Federated learning: a comparison of methods

How do different Federated Learning frameworks compare?

Cristea Vlad-Andrei

Supervisor(s): Marcel Reinders, Swier Garst

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Cristea Vlad-Andrei
Final project course: CSE3000 Research Project
Thesis committee: Marcel Reinders, Swier Garst, Lydia Chen

Abstract

Federated Learning is a machine learning paradigm for decentralized training over different clients. The training happens in rounds where each client learns a specific model which is then aggregated by a central server and passed back to the clients. Since the paradigm's inception, many frameworks that provide Federated Learning tools and infrastructure have appeared. This leads to the question of "How do different Federated Learning frameworks compare?", which is the research question of this paper. The paper's main contribution will be helping developers new to the Federated Learning field decide between NVidia Flare, OpenFL, and Flower, three popular federated learning frameworks.

1 Introduction

Federated Learning is a machine learning paradigm that allows decentralized learning to occur on different machines or clients. In this approach, each client has its own local dataset and trains its own model, which is then aggregated by a central server in each learning round. This decentralized approach is highly important in the medical industry, where sharing sensitive data such as medical records is highly regulated or even prohibited due to privacy concerns. Federated Learning provides a safe way of exchanging information between medical centers without compromising patient confidentiality. For example, the paradigm could potentially speed up research into treating and preventing various medical conditions, such as cancer, by creating a model from lung or even brain tumor scans from multiple centers.

Previous work on Federated Learning led to the proposal of the FedAvg method by Google researchers in 2016 [10]. Despite progress in the field, many research questions remain unanswered. For instance, the increasing popularity of Federated Learning has led to the creation of multiple frameworks that help the development of Federated Learning algorithms. However, most of these frameworks are similar or provide more or less the same tools, making it difficult to determine which one is best suited for a particular task. As a result, researchers and developers often face the challenge of selecting the most appropriate Federated Learning platform for their needs.

The central focus of this paper revolves around a pivotal research question: "How do three prominent Federated Learning platforms, namely OpenFL, Flare, and Flower, compare in terms of performance, usability, and setup?"[12][11][4]. To shed light on this inquiry, the paper establishes a set of metrics to evaluate these factors and subsequently presents benchmarking results for each platform. By comparing these results, the paper tries to empower researchers and developers in selecting the most suitable Federated Learning platform tailored to their specific requirements. Ultimately, the aim is to enhance productivity in the field of Federated Learning research and development. Nonetheless, the idea of comparing different Federated Learning frameworks came from UniFed, a platform that tries to accomplish that with other frameworks already. [9]

Since properly evaluating the frameworks based on performance is a complex task, the focus of the paper will be analyzing the heuristics component (which will be discussed in the Methodology section) and ultimately helping users of various prior knowledge (from developers inexperienced with Federated Learning to researchers) decide what platform to choose.

The structure of this scientific paper is carefully designed to ensure a coherent and systematic presentation of the research findings. Chapter 2 lays the foundation by outlining the methodology employed, providing a comprehensive understanding of the research process. Chapter 3 contains a brief overview of each framework, focusing on the aspects that set them apart. Following this, Chapter 4 conducts an in-depth benchmarking analysis that contains both a heuristics and a performance evaluation. In Chapter 5, the results of the benchmarking are discussed, with a particular focus on comparing the three frameworks. Responsible research practices are emphasized in Chapter 6, highlighting the ethical considerations of the study. Lastly, the concluding chapter summarizes the key findings and presents potential avenues for future research.

2 Methodology or Problem Description

In order to compare OpenFL, Flare and Flower, each framework will be assessed based on two main categories: general heuristics and performance. In the following subsections each category will be explained.

2.1 Heuristics evaluation

Heuristics represent non-functional characteristics of a framework that are crucial in order to help developers and researchers decide between one of the three frameworks that will be benchmarked. The main heuristics covered in the evaluation will be quality of documentation, available example experiments, learning curve of using and setting up a specific framework, quality of documentation of the code. Besides these general heuristics, a special part of the evaluation will include the frameworks' capabilities. This section will cover the available tools of the framework, the machine learning models supported, as well as what aggregating algorithms can be used. This will provide a deeper insight into what areas of federated learning each framework excels at and will be a central component that dictates choosing one tool over another one.

