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 building 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. It allows researchers to apply their data science workflows in any training libraries (PyTorch, TensorFlow, XGBoost, or even NumPy) in real-world FL settings. This paper introduces the key design principles of NVFlare 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 [31]. 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 (NVFlare), 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

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Contact: {hroth,yanc,chesterc,daguangx,pdogra,andyf}@nvidia.com
[1]Code is available at https://github.com/NVIDIA/NVFlare.

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 [35, 34]. NVFlare is not limited to applications in healthcare and is designed to allow cross-silo FL [15] 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 [44], FedML [11], FATE [23], Flower [2], OpenFL [30], Fed-BioMed [36], IBM Federated Learning [24], HP Swarm Learning [38], FederatedScope [40], FLUTE [7], and more. Some focus on simulated FL settings for researchers, while others prioritize production settings. NVFlare aims to be useful for both scenarios: 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

NVIDIA FLARE – or short NVFlare – stands for "**NV**IDIA **F**ederated **L**earning **A**pplication **R**untime **E**nvironment". The SDK enables researchers and data scientists to adapt their machine learning and deep learning workflows to a federated paradigm. It enables platform developers to build a secure, privacy-preserving offering for distributed multiparty collaboration.

NVFlare is a lightweight, flexible, and scalable FL 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 [25], can be implemented in NVFlare using the following main steps. Starting from an initial global model, each FL client trains the model on their local data for a while 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, NVFlare is a generic approach for supporting collaborative computing across multiple clients. NVFlare provides the *Controller* programming API for researchers to create workflows for coordinating clients for 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 has two flavors: wait (block until client results are received) or no-wait. A workflow developer can use 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.

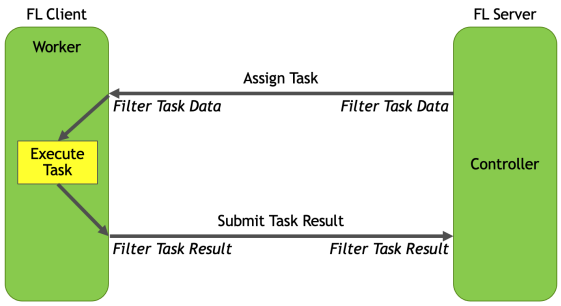


Figure 1: NVFlare job execution. The *Controller* is a Python object that controls or coordinates the *Workers* to get a job done. The controller is run on the FL server. A *Worker* is capable of performing tasks. *Workers* run on FL clients.

Key Components NVFlare is built on a componentized architecture that allows FL workloads to move 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.
- **NVFlare Dashboard** for simplified project management, secure provisioning, and deployment, orchestration.
- **Reference FL algorithms** (e.g., FedAvg, FedProx, SCAFFOLD) and workflows, like scatter and gather, cyclic, etc.
- **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.

High-Level Architecture NVFlare 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. NVFlare 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 using Notebooks or the NVFlare Console – an interactive command tool. NVFlare 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 NVFlare'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 NVFlare 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 NVFlare and the information that can be passed between them.

Workers and Controller NVFlare'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*. NVFlare provides both single- and multi-process executors to implement different computing workloads.

FLContext One of the most important features of NVFlare 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.

Communication Drivers NVFlare abstracts the communication layers out so that different deployment scenarios can implement customizable communication drivers. By default, we use GRPC for data communication in task-based communication. However, the driver can be replaced to run other communication protocols, for example, TCP. The customizable nature of communication in NVFlare allows for both server-centric and peer-to-peer communication patterns. This enables the user to utilize both scatter and gather-type workflows like FedAvg [25], decentralized training patterns like swarm learning [38], or direct peer-to-peer communication as in split learning [9].

Fig. 2 compares the times for model upload and download from the client's perspective using different communication protocols available in NVFlare using a model of $\sim$18MB in size.

The experiment runs in a multi-cloud environment with the server and eight clients running on Azure, while two clients run on AWS. One can observe that the global model download is slower as all clients are trying to download the global model at the same time, and hence the server is more busy. In contrast, the clients' model uploads happen at slightly different times and therefore are faster. One can also see how this multi-cloud setup causes the clients on AWS to take slightly longer during model download due to communication across different cloud infrastructures.

