FLUTE: A Scalable, Extensible Framework for High-Performance Federated Learning Simulations

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper we introduce "Federated Learning Utilities and Tools for Experimentation" (FLUTE), a high-performance open source platform for federated learning research and offline simulations. The goal of FLUTE is to enable rapid prototyping and simulation of new federated learning algorithms at scale, including novel optimization, privacy, and communications strategies. We describe the architecture of FLUTE, enabling arbitrary federated modeling schemes to be realized, we compare the platform with other state-of-the-art platforms, and we describe available features of FLUTE for experimentation in core areas of active research, such as optimization, privacy, and scalability. A comparison with other established platforms shows speed-ups up to $42\times$ and savings in memory footprint of $3\times$. A sample of the platform capabilities is presented in the Appendix for a range of tasks and other functionality such as scaling and a variety of federated optimizers.

1 Introduction

Distributed Training (DT) has drawn much scientific attention with focus on scaling the model training processes, either via model or data parallelism. As training datasets grow larger, the need for data parallelism has become a priority. Different approaches have been proposed over the years, as discussed in [1], aiming at more efficient training, either in the form of training platforms such as "Horovod" [2, 3] or algorithmic improvements like "Blockwise Model-Update Filtering" (BMUF) [4]. Most often, these techniques and platforms are evaluated on metrics such as data throughput (without compromising model performance), model and training dataset size, and GPU utilization and convergence rates. There are a few underlying assumptions implied for such DT scenarios, i.e., data and device uniformity and efficient network communication between the working nodes. Besides the communication/network specifications, data uniformity is paramount for successful training, ensured by repeated randomization and data shuffling steps.

On the other hand, new constraints in data management are emerging, driven by the need for privacy compliance of personal data and information [5], vastly distributed and segregated data silos, etc. As such, increasingly more data are inaccessible, either behind firewalls or on users' devices without the option of being shared for centralized training. The "*Federated Learning*" (FL) paradigm has been proposed as a strategy to address these constraints, as in [6]. Federated learning is a decentralized machine learning scheme with focus on collaborative training and user data privacy. The key idea is to enable training of a global model with the participation of multiple clients coordinated by a central server. Each client trains the model using local data and then sends the tuned parameters back to the server, where the global model is updated by aggregating the client, aka local, information.

One of the challenges when using the Federated Learning platforms is the need for scaling the learning process to millions of clients, in order to simulate real-world conditions under reasonable computing resources. As such, testing and validating any novel algorithm in realistic scenarios, e.g., using real

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

devices or close-to-real scaled deployments, can be particularly difficult. Simulation platforms play an important role, enabling researchers and developers to develop proof-of-concept implementations (POCs) and validate their performance before building and deploying in the wild. While several open-source frameworks have been developed to enable FL solutions, few offer end-to-end simulations, experiment orchestration, and scalability.

This paper introduces a novel platform "*Federated Learning Utilities and Tools for Experimentation*" (FLUTE) as a framework for running large-scale, offline FL simulations. It is designed to be flexible, to enable arbitrary model architectures[1], and to allow for prototyping novel approaches in federation, optimization, quantization, privacy, and so on. Finally, it provides an optional integration with AzureML workspaces, enabling scenarios closer to real-world applications, and leveraging platform features to manage and track experiments, parameter sweeps, and model snapshots.

The main contributions of FLUTE are: (i) A platform for high-performance FL simulations at scale (scaling to millions of clients), (ii) Flexibility in the platform to include new FL paradigms, unlocking research, experimentation, and POC development, (iii) A generic API for new model types and data formats, (iv) A pre-built list of features - state-of-the-art federation algorithms, optimizers, differential privacy, bandwidth management, client management/sampling, etc, (v) Experimental results illustrating the utility of the platform for FL research, (vi) A competitive analysis and comparisons with some of the leading FL simulation platforms[2], FedML, described in [7] and Flower, details in [8]. The goal of FLUTE is to facilitate the study of new algorithmic paradigms and optimizations, enabling more effective FL solutions in real-world deployments. FLUTE's unique architecture allows clients to be instantiated on-the-fly and then processed asynchronously, making it more efficient than other platforms. On the other hand, FLUTE does not currently address challenges like data collection, secure aggregation, device labelling, or attestation. The code for the platform is open-sourced and available at **https://github.com/AnonymousQTHM31/FLUTE**.

2 Background and Prior Work

In general, there are two different approaches concerning the architecture of FL systems: either using a central server [9], as the "coordinator or orchestrator", or opting for peer-to-peer learning, without the need of a central server [10]. FLUTE is based on the "server-client" architecture, where the server coordinates any number of clients. Besides the basic architecture, FLUTE addresses technical challenges such as the required resources, i.e. bandwidth, and computing power, efficiency, optimization and learning pipeline [11], and privacy constraints.