2.2 Performance evaluation

The performance evaluation will include a single experiment that will be replicated for each framework. The goal is to try to have the same experiment reproduced over all three frameworks in order to gain an insight into the capabilities of the underlying infrastructure when processing the same task. The

experiments will use the CIFAR10 [8] dataset and use a convolutional neural network as the machine learning model, for which the code will be visible in the appendix. The main experiment will compare the learning curve on the training dataset over the epochs.

3 Frameworks

The follow section will provide a brief overview of each framework used in the research project, namely Nvidia Flare, OpenFL and Flower.

3.1 Nvidia Flare

"NVIDIA FLARE™ (NVIDIA Federated Learning Application Runtime Environment) is a domain-agnostic, open-source, and extensible SDK for Federated Learning. It allows researchers and data scientists to adapt existing ML/DL workflow to a federated paradigm and enables platform developers to build a secure, privacy-preserving offering for a distributed multi-party collaboration.", according to the main page of the framework's documentation. [11]

Nvidia Flare is a powerful technology that offers a range of commonly-used algorithms to facilitate the development of Federated Learning Workflows while emphasizing best practices. It encompasses 2 training workflows(Scatter and Gather (SAG) and Cyclic) and 2 evaluation workflows (Cross-site Model Validation and Global Model Evaluation).

3.1.1 Key features

Training workflows
The Scatter and Gather workflow implements a hub and spoke model. In this approach, a central server disseminates tasks to be executed by client workers. After each client executed the task, the results are sent back to the server where they will be aggregated according to the selected aggregation algorithm.

Another notable workflow supported by Nvidia Flare is the Cyclic approach. This reference implementation enables the central server to issue a series of tasks that are cyclically executed among a group of participating clients. Each client processes a task and then sends it to the next client until the final client will return the final model to the server.

Evaluation workflows
Cross-site model validation is a workflow through which each model computed (both local models and the global models) is validated on each client's dataset. This works by sending all models to each client where they will be validated with the local dataset. The results are collected by the server which computes a matrix that compares all model performances vs all client datasets.

Global model validation is a simpler evaluation method where only the global model is sent to each client in order to be validated on the local datasets.

3.1.2 Privacy

Privacy preservation is a critical concern in federated learning, and FLARE addresses this through a range of privacy-preserving algorithms. These include techniques such as excluding variables, truncating weights by percentile, applying sparse vector methods, utilizing homomorphic encryption. These techniques ensure that sensitive data remains secure during the federated learning process.

Overall, NVIDIA FLARE serves as a comprehensive framework, providing researchers and developers with a simplified and efficient means to implement federated learning workflows. By incorporating best practices and privacy preservation algorithms, FLARE contributes to the advancement of federated learning technologies and their applications in various domains.

3.2 OpenFL

OpenFL is a Python federated learning platform initially developed by Intel. It mainly focuses on deep learning using neural networks, but similarly to NVIDIA Flare it is domain-agnostic and open-source.[12]

3.2.1 Key features

At present, OpenFL offers two discrete methodologies for configuring and executing experiments involving federated learning: the Director-based approach and the Aggregator-based approach. Each workflow presents distinct benefits and is appropriate for varying situations within a federation.

Director-Based Workflows
The OpenFL's Director-based workflow presents a more efficient and simplified method for establishing a federation. The aforementioned text introduces a pair of significant elements, namely the "Director" and the "Envoy."

The Director assumes the role of a primary coordinator within the federation. The aforementioned statement pertains to the management of the comprehensive orchestration of the process of federated learning. The system is responsible for managing various operations, including but not limited to the distribution of models, consolidation of model updates, and facilitation of communication among diverse participating entities. The role of the Director involves optimizing the workflow and facilitating the administration of the federation.

The term "Envoy" pertains to distinct units or apparatuses that exist within the federation. Every Envoy functions independently and conducts localized training on its corresponding datasets. Following the training process, the Envoy transmits the revised model parameters to the Director for the purpose of aggregation. This particular constituent facilitates the process of distributed and parallel training across numerous entities.

The utilization of the Director-based workflow presents notable benefits in situations where a federation necessitates durable components and a centralized coordination mechanism. The system provides a practical means of oversee-

ing the federated learning procedure and streamlines effective correspondence and cooperation among the Director and the Envoy entities involved.

Aggregator-Based Workflows
The Aggregator-based workflow, as implemented in OpenFL, is a suitable approach for situations where it is imperative to validate the workload prior to its execution. This workflow adheres to a conventional method of federated learning, emphasizing the consolidation of model updates.