Filters Filters in NVFlare are a type of *FLComponent* that have a process method to transform the *Shareable* object between the communicating parties. A Filter can 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 NVFlare's primary mechanism for data privacy protection [21, 10]:

- *ExcludeVars* to exclude variables from shareable.

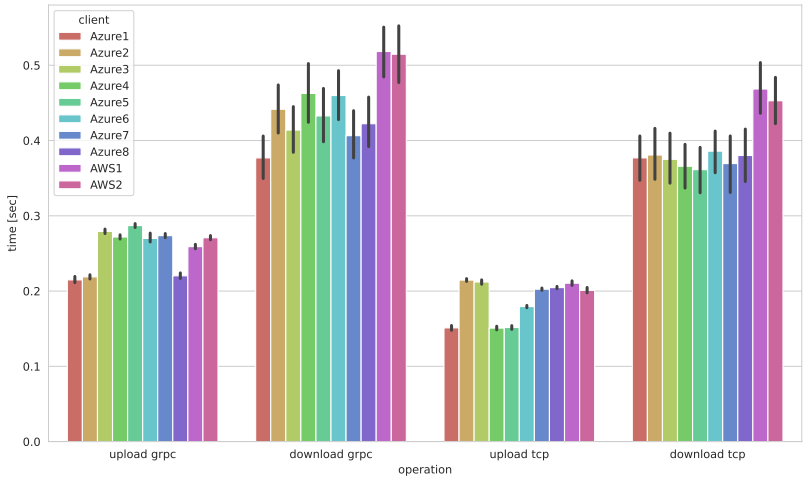


Figure 2: Comparison of GRPC and TCP communication drivers in NVFlare. The server is running on Azure. The clients are distributed between Azure and AWS. The message size is ∼18MB. Communication times were measured over 100 rounds of FedAvg. Error bars indicate the 95% confidence intervals.

- *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 when using homomorphic encryption for secure aggregation[2]. We compare raw data to encrypted model gradients uploaded in Table 1 when hosting the server on AWS[3] and connecting 30 client instances using an on-premise GPU cluster. One can see the longer upload times due to the larger message sizes needed by homomorphic encryption.

Table 1: Federated learning exchanging homomorphic encrypted vs. 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

Event Mechanism NVFlare 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

[2] `https://developer.nvidia.com/blog/federated-learning-with-homomorphic-encryption`

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

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

NVFlare 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. NVFlare 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 and act on such information. 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 NVFlare 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 NVFlare 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 efficient use of resources, e.g., training multiple clients on different GPUs. The simulator follows the same job execution as in real-world NVFlare deployment. Therefore, components developed in simulation can be directly deployed in real-world federated scenarios.

3.2 Secure Provisioning in NVFlare

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

Authentication NVFlare ensures the identities of communicating peers using 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 containing 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 NVFlare system with the startup kit. Communication confidentiality is also achieved with the use of TLS-based messaging.

Federated Authorization NVFlare's admin command system is very rich and powerful. Not every command is for everyone. NVFlare 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 authorization policy that specifies what a role can or cannot do to the

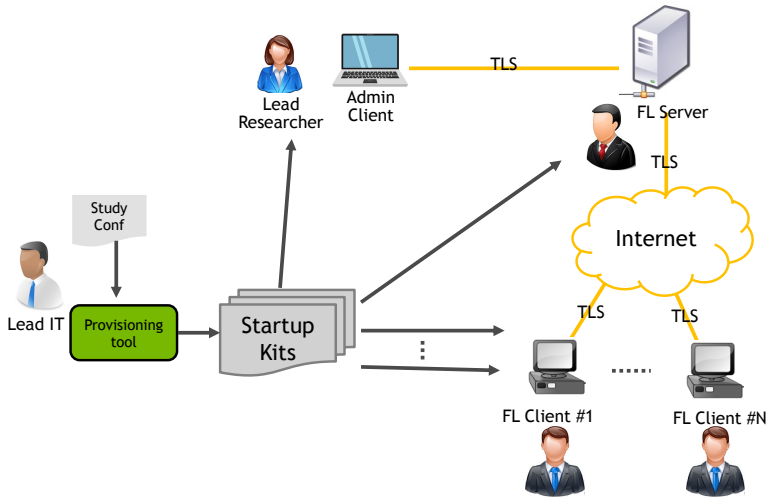


Figure 3: High-level steps for running a real-world study with secure provisioning with NVFlare.

client. For example, one client could allow a role to run jobs from any researchers. In contrast, another client may only allow jobs submitted by its researchers (i.e., the client and the job submitter belong to the same organization).