Such challenges can be attributed to either the ML side of federated learning, such as the distributed nature of the tasks, or the engineering side where the available resources are limited:

Communication overhead: FL relies heavily on the communication between server and the clients to complete any training iteration. The fact that some of the clients and the server can be in different networks may cause limited connectivity, high latency and others. Different approaches have been proposed, e.g., gradient quantization and sparsification [6, 12], different architectures per client [13], use of adapters for federating transformer models, etc. Most of these approaches are already implemented in FLUTE, e.g., quantization results shown in Appendix B.

Hardware heterogeneity: Computing capabilities of each client can vary, i.e., CPU, memory, battery level, storage are not expected to be the same across all nodes. This can affect both the selection and availability of the participating devices and it can bias the learning process. Different approaches have been also proposed to address clients that fall behind, i.e. "stragglers", the most popular of them allowing for asynchronous updates and client dropouts. FLUTE provides a flexible, asynchronous framework to incorporate workers of different capabilities. Also, there is an intuitive way of modeling faster/slower nodes as part of the training process.

Unbalanced and/or non-IID data: Local training data are individually generated according to the client usage, e.g., users spending more time on their devices tend to generate more training data than others. Therefore, it is expected that these locally segregated training sets may not be either

[1]The repository provides some examples and users are urged to add their implementations.

[2]Based on the availability of simulation functionality and the number of downloads from their github repo.

a representative sample of the global data distribution or uniformly distributed between clients. A simple strategy to overcome communication overheads was proposed with the "*Federated Averaging*" (FedAvg) algorithm [6]. In this approach, the clients perform several training iterations, and then send the updated models back to the server for aggregation based on a weighted average. FedAvg is one of the go-to training strategies for FL, given the simplicity and the consistently good results achieved in multiple experiments. On the other hand, FedAvg is not the best aggregation strategy, especially in the case of non-IID local data distributions. Over time, new approaches have emerged to overcome these limitations, for example, adaptive optimizers [14], SCAFFOLD [15], the DGA algorithm [16], which proposes an optimization strategy to address the heterogeneity problems on data and devices.

Threats: The ever increasing interest for applying FL in different scenarios has brought interest in malicious attacks and threat models, as described [17]. FL itself cannot assure either data privacy, or robustness to diverse attacks proven to be effective in breaking privacy or destroying the learning process. Without any mitigation, both the server and clients can be attacked by malicious users. For example, attackers can try to poison the model by sending back to the server fake model parameters [18] or fake the server and send a malicious model to the clients stealing the local information [19]. To tackle this issue, FL strategies started to incorporate techniques like Differential Privacy (DP), as detailed in [20] or Multi-Party Computation (MPC), which only reveals the computation result while maintaining the confidentiality of all the intermediate computations [21, 22]. Defenses to inference and backdoor attacks, based on DP and masking, are already part of FLUTE.

Simulation and prototyping: Building federated learning solutions can require significant up-front engineering investment, often with an unclear or uncertain outcome. Simulation frameworks enable FL researchers and engineers to estimate the potential utility of a particular solution, and investigate novel approaches, before making any significant investments. Recently, several frameworks have been proposed for FL simulations, including *TensorFlow Federated* [3], *FedML* [7] and *Flower* [8], with each having different focus and different simulation scope of their proof-of-concept scenarios. We extensively compare the main features of these frameworks alongside FLUTE in Section 4, showing the competitive advantage of FLUTE vs. the other platforms.

3 FLUTE Platform: Design and Features

The main goal of FLUTE is to be a scalable framework for rapid prototyping, encouraging researchers to propose novel FL solutions addressing real-world applications, in scale, data volume, etc, focusing on the following design constraints/specs:

- *Scalability*: Capacity to process many thousands of clients on any given round. FLUTE allows to run large-scale experiments using up to 10,000 clients with reasonable turn-around time, since scale is a critical factor in understanding practical metrics such as convergence and privacy-utility trade-offs.
- *Flexibility*: Allow for any combination of model, dataset, and optimizer. FLUTE Support for diverse FL configurations, including standardized implementations such as DGA and FedAvg, with Pytorch being the framework of choice for implementing the models.
- *Versatility*: Allow the end users to easily plug in customized/new techniques like differential privacy or gradient quantization. FLUTE provides an open architecture allowing users to incorporate new algorithms in a straightforward fashion.