The Aggregator-based workflow involves the fragmentation of a given workload or task into several sub-tasks, which are subsequently allocated to distinct entities or devices within the federation. The aforementioned entities engage in localized training procedures for their designated sub-tasks and subsequently transmit the model updates to a centralized Aggregator. Subsequently, the Aggregator executes the process of aggregating the updates of the models and generates the revised global model.

This particular workflow confers advantages in situations where it is imperative to authenticate the workload or subtasks prior to complete implementation. The utilization of this technique permits meticulous regulation of the training procedure and facilitates the implementation of verification and validation measures to guarantee precise and dependable model aggregation.

Each workflow presents distinct benefits and can be chosen according to the particular demands of the federated learning context. The workflow based on Director presents a streamlined and coordinated methodology, whereas the workflow based on Aggregator offers enhanced control and verification functionalities prior to ultimate aggregation.

3.2.2 Security

OpenFL uses TLS encryption [19] for client-server communication. The main ways of generating certificates are through manual and semi-automatic PKI workflows [18]. The manual workflow is already embedded in the aggregator-based workflow, while the semi-automatic one involves creating a certificate authority server that listens for signing requests. These requests come from the clients of the federation and need to be signed by the security authority via a token.

3.3 Flower

Flower is a problem-agnostic Federated Learning platform that has as one of its main design goals simplicity of setting up and usage. It is designed for both researchers and developers to use and it provides an infrastructure that is also independent of the underlying machine learning framework used.

3.3.1 Key features

Framework agnostic
One of the main advantages of using Flower is that it is machine-learning framework agnostic. This means that developers and researchers who have already worked with a specific machine learning framework can quickly adapt their experiments to a federated setting.

In the official documentation it is stated that "Flower can be used with any machine learning framework" [4]. Tutorials and setup guides are available for incorporating flower with the following frameworks : PyTorch, TensorFlow, Hugging Face Transformers, PyTorch Lightning, MXNet, scikit-learn, JAX, TFLite, fastai, Pandas and Numpy.

Customization
Customization is another key feature of the framework. The most important customization options are for creating custom aggregation strategies as well as changing the client API which is responsible for how each client in the federated network behaves.

The aggregation strategy can be changed in 2 main ways: by altering the configuration of an existing strategy or by creating a completely new strategy from scratch. While the former is much easier to adapt, it only provides small configuration changes such as manually passing the initial parameters of a neural network (which by default are sampled from a random client) or changing the fraction of clients sampled for training. The latter, however, allows building a whole strategy from scratch which can provide a higher control over each client node or even the aggregation. For example, in a custom strategy, each client can have different learning rates and the client sampling strategy can also be changed.

The client API can also be customized for more complex optimizations. The default client uses a Numpy configuration that is especially useful for serialization and deserialization which happens during the client-server communication rounds. However, by implementing a custom client the user can modify these processes and make the federated setting more performant. For example, the normal Numpy arrays can be converted into sparse matrices, before being sent to the server. This would not be possible to customize in the default configuration, where the Numpy client provides an interface that contains a built-in serialization/deserialization process which only accepts Numpy arrays. This way we can save bandwidth, as in certain cases where the weights of a complex convolutional neural network are sparse, converting them to a sparse matrix can greatly improve the size of the package being communicated to the server.

4 Benchmarking

The benchmarking process will be conducted in accordance with the methodology elucidated in chapter 2, encompassing an examination of two primary constituents: heuristics and performance evaluation.

4.1 Nvidia Flare Benchmarking

4.1.1 Heuristics evaluation

- **Documentation**
  The quality of the documentation is one of the areas where Nvidia Flare excels the most. The main parts of

documentation include key functionalities, example applications as well as separate guides for regular usage as well as programming guides for developers that want to build experiments on top of the available tools.

The documentation also includes different layers of depth that could help a wide range of users get started with the framework. Developers would much benefit for the extensive API documentation that includes all of the possible ways of manipulating the Nvidia Flare components, while researchers could make use of the multitude of example experiments to test the frameworks' limits.

The main focus of the documentation revolves around simulated experiments that run on a local machine. However, a special section is dedicated to Real-World Federated learning, where users can learn how to deploy experiments on multiple machines that run on-premises or on cloud.

- **Available examples**

  The overarching theme of the documentation (which is the different layers of depth it provides) can also be seen in the example experiments. There are several different sections that cover setting up the framework. After analysing each main section the ordering starting from low to high level of complexity is the following :

  1. Using and setting up Nvidia Flare.
  2. Experimenting with the main workflows of the framework.
  3. Using different Federating Learning Algorithms
  4. Deep learning experiments.
  5. Incorporating federated statistics.
  6. Experimenting with different development modes.
  7. Complex medical image analysis experiments.

