NVIDIA FLARE: Federated Learning from Simulation to Real-World

Holger R. Roth Yan Cheng Yuhong Wen Isaac Yang Ziyue Xu Yuan-Ting Hsieh Kristopher Kersten Ahmed Harouni Can Zhao Kevin Lu Zhihong Zhang Wenqi Li Andriy Myronenko Dong Yang Sean Yang Nicola Rieke Abood Quraini Chester Chen Daguang Xu Nic Ma Prerna Dogra Mona Flores Andrew Feng

NVIDIA Corporation*
Shanghai, China
Munich, Germany
Bethesda, Santa Clara, USA

Abstract

Federated learning (FL) enables the building of robust and generalizable AI models by leveraging diverse datasets from multiple collaborators without centralizing the data. We created NVIDIA FLARE[1] as an open-source software development kit (SDK) to make it easier for data scientists to use FL in their research and real-world applications. The SDK includes solutions for state-of-the-art FL algorithms and federated machine learning approaches, which facilitate building workflows for distributed learning across enterprises and enable platform developers to create a secure, privacy-preserving offering for multiparty collaboration utilizing homomorphic encryption or differential privacy. The SDK is a lightweight, flexible, and scalable Python package, and allows researchers to bring their data science workflows implemented in any training libraries (PyTorch, TensorFlow, XGBoost, or even NumPy) and apply them in real-world FL settings. This paper introduces the key design principles of FLARE and illustrates some use cases (e.g., COVID analysis) with customizable FL workflows that implement different privacy-preserving algorithms.

1 Introduction

Federated learning (FL) has become a reality for many real-world applications [22]. It enables multinational collaborations on a global scale to build more robust and generalizable machine learning and AI models. In this paper, we introduce NVIDIA FLARE, an open-source software development kit (SDK) that makes it easier for data scientists to collaborate to develop more generalizable and robust AI models by sharing model weights rather than private data. While FL is attractive in many industries, it is particularly beneficial for healthcare applications where patient data needs to be protected. For example, FL has been used for predicting clinical outcomes in patients with COVID-19 [6] or to segment brain lesions in magnetic resonance imaging [26, 25]. FLARE is not limited to applications in healthcare and is designed to allow cross-silo FL [11] across enterprises for different industries and researchers.

In recent years, several efforts (both open-source and commercial) have been made to bring FL technology into the healthcare sector and other industries, like TensorFlow Federated [1], PySyft [32], FedML [10], FATE [16], Flower [2], OpenFL [21], Fed-BioMed [27], IBM Federated Learning [17], HP Swarm Learning [29], FederatedScope [30], FLUTE [7], and more. Some focus on simulated FL settings for researchers, while others prioritize production settings. FLARE aims to be useful for both sce-

*Contact: {hroth,yanc,chesterc,daguangx,pdogra,andyf}@nvidia.com

[1]Code is available at https://github.com/NVIDIA/NVFlare.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

narios: 1) for researchers by providing efficient and extensible simulation tools and 2) by providing an easy path to transfer research into real-world production settings, supporting high availability and server failover, and by providing additional productivity tools such as multi-tasking and admin commands.

2 NVIDIA FLARE Overview

FLARE stands for "Federated Learning Application Runtime Environment". The SDK enables researchers and data scientists to adapt their existing machine learning and deep learning workflows to a federated paradigm and enables platform developers to build a secure, privacy-preserving offering for distributed multiparty collaboration.

FLARE is a lightweight, flexible, and scalable federated learning framework implemented in Python that is agnostic to the underlying training library. Developers can bring their own data science workflows implemented in PyTorch, TensorFlow, or even in pure NumPy, and apply them in a federated setting. A typical FL workflow such as the popular federated averaging (FedAvg) algorithm [18], can be implemented in FLARE using the following main steps. Starting from an initial global model, each FL client trains the model on their local data for a certain amount of time and sends model updates to the server for aggregation. The server then uses the aggregated updates to update the global model for the next round of training. This process is iterated many times until the model converges.