FLUTE provides a range of built-in functionality while the implemented state-of-the-art algorithms cover important areas in FL. In more detail:

Federated Optimization: FLUTE already supports a range of federated optimizers by adjusting the gradient aggregation process, and the server-side optimizers, making FedAvg [6], FedYogi or FedAdam [23], and DGA [16, 24], straightforward to apply. Also, the FLUTE client scaling capabilities are enhanced by switching to large-batch optimizers on the server-side, like LAMB [25], and LARS [26], validated by the experimental evidence, shown in Appendix B.

Differential Privacy: In FLUTE, either local or global DP can be used, depending on whether the clients or the server are responsible for doing the clipping and noise-addition. In both cases, this step

is directly applied on the pseudo-gradients, i.e., the difference between current and previous weights after each user's data is processed. The pseudo-gradients are re-scaled so that their norm is at most C, ensuring the norm of the difference between any two of them (the sensitivity) is bounded. We typically use Gaussian noise, with variance $\sigma^2 = 2\log\left(\frac{1.25}{\delta}\right)\frac{C^2}{\epsilon^2}$ picked so that the aggregation is at most (ϵ, δ)-DP w.r.t. each client. In the case of DGA [16], the aggregation weights also go through the same procedure, since they are data-dependent.

Bandwidth Optimization: We have incorporated an approach similar to that of [27]. At each layer of the neural network, we first get the dynamic range of the gradient components, and then create a histogram of 2^B bins between these two values. Next, we replace each gradient component by the label of the closest bin. That way, we need just to communicate the bin indices, together with the min. and max. values. Besides gradient compression and quantization, We also in the process of incorporating FL-algorithms with different model architectures [13] and the use of adapters in NLU tasks, achieving even further bandwidth compression.

Computing Resources: AzureML (AML) [28] is the preferred computing FLUTE environment for staging, executing, tracking, and summarizing the FL experiments. FLUTE has a native AML integration included for job submissions, allowing the users to use the built-in CLI or web interface for job/experiment tracking, and for visualization purposes. While FLUTE needs only the experiment-related configurations, AML expects the computing environment parameters in a configuration file, such as target, cluster, code, etc. Besides AML, FLUTE can also run seamlessly on stand-alone devices, such as laptop and desktop machines (like those used for Section 4), using the local GPUs if/when available.

FLUTE design is based on a central server architecture, as depicted in Figure 1. The logical workflow is: (i) Send an initial global model to participating clients, (ii) Train instances of the global model with the locally available data on each client, (iii) Send training information, e.g., updated model parameters, logits (if required), and/or gradients/pseudo-gradients back to the server, (iv) Combine the returned information on the server to produce a new global model, (v) Optionally, update the global model with an additional server-side rehearsal step, (vi) Send the updated global model to the subset of clients, (vii) Repeat steps (ii) - (vi) after sampling a new subset of clients for the next training iteration.

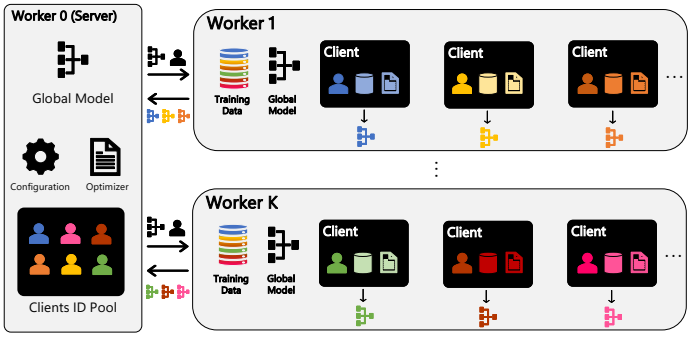

Figure 1: Client-server communication protocol. At each iteration, the server sends a copy of the global model to each worker, samples a number of clients and asynchronously assigns them to a worker as they become available

A FLUTE job leverages one or more multi-gpu VMs running up to a total of K worker processes, each executing tasks assigned by the server (Worker 0). The number of workers is decoupled from the number of clients P, allowing the platform to scale to millions of clients even when $K \ll P$. During training, the server plays the role of both the *orchestrator* and the *aggregator*. First, it distributes a copy of the global model(s), training data and the list of all the client-IDs among the workers. The workers, on their turn, process the clients' data to produce new models, and send the models back to the server. After a number of clients is processed, the server will aggregate all the resulting models, typically into a single global model. Algorithm 1 describes in detail this process and Figure 2 exemplifies the execution flow.