NVFlare 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 NVFlare enhances the overall system security by allowing each client to define its policies for authorization, data privacy (filters), and 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 all jobs running on it to be 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.

4 Federated Data Science

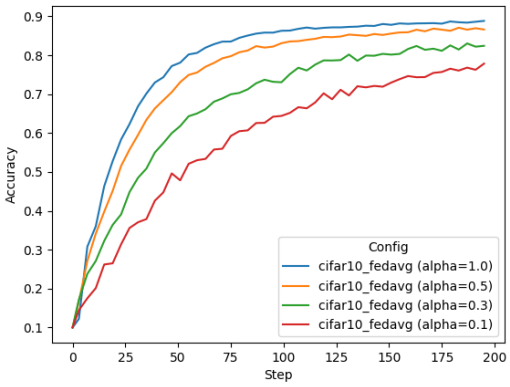
As a general distributed computing platform, NVFlare can be used for various applications in different industries. Here we describe some of the most common use cases where NVFlare was deployed.

4.1 Federated Deep Learning

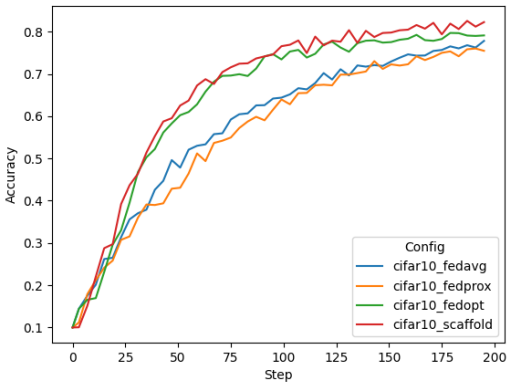
A go-to example dataset for benchmarking different FL algorithms is CIFAR-10 [17]. NVFlare allows users to experiment with different algorithms and data splits using different levels of heterogeneity based on a Dirichlet sampling strategy [37]. Figure 4a shows the impact of varying alpha values, where lower values cause higher heterogeneity on the performance of the FedAvg.

Apart from FedAvg, currently available in NVFlare include FedProx [20], FedOpt [29], and SCAFFOLD [16]. 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 perform better with the same number of training steps as FedAvg and FedProx.

Other algorithms available in or coming soon to NVFlare include federated XGBoost [5], Ditto [19], FedSM [41], Auto-FedRL [8], and more.



(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 NVFlare.

4.2 Federated Machine Learning

Traditional machine learning methods, such as linear models, support vector machine (SVM), and k-means clustering, can be formulated under a federated setting.

With certain libraries, the federated machine learning algorithms need to be designed considering two factors: algorithm-wise, each of these models has distinct training schemes and model representations; and implementation-wise, popular libraries providing these functionalities (e.g., scikit-learn, XGBoost) have different APIs and inner logics. Hence, when developing an FL variant of a particular traditional machine learning method, several questions need to be answered at these two levels:

First, at the algorithm level, we need to break down the optimization process into individual steps/rounds (if possible) and have answers to three major questions:

1. What information should clients share with the server?
2. How should the server aggregate the collected information from clients?
3. What should clients do with the global aggregated information received from the server?

Second, at the implementation level, we need to know what APIs are available and how to utilize them in a federated pipeline to implement a distributed version of the algorithm.

A major difference between federated traditional machine learning and federated deep learning is that, for traditional machine learning methods, the boundary between "federated" and "distributed", or even "ensemble", can be much more vague than for deep learning. Due to the characteristics of a given algorithm and its API design, the concepts can be equivalent. Take XGBoost and SVM, for example: Algorithm-wise, XGBoost can distribute the training samples to several workers and construct trees based on the collected histograms from each worker. Such a process can be directly adopted under a federated setting because the communication cost is affordable. In this case, "federated" is equivalent to "distributed" learning. API-wise, some algorithms can be constrained by their implementation. Take scikit-learn's SVM for instance. Although theoretically SVM can be formulated as an iterative optimization process, the API only supports one-shot "fitting" without the capability of separately calling

the optimization steps. Hence a federated SVM algorithm using the scikit-learn library can only be implemented as a two-step process. In this case, "federated" is equivalent to "ensemble".