Though used heavily for federated deep learning, FLARE is a generic approach for supporting collaborative computing across multiple clients. FLARE provides the *Controller* programming API for researchers to create workflows for coordinating clients for the purpose of collaboration. FedAvg is one such workflow. Another example is cyclic weight transfer [4]. The central concept of collaboration is the notion of "task". An FL controller assigns tasks (e.g., deep-learning training with model weights) to one or more FL clients and processes results returned from clients (e.g., model weight updates). The controller may assign additional tasks to clients based on the processed results and other factors (e.g., a pre-configured number of training rounds). This task-based interaction continues until the objectives of the study are achieved.

The API supports typical controller-client interaction patterns like broadcasting a task to multiple clients, sending a task to one or more specified clients, or relaying a task to multiple clients sequentially. Each interaction pattern comes with two flavors: wait (block until results from clients are received) or no-wait. A workflow developer can use any of these interaction patterns to create innovative workflows. For example, the *ScatterAndGather* controller (typically used for FedAvg-like algorithms) is implemented with the *broadcast_and_wait* pattern, and the *CyclicController* is implemented with the *relay_and_wait pattern*. The controller API allows the researcher to focus on the control logic without needing to deal with underlying communication issues. Figure 1 shows the principle. Each FL client acts as a worker that simply executes tasks assigned to it (e.g., model training) and returns execution results to the controller. At each task interaction, there can be optional filters that process the task data or results before passing it to the *Controller* (on the server side) or task executor (client side). The filter mechanism can be used for data privacy protection (e.g., homomorphic encryption/decryption or differential privacy) without having to alter the training algorithms.

Key Components NVIDIA FLARE is built on a componentized architecture that gives the flexibility to take FL workloads from research and simulation to real-world production deployment. Some of the key components of this SDK include:

- **FL Simulator** for rapid development and prototyping.
- **FLARE Dashboard** for simplified project management, secure provisioning, and deployment, orchestration.
- **Reference FL algorithms** (e.g., FedAvg, FedProx, SCAFFOLD) and workflows (e.g., Scatter and Gather, Cyclic).
- **Privacy preservation** with differential privacy, homomorphic encryption, and more.
- **Specification-based API** for extensibility, allowing customization with plug-able components.
- **Tight integration** with other learning frameworks like MONAI [3], XGBoost [5], and more.

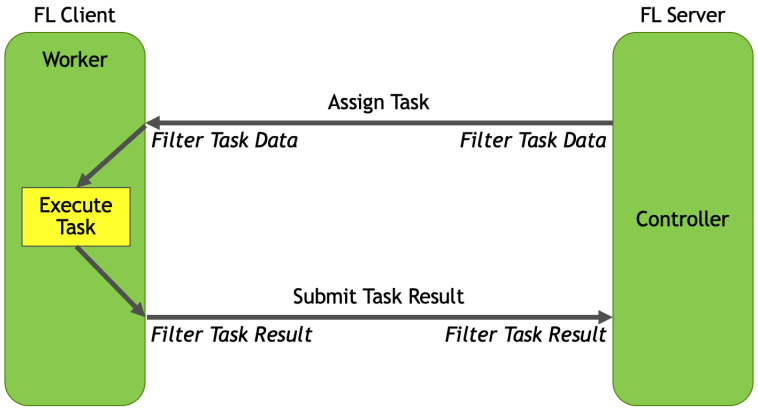


Figure 1: FLARE job execution. The *Controller* is a Python object that controls or coordinates the *Workers* to get a job done. The controller is run on the FL server. A *Worker* is capable of performing tasks. *Workers* run on FL clients.

High-Level Architecture FLARE is designed with the idea that less is more, using a specification-based design principle to focus on what is essential. This allows other people to be able to do what they want to do in real-world applications by following clear API definitions. FL is an open-ended space. The API-based design allows others to bring their implementations and solutions for various components. Controllers, task executors, and filters are just examples of such extensible components. FLARE provides an end-to-end operation environment for different personas. It provides a comprehensive provisioning system that creates security credentials for secure communications to enable the easy and secure deployment of FL applications in the real world. It also provides an FL Simulator for running proof-of-concept studies locally. In production mode, the researcher conducts an FL study by submitting jobs using admin commands either on Notebook or FLARE Console – an interactive command tool. FLARE provides many commands for system operation and job management. With these commands, one can start and stop a specific client or the entire system, submit new jobs, check the status of jobs, create a job by cloning from an existing one, and much more.