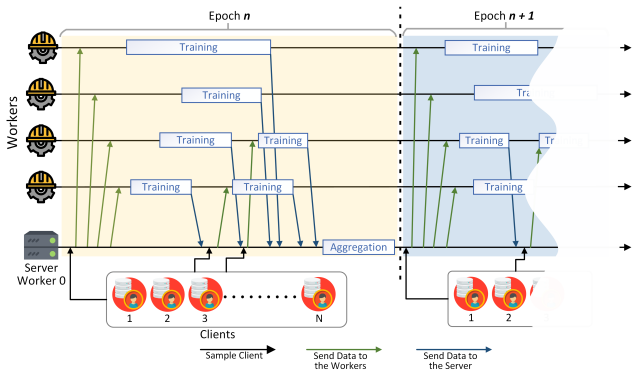

Figure 2: FLUTE Execution Flow: The server samples $\mathcal{N}$ of the clients and sends them to the K workers. Every time one of the workers finishes processing the client data, it returns the gradient and draws the next client until all clients are processed.

The distributed nature of FLUTE is based on PyTorch, using torch.distributed package as the communication backbone. The interface between server and workers is based on "*messages*", that contain model parameters, client-IDs and training instructions. There are four message-types that can be passed from server to worker: 1. `Update` creates a copy of the model parameters and learning rate on the worker, 2. `Train` triggers the execution of a training step on a worker, for a given client. The resulting model (or pseudo-gradient) is passed from worker to server, 3. `Evaluate` triggers the execution of an evaluation step in a given client. Resulting metrics are passed from worker to server, 4. `Terminate` shuts down the worker thread.

Algorithm 1 FLUTE Orchestration: $\mathcal{P}$ is a Client Pool, which contains IDs of each client, $P = |\mathcal{P}|$, M is the federation rounds to be executed, K is the number of Workers, $\mathcal{N}$ is the subset of clients per iteration and $N = |\mathcal{N}|$ the number of clients per iteration

Server-Side Worker-0:
for each federated round from $1, \ldots, M$ **do**
  $\mathcal{N} \leftarrow$ Sample N clients from $\mathcal{P}$
  repeat
    Wait for $worker_k$ to finish
    Save pseudo-gradient response from $worker_k$
    $c \leftarrow$ Sample client-ID from $\mathcal{C}$
    Dispatch model and c to $worker_k$
  until all client-IDs c in $\mathcal{N}$ have been processed
  Aggregate pseudo-gradients
  Update model with optimization step
end for

Client-Side Worker-k, with k > 0:
Load client and model data
Execute local training processes
Send pseudo-gradients and statistics about local training to Worker-0

FLUTE leverages the communication scheme in Figure 1 by loading a local copy of the training data on each worker prior to training, significantly reducing the traffic communication between server and workers, only sending client indices. In this research simulator, all clients are implemented as isolated object instances, therefore local data on each client stays within the local storage boundaries and it is never sent to the server or aggregated with other local data sources. Only metrics, model parameters or gradients are communicated between the server and clients – these quantities can be encrypted [3] if necessary.

4 Comparison with related platforms

FLUTE allows customized training procedures and complex algorithmic implementations in scale, making it a valuable tool to rapidly validate the feasibility of novel FL solutions, while avoiding the need to deal with complications that production environments present. To this day, different FL platforms have been proposed, however, most of them have been designed with a specific purpose limiting their flexibility to experiment with complex large-scale FL scenarios in a reasonable amount of time using limited resources. Table 1 shows a detailed comparison of the most common FL platforms features and their main focus.

[3] Encryption and secure aggregation are not currently implemented in FLUTE - these are security mechanisms which aren't strictly necessary for simulations.

	FLUTE	FedML (Parrot)	Flower (Simulator)
Focus	Research and Simulation	Research and Production	Research and Production
ML Framework	PyTorch	PyTorch	PyTorch / TensorFlow
Communication Protocols Supported	Gloo,NCCL	SP, MPI	Ray, gRPC
Support Security/Privacy related functions	✓	✓	✓
Support Multiple Federated Optimization Techniques	✓	✓	✓
Flexible and Generic API	✓	✓	✓
Cloud Integration	✓	✓	✗
Multi-GPU Support	✓	✓	✗
Performance Optimizations	✓	✗	✗
Easily Extensible to Production	✗	✓	✓

Table 1: Comparison between FLUTE and Popular Federated Learning Simulation Platforms. This analysis is focused on the simulators provided by these platforms only.

Some production-oriented frameworks also allow researchers to work in a simulation environment using the same platform. However, these frameworks suffer from lack of flexibility and limited functionality since their architecture is optimized towards productization, especially in complex FL scenarios. We selected the *FedML* and *Flower* platforms as the most representative, based on their number of stars on GitHub.

FLUTE architecture provides a significant advantage in runtime and memory utilization, leveraging the benefits when using NCCL in GPUs, the communication optimization, etc, outperforming all other platforms by $42\times$ in speed and $3\times$ in memory consumption, Tables 2 and 4.