For clarification, we provide the full formulation for tree-based federated XGBoost, illustrated in Fig. 5:

1. XGBoost, by definition, is a sequential optimization process: each step adds one extra tree to the model to reduce the residual error. Hence, federated XGBoost can be formulated as follows: each round of FL corresponds to one boosting step at the local level. Clients share the newly added tree trained on local data with the server at the end of local boosting.

2. The model representation is a decision/regression tree. To aggregate the information from all clients, the server will bag all received trees to form a "forest" to be added to the global boosting model.

3. With the updated global model from the server, each client will continue the boosting process by learning a new tree starting from the global model of the boosted forest.

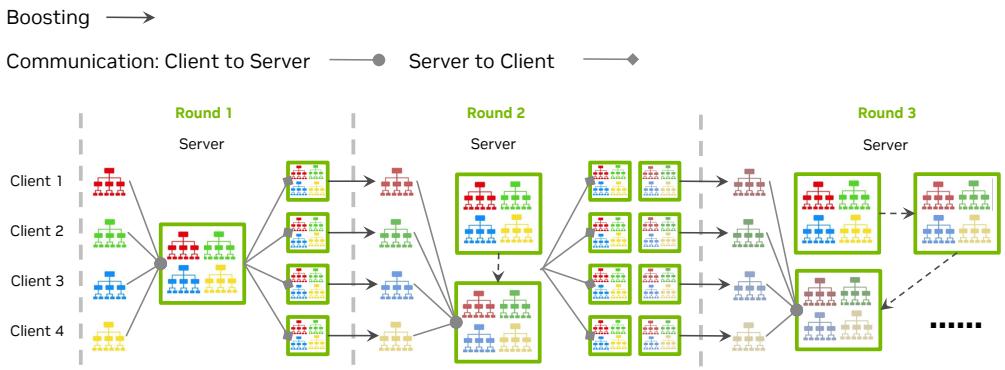


Figure 5: Tree-based federated XGBoost: a "boosting of forests."

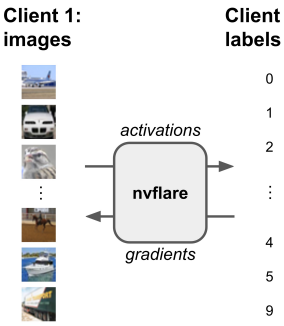
4.3 Split learning

Split learning assumes a vertical data partitioning [42] that can be useful in many distributed learning scenarios involving neural network architectures [9].

As an introductory example, we can assume that one client holds the images, and the other holds the labels to compute losses and accuracy metrics. Activations and corresponding gradients are being exchanged between the clients using NVFlare, as illustrated in Fig. 6. We use a cryptographic technique called private set intersection (PSI) [39] to compute the alignment between images and labels on both clients. NVFlare's implementation of PSI can be extended to multiple parties and applied to other use cases than split learning, e.g., requiring a secure and privacy-preserving alignment of different databases.

Using NVFlare's capability to implement different communication patterns, we can investigate the communication speed-ups one can achieve by implementing split learning using direct peer-to-peer communication as opposed to routing the messages between the two clients through a central server.

The table in Fig. 6 compares the training speeds of split learning on the CIFAR-10 dataset in a local simulation scenario. First, we use the same PyTorch script to simulate split learning. Then, we implement two distributed solutions using NVFlare. One that routes the messages through the server and one using a direct peer-to-peer connection between the clients. As expected, the direct peer-to-peer connection is more efficient, achieving only a slight overhead in total training time compared to the standalone PyTorch script, which could not be translated to real-world scenarios.