With FLARE's component-based design, a job is just a configuration of components needed for the study. For the control logic, the job specifies the controller component to be used and any components required by the controller.

3 System Concepts

A FLARE system is a typical client-server communication system that comprises one or more FL server(s), one or more FL client(s), and one or more admin clients. The FL Servers open two ports for communication with FL clients and admin clients. FL clients and admin clients connect to the opened ports. FL clients and admin clients do not open any ports and do not directly communicate with each other. The following is an overview of the key concepts and objects available in FLARE and the information that can be passed between them.

Workers and Controller FLARE's collaborative computing is achieved through the *Controller*/*Worker* interactions.

Shareable Object that represents a communication between server and client. Technically, the *Shareable* is implemented as a Python dictionary that could contain different information, e.g., model weights.

Data Exchange Object (DXO) Standardizes the data passed between the communicating parties. One can think of the *Shareable* as the envelope and the *DXO* as the letter. Together, they comprise a message to be shared between communicating parties.

FLComponent The base class of all the FL components. Executors, controllers, filters, aggregators, and their subtypes are all *FLComponents*. *FLComponent* comes with some useful built-in methods for logging, event handling, auditing, and error handling.

Executors Type of *FLComponent* for FL clients that has an execute method that produces a *Shareable* from an input *Shareable*. FLARE provides both single- and multi-process executors to implement different computing workloads.

FLContext One of the most important features of FLARE is to pass data between the FL components. *FLContext* is available to every method of all common FLComponent types. Through *FLContext*, the component developer can get services provided by the underlying infrastructure and share data with other components of the FL system.

Filters Filters in FLARE are a type of *FLComponent* that have a process method to transform the *Shareable* object between the communicating parties. A Filter can be used to provide additional processing to shareable data before sending or after receiving from a peer. Filters can convert data formats and a lot more and are FLARE's primary mechanism for data privacy protection [15, 9]:

- *ExcludeVars* to exclude variables from shareable.
- *PercentilePrivacy* for truncation of weights by percentile.
- *SVTPrivacy* for differential privacy through sparse vector techniques.
- Homomorphic encryption filters used for secure aggregation.

As an example, we show the average encryption, decryption, and upload times. We compare raw data to encrypted model gradients uploaded in Fig. 2 and Table 1 when hosting the server on AWS[2] and connecting 30 clients. One can see the longer upload times due to the larger message sizes needed by homomorphic encryption.

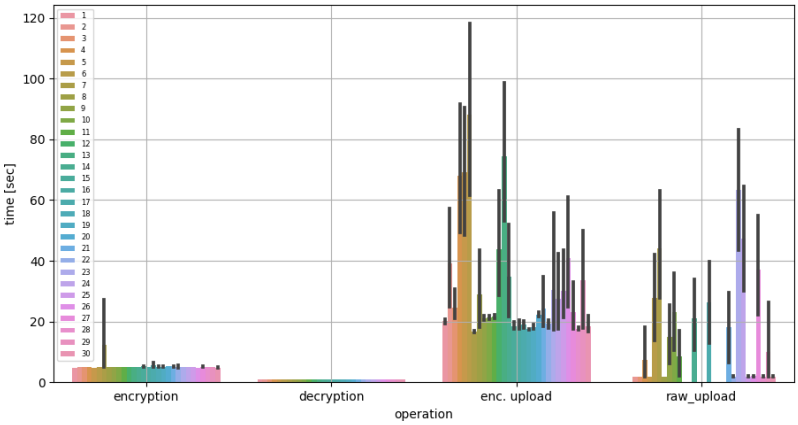


Figure 2: Running federated learning with 30 clients and the server on AWS.

Table 1: Federated learning exchanging homomorphic encrypted vs. g raw model updates.

Time in seconds	Mean	Std. Dev.
Encryption	5.01	1.18
Decryption	0.95	0.04
Enc. upload	38.00	71.17
Raw upload	21.57	74.23

[2]For reference, we used an m5a.2xlarge instance with eight vCPUs, 32-GB memory, and up to 2,880 Gbps network bandwidth.