Task				FedML (MPI) 0.7.303			FLUTE (NCCL) 1.0.0		
Model	Dataset	Clients	Rounds	Acc	Time	GPU memory	Acc	Time	GPU memory
lr	mnist	1000	100	81	00:03:09	3060 MB	81	**00:01:35**	**1060 MB**
cnn	fedmnist	3400	800	83	05:49:52	5180 MB	83	**00:08:22**	**1770 MB**
resnet18	fedcifar100	500	4000	34	15:55:36	5530 MB	33	**01:42:01**	**1900 MB**
rnn	fedshakespeare	715	1200	57	06:46:21	3690 MB	57	**00:21:50**	**1270 MB**

Table 2: GPU Performance comparison FLUTE 1.0.0 vs FedML 0.7.303 on 4x NVIDIA RTX A6000 using FedML Datasets. Test accuracy is reported from the last communication round. The training configuration and datasets details for these experiments are reported in Appendix B.

The advantage of FLUTE relies on its ability to asynchronously assign new clients to the workers, as they become available, and receive their outputs. On the other hand, FedML[4] links the number of workers with the number of MPI processes, which is reflected as the number of parallel clients during training. FLUTE design allows processing multiple clients per worker decoupling the need for $1:1$ mapping between clients and training processes. Another FLUTE strength is that each worker holds a preloaded local copy of the training data, avoiding communication overheads during training as the Server only sends indices of clients to instantiate. An additional comparison of FLUTE with the Gloo backend vs Flower 1.0.0 is presented in Appendix A.

5 Discussion and Conclusions

In recent years, researchers have made significant efforts to address the challenges FL raises, especially when it comes to setting up FL-friendly environments – privacy guarantees, time-consuming processes, communication costs and beyond. Herein, we presented FLUTE, a versatile, open-architecture platform for high-performance federated learning simulation that is available as open source. FLUTE provides scaling capabilities, several state-of-the-art federation approaches and related features such as differential privacy, and a flexible API enabling extensions and the introduction of novel approaches. FLUTE is model and task independent, and provides facilities for easy integration of new model architectures based on PyTorch.

The goal of FLUTE is to enable rapid experimentation and prototyping, and facilitate the development of new FL research efforts. We encourage the research community to explore new research using FLUTE and invite contributions to the public source repository.

[4]FedML Simulator (Parrot) on its release 0.7.303, commit ID 8f7f261f (`https://github.com/FedML-AI/FedML/tree/8f7f261f44e58d0cb5a416b0d6fa270b42a91049`)

References

[1] T. Ben-Nun and T. Hoefler. Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis. *ACM Computing Surveys*, 4(65), 2019.

[2] A. Sergeev and M. D. Bals. Horovod: Fast and Easy Distributed Deep Learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A System for Large-Scale Machine Learning. *arXiv preprint arXiv:1605.08695*, 2016.

[4] Kai Chen and Qiang Huo. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Proc. ICASSP*, March 2016.

[5] B. Wolford. A Guide to GDPR Data Privacy Requirements. `https://gdpr.eu/data-privacy/`, 2021.

[6] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B.A.y. Arcas. Communication-efficient Learning of Deep Networks from Decentralized Data. In *Proc. International Conference on Artificial Intelligence and Statistics*, pages 1273—1282, 2017.

[7] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning, 2020.

[8] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2020.

[9] P. Patarasuk and X. Yuan. Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, 2009.

[10] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee. Asynchronous Decentralized Learning of a Neural Network. *arXiv preprint arXiv:2004.05082v1*, 2020.

[11] O. Shamir, N. Srebro, and T. Zhang. Communication Efficient Distributed Optimization Using an Approximate Newton-type Method. *arXiv preprint arXiv:1312.7853*, 2013.

[12] Divyansh Jhunjhunwala, Advait Gadhikar, Gauri Joshi, and Yonina C. Eldar. Adaptive quantization of model updates for communication-efficient federated learning, 2021.

[13] Yae Jee Cho, Andre Manoel, Gauri Joshi, Robert Sim, and Dimitrios Dimitriadis. Heterogeneous ensemble knowledge transfer for training large models in federated learning. In *Proc. of the 31st Intern. Joint Conf. on Artificial Intelligence, IJCAI-22*, pages 2881–2887. Intern. Joint Conf. on Artificial Intelligence Org., Juky 2022.

[14] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *9th Intern. Conf. on Learning Representations, ICLR-21*, May 2021.

[15] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proc. of the 37th Inter. Conf. on Machine Learning*, Proc. of Machine Learning Research, pages 5132–5143. PMLR, July 2020.

