Configuration

In deployment mode, we configure a set of heterogeneous GPU instances to approximate realistic cross-region federated learning environments. Figure 10 illustrates the deployment topology, in which the federated server is hosted in Canada, while five clients are distributed across the United States, Poland, Spain, and Japan.

Table 6 summarizes the hardware configurations used in this setup. The server is deployed on an RTX 5090 instance, while client nodes run on a mix of RTX 5090 and RTX 4090 instances. This configuration reflects a practical cross-silo FL scenario with geographically distributed participants and heterogeneous compute resources. We use this setup to validate that all benchmarks can be executed in a real deployment environment without requiring any modifications to the application code.

G Additional Experimental Results

Figure 11 presents the evolution of evaluation metrics over federated learning rounds for the financial fraud detection and audio tagging tasks. FedAvg, FedProx, and FedAvgM exhibit similar convergence behavior, with FedProx achieving the best final performance on both tasks. In contrast, the FedOpt family (FedAdam, FedAdagrad, and FedYogi) performs poorly and exhibits instability. This can be attributed to two main factors: (1) both tasks involve highly non-IID client data distributions, which complicate optimization; and (2) FedOpt methods are sensitive to hyperparameter choices, requiring careful tuning to achieve stable and competitive performance.

For the deployment experiments, we additionally report both peak and average client-side memory usage in Table 7. CPU memory consumption is primarily driven by data preprocessing and loading, whereas GPU memory usage is dominated by model training. The financial fraud detection task, which employs a lightweight MLP model, exhibits significantly lower GPU memory usage compared to the other tasks.