Event Mechanism FLARE comes with a powerful event mechanism that allows dynamic notifications to be sent to all event handlers. This mechanism enables data-based communication among decoupled components: one component fires an event when a certain condition occurs, and other components can listen to that event and processes the event data. Each *FLComponent* is automatically an event handler. To listen to and process an event, one can simply implement the *handle_event()* method and process desired event types. Events represent some important moments during the execution of the system logic. For example, before and after aggregation or when important data becomes available, e.g., a new "best" model was selected.

3.1 Productivity Features

FLARE contains features that enable efficient, collaborative, and robust computing workflows.

Multi-tasking For systems with a large capacity, computing resources could be idle most of the time. FLARE implements a resource-based multi-tasking solution, where multiple jobs can be run concurrently when overall system resources are available. Multi-tasking is made possible by a job scheduler on the server side that constantly tries to schedule a new job. For each job to be scheduled, the scheduler asks each client whether they can satisfy the required resources of the job (e.g., number of GPU devices) by querying the client's resource manager. If all clients can meet the requirement, the job will be scheduled and deployed to the clients.

High Availability and Server Failover To avoid the FL server as a single point of failure, a solution has been implemented to support multiple FL servers with automatic cut-over when the currently active server becomes unavailable. Therefore, a component called *Overseer* is added to facilitate automatic cut-over. The *Overseer* provides the authoritative endpoint info of the active FL server. All other system entities (FL servers, FL clients, admin clients) constantly communicate (i.e., every 5 seconds) with the Overseer to obtain such information and act on it. If the server cutover happens during the execution of a job, then the job will continue to run on the new server. Depending on how the controller is written, the job may or may not need to restart from the beginning but can continue from a previously saved snapshot.

Simulator FLARE provides a simulator to allow data scientists and system developers to easily write new *FLComponents* and novel workflows. The simulator is a command line tool to run a FLARE job. To allow simple experimentation and debugging, the FL server and multiple clients run in the same process during simulation. A multi-process option allows making efficient use of resources, e.g., training multiple clients on different GPUs. The simulator follows the same job execution as in real-world FLARE deployment. Therefore, components developed in simulation can be directly deployed in real-world federated scenarios.

3.2 Secure Provisioning in FLARE

Security is an important requirement for federated learning systems. FLARE provides security solutions in the following areas: authentication, communication confidentiality, user authorization, data privacy protection, auditing, and local client policies.

Authentication FLARE ensures the identities of communicating peers with the use of mutual Transport Layer Security (TLS). Each participating party (FL Servers, Overseer, FL Clients, Admin Clients) must be properly provisioned. Once provisioned, each party receives a startup kit, which contains TLS credentials (public cert of the root, the party's own private key and certificate) and system endpoint information, see Fig. 3. Each party can only connect to the FLARE system with the startup kit. Communication confidentiality is also achieved with the use of TLS-based messaging.

Federated Authorization FLARE's admin command system is very rich and powerful. Not every command is for everyone. FLARE implements a role-based user authorization system that controls what a user can or cannot do. At the time of provision, each user is assigned a role. Authorization policies specify which commands are permitted for which roles. Each FL client can define its own authorization policy that specifies what a role can or cannot do to the client. For example, one client could allow a role to run jobs from any researchers, whereas another client may only allow jobs submitted by its own researchers (i.e., the client and the job submitter belong to the same organization).