[16] D. Dimitriadis, K. Kumatani, R. Gmyr, Y. Gaur, and E. S. Eskimez. Federated Transfer Learning with Dynamic Gradient Aggregation. *arXiv preprint arXiv:2008.02452*, 2020.

[17] Pengrui Liu, Xiangrui Xu, and Wei Wang. Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives. *Cybersecurity*, 5, Feb. 2022.

[18] Jiale Zhang, Bing Chen, Xiang Cheng, Huynh Thi Thanh Binh, and Shui Yu. PoisonGAN: Generative Poisoning Attacks Against Federated Learning in Edge Computing Systems. *IEEE Internet of Things Journal*, 8(5):3310–3322, 2021.

[19] David Enthoven and Zaid Al-Ars. An overview of federated deep learning privacy attacks and defensive strategies. *arXiv preprint arXiv:2004.04676*, 2020.

[20] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020. doi: 10.1109/TIFS.2020.2988575.

[21] David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications, 2020.

[22] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning, 2019.

[23] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konecny, S. Kumar, and McMahan. H. B. Adaptive Federated Optimization. *arXiv preprint arXiv:2003.00295v1*, 2020.

[24] Dimitrios Dimitriadis, Ken'ichi Kumatani, Robert Gmyr, Yashesh Gaur, and Sefik Emre Eskimez. Dynamic Gradient Aggregation for Federated Domain Adaptation. *arXiv preprint arXiv:2106.07578*, 2021.

[25] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes, 2020.

[26] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks, 2017.

[27] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30:1709–1720, 2017.

[28] AzureML Team. AzureML: Anatomy of a Machine Learning service. In *Proc. of The 2nd Intern. Conf. on Predictive APIs and Apps*, Proc. of Machine Learning Research, pages 1–13. PMLR, Aug. 2016.

[29] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. LibriSpeech: an ASR corpus based on public domain audio books. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, 2015.

[30] Dimitrios Dimitriadis, Kenichi Kumatani, Robert Gmyr, Yashesh Gaur, and Sefik Emre Eskimez. A federated approach in training acoustic models. In *In Proceedings of Interspeech'20*, 2020.

[31] Yann LeCun and Corinna Cortes. MNIST handwritten digit database, 2010. URL `http://yann.lecun.com/exdb/mnist/`.

[32] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[33] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. The pushshift reddit dataset. In *Proc. of the Intern. AAAI conference on web and social media*, volume 14, pages 830–839, 2020.

[34] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *Stanford Tech. Report*, 2009.

[35] Qiming Diao, Minghui Qiu, Chao-Yuan Wu, Alexander J. Smola, Jing Jiang, and Chong Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[36] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proc.of ACL International Conference EMNLP'15*, 2015.

[37] Xiaotao Gu, Yuning Mao, Jiawei Han, Jialu Liu, You Wu, Cong Yu, Daniel Finnie, Hongkun Yu, Jiaqi Zhai, and Nicholas Zukoski. Generating representative headlines for news stories. In *Proc. of WWW Conf'20*, 2020.

[38] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[39] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65, 1958.

[40] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [No]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] A link to code repository and all the relevant experiments is added in the main body of the paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Only a couple of experiments have such error bars. However, such format was not deemed necessary for the rest
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] All new assets can be found in the code repository
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] All datasets used have been open-sourced
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Comparison with different platforms

To enable the comparisons, the training configurations for the experiments presented in Table 2 are based on the FedML Benchmarking Recipes [5] using the setup and models as detailed in the FedML repository[6]. The datasets are also provided by FedML[7]. The FLUTE scripts can be found at the *Experiments* folder in the repository. More details about the specific setups are detailed in Table 3.

Model	Dataset	Algorithm	# Clients	Clients/round	Batch Size	Client Optim.	lr	Local Epochs	# Rounds	Test Freq
Log. Regr.	mnist	FedAvg	1000	10	10	SGD	0.03	1	100	20
CNN	fedmnist	FedAvg	3400	10	20	SGD	0.1	1	800	50
ResNet18	fedcifar100	FedAvg	500	10	20	SGD	0.1	1	4000	50
RNN	fedshakespeare	FedAvg	715	10	4	SGD	0.8	1	1200	50

Table 3: Training configuration for FedML Benchmarking

The Flower platform seems more efficient when multiple CPUs are employed – the platform is fairly inefficient by design when multiple GPUs are used during simulation. To run a fair comparison, we compare the FLUTE CPU performance (using the Gloo backend) against Flower. We evaluate the overall time of the job in FLUTE 1.0.0 vs Flower 1.0.0 [8], as in Table 4 using the same setup for the *"lr mnist"* task described in Table 3. FLUTE results outperform up to $54\times$ faster than Flower on GPUs given that their simulation capabilities are not optimized for multi-GPU jobs. Also, for the CPU task FLUTE shows a competitive advantage running Gloo backend, over Flower being $9\times$ faster.