Setup	Training Time [min]
Simulated PyTorch	19
Routing through server (TCP)	27
Peer-to-peer (TCP)	25

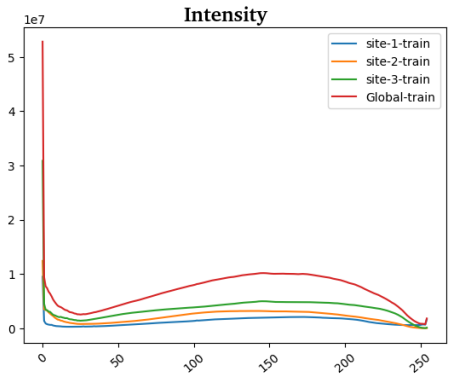
Figure 6: Simple split learning scenario using CIFAR-10. The table compares multiple communication patterns. Using 50,000 training samples and 15,625 rounds of communication with a batch size of 64.

4.4 Federated Statistics

NVFlare 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. NVFlare will calculate and combine the statistics for each feature in the dataset 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. 7.

	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.



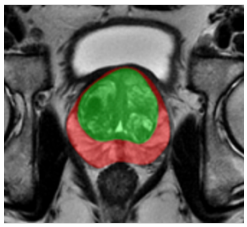
(b) Histogram visualization.

Figure 7: Federated statistics with NVFlare.

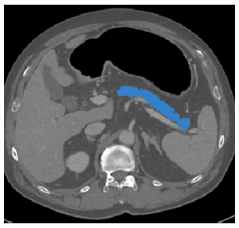
5 Real-world Use Cases

NVFlare 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 [32], prostate segmentation [33], pancreas segmentation [37], 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 8: Real-world use cases of NVFlare.

6 Summary & Conclusion

We described NVFlare, 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, NVFlare'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.

NVFLare's unique flexibility and agnostic approach towards the deployed training libraries make it the perfect solution for integrating with different deep learning frameworks, including popular ones used for training large language models (LLM). With our dedication to addressing the current limitations of communication protocols, we are working towards supporting the communication of large message sizes, enabling the federated fine-tuning of AI models with billions of parameters, such as those used for ChatGPT [28] and GPT-4 [27]. Moreover, our team is implementing parameter-efficient federated methods to adapt LLM models to downstream tasks [43], utilizing techniques such as prompt tuning [18] and p-tuning [22], adapters [13, 12], LoRA [14], showing promising performance. Our commitment to innovation and excellence in this field ensures that we continue to push the boundaries of what is possible with federated learning.

We did not go into all details of exciting features available in NVFlare, like homomorphic encryption, TensorBoard streaming, provisioning web dashboard, integration with MONAI[4] [26, 3], etc. However, we hope that this overview of NVFlare gives a good starting point for developers and researchers on their journey to using FL and federated data science in simulation and the real world.

NVFlare is an open-source project. We invite the community to contribute and grow NVFlare. For more information, please visit the code repository at `https://github.com/NVIDIA/NVFlare`.

References

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

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

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

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

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

[5] T. Chen and C. Guestrin. 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, 2016. ACM.

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

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

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

[9] O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

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

[11] 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.

[12] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[13] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[14] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

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

[17] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

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

[20] 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.

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

[22] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021.

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

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

[25] 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.

[26] MONAI Consortium. MONAI: Medical Open Network for AI, 9 2022.

[27] OpenAI. Gpt-4 technical report, 2023.

[28] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[29] 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.

[30] 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.

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

[32] H. R. Roth, K. Chang, P. Singh, N. Neumark, W. Li, V. Gupta, S. Gupta, L. Qu, A. Ihsani, B. C. Bizzo, et al. 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, 2020.

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

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

[35] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas. 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, 2018.

[36] S. Silva, A. Altmann, B. Gutman, and M. Lorenzi. 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, 2020.

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

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

[39] Wikipedia contributors. Private set intersection — Wikipedia, the free encyclopedia, 2023. [Online; accessed 27-April-2023].

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

[41] A. Xu, W. Li, P. Guo, D. Yang, H. R. Roth, A. Hatamizadeh, C. Zhao, D. Xu, H. Huang, and Z. Xu. 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, 2022.

[42] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[43] H. Zhao, W. Du, F. Li, P. Li, and G. Liu. Reduce communication costs and preserve privacy: Prompt tuning method in federated learning. *arXiv preprint arXiv:2208.12268*, 2022.

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