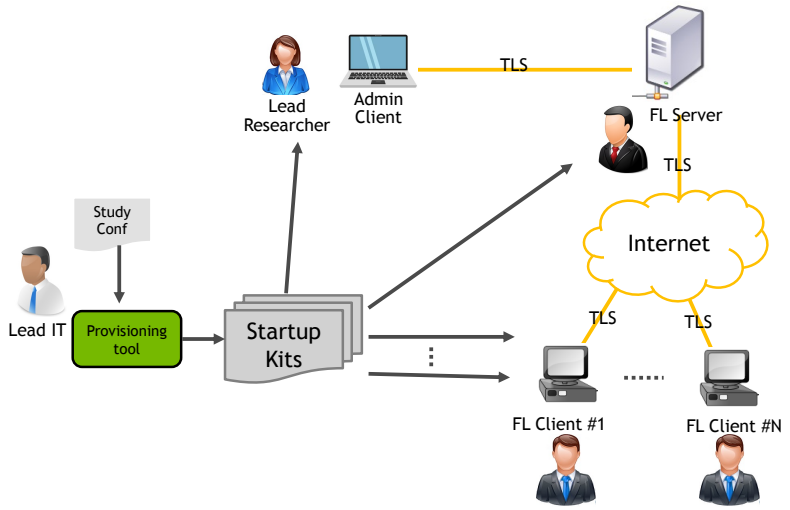


Figure 3: High-level steps for running a real-world study with secure provisioning with FLARE.

FLARE automatically records all user commands and job events in system audit files on both the server and client sides. In addition, the audit API can be used by application developers to record additional events in the audit files.

Client-Privacy FLARE enhances the overall system security by allowing each client to define its own policies for authorization, data privacy (filters), as well as computing resource management. The client can change its policies at any time after the system is up and running without having to be re-provisioned. For example, the client could require that all jobs running on it are subject to a set of filters. The client could also change the number of computing resources (e.g., GPU devices) to be used by the FL client.

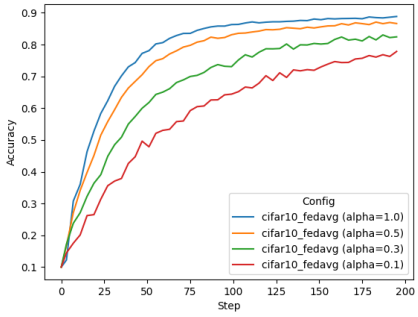
Federated Data Science As a general distributed computing platform, FLARE can be used for various applications in different industries. Here we describe some of the most common use cases where FLARE was deployed so far.

Federated Learning A go-to example dataset for benchmarking different FL algorithms is CIFAR-10. FLARE allows users to experiment with different algorithms and data splits using different levels of heterogeneity based on a Dirichlet sampling strategy [28]. Figure 4a shows the impact of different alpha values, where lower values cause higher heterogeneity on the performance of the FedAvg.

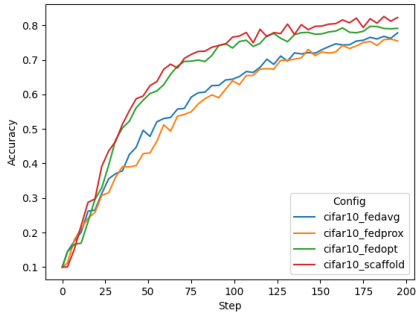
Apart from FedAvg, currently available in FLARE include FedProx [14], FedOpt [20], and SCAFFOLD [12]. Figure 4b compares an α setting of 0.1, causing a high data heterogeneity across clients and its impact on more advanced FL algorithms, namely FedProx, FedOpt, and SCAFFOLD. FedOpt and SCAFFOLD show markedly better convergence rates and achieve better performance than FedAvg and FedProx with the same alpha setting. SCAFFOLD achieves this by adding a correction term when updating the client models, while FedOpt utilizes SGD with momentum to update the global model on the server. Therefore, both achieve better performance with the same number of training steps as FedAvg and FedProx.

Other algorithms available in or coming soon to FLARE include federated XGBoost [5], Ditto [13], FedSM [31], Auto-FedRL [8], and more.

Federated Statistics FLARE provides built-in federated statistics operators (*Controller* and *Executors*) that will generate global statistics based on local client statistics. Each client could have one or more datasets, such as "train" and "test" datasets. Each dataset may have many features. For each feature in the dataset, FLARE will calculate the statistics and combine them to produce global statistics for all the numeric features. The output gathered on the server will be the complete statistics for all datasets in clients and global, as illustrated in Fig. 5.



(a) FedAvg with increasing levels of heterogeneity (smaller α values).

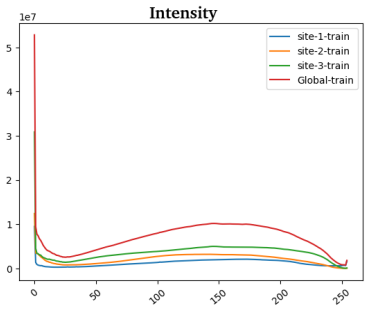


(b) FL algorithms with a heterogeneous data split (α=0.1).