	FLUTE 1.0.0 Gloo/NCCL		Flower 1.0.0 gRPC	
Accelerator	Acc	Time	Acc	Time
CPU	80	**00:03:20**	80	00:30:14
GPU 2x	80	**00:01:31**	80	01:21:44
GPU 4x	81	**00:01:26**	79	00:56:45

Table 4: Performance comparison FLUTE 1.0.0 vs Flower 1.0.0 on 4x NVIDIA RTX A6000, AMD EPYC 7V12 64-Core Processor. Test accuracy is reported from the last communication round.

B Case Studies

This section provides insights by exploring a variety of features of the FLUTE platform. The list of presented datasets, tasks and experimental results is by no means exhaustive. In this context, we do not present any of the models used since the platform allows training on any architecture currently supported by PyTorch.

B.1 Sample of Baseline Experiments and Datasets

We provide some of the available models/tasks as part of the FLUTE distribution. This list of models and tasks is **not** exhaustive since the flexibility of the platform allows extensions in models such as GNN's, GBT's and others:

- **ASR Task: LibriSpeech**: FLUTE offers a Speech Recognition template task based on the LibriSpeech task [29]. The dataset contains about 1,000 hours of speech from 2,500 speakers reading books. Each of the speakers is labeled as a different client. In one of the

[5] FedML Benchmarking Results `https://doc.fedml.ai/simulation/benchmark/BENCHMARK_MPI.html`

[6] FedML Benchmarking Examples `https://github.com/FedML-AI/FedML/tree/master/python/examples/simulation/mpi_fedavg_datasets_and_models_example`

[7] FedML Datasets `https://github.com/FedML-AI/FedML/tree/master/python/fedml/data`

[8] Flower Simulator on its release 1.0.0, commit ID 4e7fad9 - https://github.com/adap/flower/tree/4e7fad99389a5ee511730841b61f279e3359cb16

ASR task examples, a sequence-to-sequence model was used for training, more details can be found in [30].

- **Computer Vision Task: MNIST and EMNIST**: Two different datasets, i.e., the MNIST [31] and the EMNIST [32] dataset are used for Computer Vision tasks. The EMNIST dataset is a set of handwritten characters and digits captured and converted to 28×28 pixel images maintaining the image format and data-structure and directly matching the MNIST dataset. Among the many splits of EMNIST dataset, we use the "EMNIST Balanced", containing $\tilde{1}$32k images with 47 balanced classes.
- **NLP Tasks: Reddit**: Different NLP tasks are supported in FLUTE with 2 use-cases for MLM and next-word prediction using Reddit data [33]. The Reddit dataset consists of users' tweets grouped in months when published on the social network. For the use-cases, we use 2 months of Reddit data with 2.2M users.The initial/seed models used are based either on HuggingFace or a baseline LM model, as described below.
- **Sentiment Analysis: sent140, IMDb, YELP**: Sent140 [34], is a sentiment analysis dataset consisting of tweets, automatically annotated from the emojis found in them. The dataset consists of 255k users, with mean length of 3.5 samples per user. IMDb is based on movie reviews of 1012 users providing 140k reviews with 10 rating classes [35]. The YELP dataset is based on restaurant review from Yelp and the sentiment label is from 1 to 5 [36]. It contains 2.5k users with 425k reviews.
- **Baseline LM Model**: A baseline LM model is used for most of the experiments in Section B. A two-layer GRU with 512 hidden units, 10,000 word vocabulary, and embedding dimension 160 is used for fine-tuning during the FL experiments. The seed model is pretrained on the Google News corpus [37].

B.2 DP Experiments

In this experiment, we explore privacy-utility trade-offs in LM training, using the baseline model described in Section B.1. Local differential privacy was applied with ϵ parameter LDP ϵ. We track the per-client noise across all clients and yield a final global RDP ϵ after 500 rounds of training. We express the privacy-utility tradeoff as the ratio of accuracy and RDP ϵ. Clearly, for a fixed number of training rounds, we achieve better privacy and better accuracy by sampling a larger number of clients. We observe a penalty of about 4.3% relative to the non-private baseline ($\epsilon = \infty$).