Figure 4: Federated learning experiments with FLARE.

	count	histogram
site-2-train	6012	[[0.0, 1.0, 12430030], [1.0, 2.0, 3511491], [2...
site-4-train	1345	NaN
site-3-train	10192	[[0.0, 1.0, 30867349], [1.0, 2.0, 4553187], [2...
site-1-train	3616	[[0.0, 1.0, 9512374], [1.0, 2.0, 1381654], [2....
Global-train	21165	[[0.0, 1.0, 52809753], [1.0, 2.0, 9446332], [2...

(a) Federated statistics. Note, the data of "site-4" violates the client's privacy policy and therefore does not share its statistics with the server.

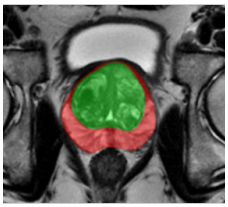


(b) Histogram visualization.

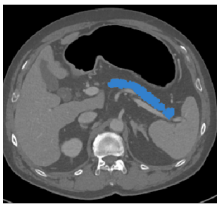
Figure 5: Federated statistics with FLARE.

Real-world Use Cases FLARE and its predecessors have been used in several real-world studies exploring FL for healthcare scenarios. The collaborations between multinational institutions tested and validated the utility of federated learning, pushing the envelope for training robust, generalizable AI models. These initiatives included FL for breast mammography classification [23], prostate segmentation [24], pancreas segmentation [28], and most recently, chest X-ray (CXR) and electronic health record (EHR) analysis to predict the oxygen requirement for patients arriving in the emergency department with symptoms of COVID-19 [6].

(a) Mammography.



(b) Prostate.



(c) Pancreas.

(d) CXR & EHR.

Figure 6: Real-world use cases of FLARE.

4 Summary & Conclusion

We described NVIDIA FLARE, an open-source SDK to make it easier for data scientists to use FL in their research and to allow an easy transition from research to real-world deployment. As discussed above, FLARE's *Controller* programming API supports various interaction patterns between the server and clients over internet connections, which could be unstable. Therefore, the API design mitigates

various failure conditions and unexpected crashes of the client machines such as allowing developers to process timeout conditions properly. FLARE is an open-source project. We invite the community to contribute and grow FLARE.

We did not go into all details of exciting features available in FLARE, like homomorphic encryption, TensorBoard streaming, provisioning web dashboard, integration with MONAI[3] [19, 3], etc. However, we hope that this overview of FLARE gives a good starting point for developers and researchers on their journey to using FL and federated data science in simulation and in the real world. For more information, please visit the code repository at `https://github.com/NVIDIA/NVFlare`.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283.

[2] Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., de Gusmão, P. P., and Lane, N. D. (2020). Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.

[3] Cardoso, M. J., Li, W., Brown, R., Ma, N., Kerfoot, E., Wang, Y., Murrey, B., Myronenko, A., Zhao, C., Yang, D., et al. (2022). Monai: An open-source framework for deep learning in healthcare. *arXiv preprint arXiv:2211.02701*.

[4] Chang, K., Balachandar, N., Lam, C., Yi, D., Brown, J., Beers, A., Rosen, B., Rubin, D. L., and Kalpathy-Cramer, J. (2018). Distributed deep learning networks among institutions for medical imaging. *Journal of the American Medical Informatics Association*, 25(8):945–954.

[5] Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

[6] Dayan, I., Roth, H. R., Zhong, A., Harouni, A., Gentili, A., Abidin, A. Z., Liu, A., Costa, A. B., Wood, B. J., Tsai, C.-S., et al. (2021). Federated learning for predicting clinical outcomes in patients with covid-19. *Nature medicine*, 27(10):1735–1743.

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

[8] Guo, P., Yang, D., Hatamizadeh, A., Xu, A., Xu, Z., Li, W., Zhao, C., Xu, D., Harmon, S., Turkbey, E., et al. (2022). Auto-fedrl: Federated hyperparameter optimization for multi-institutional medical image segmentation. *arXiv preprint arXiv:2203.06338*.

[9] Hatamizadeh, A., Yin, H., Molchanov, P., Myronenko, A., Li, W., Dogra, P., Feng, A., Flores, M. G., Kautz, J., Xu, D., et al. (2022). Do gradient inversion attacks make federated learning unsafe? *arXiv preprint arXiv:2202.06924*.

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

[11] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2019). Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*.

[12] Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S., and Suresh, A. T. (2020). Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR.

[13] Li, T., Hu, S., Beirami, A., and Smith, V. (2021). Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR.

[14] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450.

[3] `https://monai.io`

[15] Li, W., Milletarì, F., Xu, D., Rieke, N., Hancox, J., Zhu, W., Baust, M., Cheng, Y., Ourselin, S., Cardoso, M. J., et al. (2019). Privacy-preserving federated brain tumour segmentation. In *International workshop on machine learning in medical imaging*, pages 133–141. Springer.

[16] Liu, Y., Fan, T., Chen, T., Xu, Q., and Yang, Q. (2021). Fate: An industrial grade platform for collaborative learning with data protection. *J. Mach. Learn. Res.*, 22(226):1–6.

[17] Ludwig, H., Baracaldo, N., Thomas, G., Zhou, Y., Anwar, A., Rajamoni, S., Ong, Y., Radhakrishnan, J., Verma, A., Sinn, M., et al. (2020). Ibm federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987*.

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

[19] MONAI Consortium (2022). MONAI: Medical Open Network for AI.

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

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

[22] Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S., Bakas, S., Galtier, M. N., Landman, B. A., Maier-Hein, K., et al. (2020). The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7.

[23] Roth, H. R., Chang, K., Singh, P., Neumark, N., Li, W., Gupta, V., Gupta, S., Qu, L., Ihsani, A., Bizzo, B. C., et al. (2020). Federated learning for breast density classification: A real-world implementation. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 181–191. Springer.

[24] Sarma, K. V., Harmon, S., Sanford, T., Roth, H. R., Xu, Z., Tetreault, J., Xu, D., Flores, M. G., Raman, A. G., Kulkarni, R., et al. (2021). Federated learning improves site performance in multicenter deep learning without data sharing. *Journal of the American Medical Informatics Association*, 28(6):1259–1264.

[25] Sheller, M. J., Edwards, B., Reina, G. A., Martin, J., Pati, S., Kotrotsou, A., Milchenko, M., Xu, W., Marcus, D., Colen, R. R., et al. (2020). Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific reports*, 10(1):1–12.

[26] Sheller, M. J., Reina, G. A., Edwards, B., Martin, J., and Bakas, S. (2018). Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In *International MICCAI Brainlesion Workshop*, pages 92–104. Springer.

[27] Silva, S., Altmann, A., Gutman, B., and Lorenzi, M. (2020). Fed-biomed: A general open-source frontend framework for federated learning in healthcare. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 201–210. Springer.

[28] Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. (2020). Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*.

[29] Warnat-Herresthal, S., Schultze, H., Shastry, K. L., Manamohan, S., Mukherjee, S., Garg, V., Sarveswara, R., Händler, K., Pickkers, P., Aziz, N. A., et al. (2021). Swarm learning for decentralized and confidential clinical machine learning. *Nature*, 594(7862):265–270.

[30] Xie, Y., Wang, Z., Chen, D., Gao, D., Yao, L., Kuang, W., Li, Y., Ding, B., and Zhou, J. (2022). Federatedscope: A comprehensive and flexible federated learning platform via message passing. *arXiv preprint arXiv:2204.05011*.

[31] Xu, A., Li, W., Guo, P., Yang, D., Roth, H. R., Hatamizadeh, A., Zhao, C., Xu, D., Huang, H., and Xu, Z. (2022). Closing the generalization gap of cross-silo federated medical image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20866–20875.

[32] Ziller, A., Trask, A., Lopardo, A., Szymkow, B., Wagner, B., Bluemke, E., Nounahon, J.-M., Passerat-Palmbach, J., Prakash, K., Rose, N., et al. (2021). Pysyft: A library for easy federated learning. In *Federated Learning Systems*, pages 111–139. Springer.