Clients/Iter.	LDP ϵ	RDP ϵ	Acc @1 (%)	Tradeoff
10,000	100	0.108	17.20	0.00172
1,000	100	0.333	14.80	0.00148
10,000	500	1.41	20.00	0.000399
1,000	500	12.6	18.00	0.000361
10,000	750	3.65	20.30	0.00027
1,000	750	38.3	18.80	0.000251
10,000	1,000	7.66	20.30	0.000203
1,000	1,000	79.7	19.40	0.000194
10,000	inf	inf	21.70	0.0
1,000	inf	inf	21.40	0.0

Table 5: Next-word prediction: Results comparing language model accuracy and privacy for various client sampling and LDP ϵ parameters. All experiments trained for 500 rounds. Trade-off is defined as the ratio of accuracy and RDP ϵ

B.3 Quantization Experiment

In Table B.3, the accuracy for a next-word-prediction task is shown, on the Reddit dataset and the baseline LM model described in Section B.1, while using different values of quantization B. As expected, using less bits, while training for the same number of epochs, leads to decreased performance in terms of accuracy.

We have also done experiments varying the sparsity level, while keeping the quantization constant at 8 bits, cf. Table B.3. In this particular experiment, we had gains in bandwidth of up to 16x with no

	Quant. (bits)	Acc @1 (%)	Rel. Improv. (%)
Seed Model	N/A	9.83	(56.62)
SST	N/A	22.30	(1.59)
FL Training	32	22.70	0
	10	22.40	(1.32)
	8	22.20	(2.25)
	4	21.30	(5.87)
	3	18.80	(17.21)
	2	17.80	(21.58)

Table 6: Next-word prediction task: Top-1 accuracy after gradient quantization. The number of bits per gradient coefficient varies $2 - 32$.

significant change in performance. Error compensation techniques [38] could be attempted in order to increase the performance at higher sparsity levels. The difference in performances for the case of 8-bit quantization level in Tables B.3 and B.3 is due to noise during the training process.

% Sparsity	Gain in Bandwidth	Acc @1 (%)
0.0	4x	22.60
75.0	16x	21.70
95.0	80x	19.00
99.0	400x	17.70

Table 7: Performance obtained by varying sparsity level on gradients while keeping quantization fixed at 8 bits – gains in bandwidth are relative to standard 32 bits gradient. The performance reported is the best one over 5000 iterations, with 1000 clients being processed at each iteration.

B.4 Performance for Variable/Different Number of Clients

The number of clients processed at each round is a variable we can control on FLUTE. Here, we show how long a round typically takes using a varying number of clients, on a simulation with 1 server + 3 workers attached to RTX A6000 GPUs and 2.45GHz AMD EPYC cores. Notice that, since clients are processed sequentially by each worker, runtime scales linearly; FLUTE provides options to speed-up this process, such as processing clients in multiple threads and pre-encoding the data.

Number of Clients	Runtime (sec.)
1,000	22.1 ± 0.6
5,000	111.3 ± 2.4
10,000	219.0 ± 2.3
50,000	1103.7 ± 11.3

Table 8: How long it takes for 3 workers to process different number of clients, on a simple NLG experiment using a GRU model and the Reddit dataset. Averages are computed over 20 iterations.

Table 8 shows that FLUTE scales gracefully the number of clients per iteration, without any upper bound to that number. We can also look at the predictive performance attained for different numbers of clients, and study how it changes as a function of the optimizer used.

In Table 9 , we compare 4 different scenarios for optimizers, increasing the number of clients, showing that the accuracy remains stable for most of them. However, the Adam optimizer decreases its accuracy as the number of clients increase, compared to SGD-LAMB that reaches a better performance with a larger number of clients.

B.5 Comparing Optimizers

This experiment of next-word prediction, using the Reddit dataset and baseline LM model described in Section B.1, explores model training performance for a variety of state-of-the-art optimizer choices.

	Optimizer	Acc @ 1 (%)
1k clients/iter	Seed Model	9.80
	Adam (Baseline)	22.70
	RL-based DGA	22.80
10k clients/iter	Adam (Baseline)	20.80
	SGD-LARS	17.00
	Adam-LARS	21.40
	SGD-LAMB	23.00
Variable number	Adam	22.30

Table 9: Next-word Prediction task: Top-1 accuracy achieved varying number of clients and optimizers.

We trained a recurrent language model, fixing the number of clients per round to 1,000, and varying the choice of optimizer in the central aggregator. Specifically, we applied standard SGD [39], ADAM [40], LAMB [25], and LARS [26]. Table 10 illustrates the performance of each optimizer, including maximum validation accuracy, and convergence rate: the number of rounds to reach 95% of the max. accuracy. Note there is no hyper-parameter tuning of the optimizers for this experiment.

Optimizer	Acc @1 (%)	Convergence Round
LAMB	23.10	115
ADAM	22.70	641
SGD	20.60	2172
LARS	17.40	414

Table 10: Next-word prediction task: Top-1 Accuracy and training rounds to 95% convergence for various central optimizer choices.