- **Setup and learning curve**

  Setting up Nvidia Flare is described in a special part of the documentation. The only requirement specified is using Python 3.8+, however, the list of all supported operating systems are not mentioned. The platform does not appear to work on Windows systems, as being tested on a Windows 10 machine. The supported Linux distributions are not mentioned, which can make the setup process a trial-by-error experience, especially for users that are not software developers. The only reference to an operating system is Ubuntu 20.04, which is mentioned as part of the tutorial to set up a virtual environment. Also for newer versions of Ubuntu the user has to setup the framework only using Python3.8.

  After following the tutorial, the main command that runs the Nvidia Flare simulator was not working. The fix to the issue could be found by browsing the GitHub issue page, which might even cause potential users to give up setting up the tool since the solution is hard to find.

- **Quality of code documentation**

  The high quality of the code documentation can be observed for the example applications, where individual Jupyter notebooks are provided that explain step-by-step what each block of code does.

  Besides the notebooks, the main APIs are properly documented in the code, however, using the website API documentation was much easier to follow.

- **Frameworks capabilities**

  Nvidia Flare provides a wide range of capabilities that can also be understood from the quality of the documentation as well the example applications. The framework is suitable for developers (that work in the federated learning industry or not) as well as researchers and each user group can easily understand where to start just by reading through the documentation which is properly layered based on the complexity of the concepts as well as the expertise of the user. Also, the framework provides simulated as well as real-world federated learning capabilities which makes it a versatile tool.

  The framework is model-agnostic allowing users to develop their own models that can be either deep-learning models, machine-learning, or even statistical workflows.

  The available aggregation algorithms are Scaffold [6], FedProx [7], Fed Average [10] and Fed Opt [15].

4.1.2 Performance evaluation

The results of the CIFAR10 experiment can be found in the graphs presented in Figure 1 .

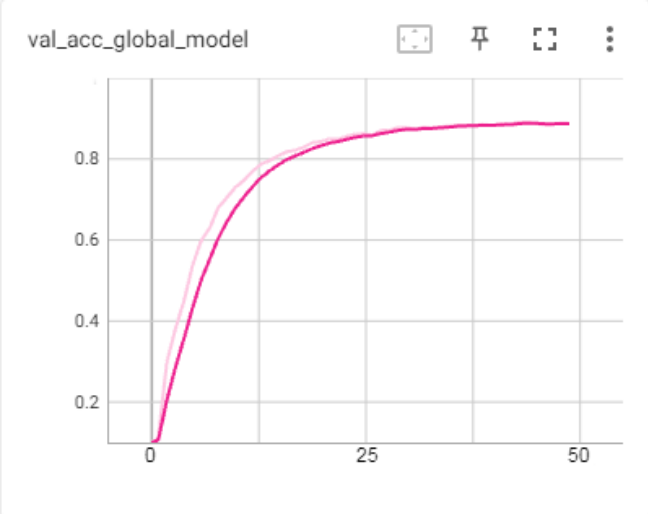

Figure 1: Accuracy over 50 rounds.

4.2 OpenFL Benchmarking

4.2.1 Heuristics evaluation

- **Documentation**

  The OpenFL documentation includes a section for each main part of the framework which are: manual, architecture of the system, and troubleshooting.

  The manual is the most important part of the documentation since it includes an installation and setup guide as well as example applications.

  Even though the documentation covers the most important topics of the framework, it fails to properly separate the topics based on complexity, which means the user can easily be lost within the complex technicalities of the system while just trying to set up the framework. Thus, the best way of understanding the framework is first going through the example applications and then coming back to the documentation.

- **Available examples**

  The documentation is misleading with regards to the tutorials since the official website documentation only mentions aggregator-based workflow tutorials, while the official github also contains tutorials for the director-based workflow.

  The OpenFL GitHub repository [16] contains a wide variety of tutorials in the form of jupyter [17] notebooks that cover experiments that use either TensorFlow [3] or PyTorch [13] as machine learning frameworks for each individual client.

  Even though the variety of the example experiments is impressive, the code is not commented in-depth and the user might need to read other pieces of documentation or even code in order to understand the underlying processes of the experiment.

- **Setup and learning curve**

  Setting up OpenFL is straightforward following the tutorial in the official documentation. However, there are no operating systems requirements, even though the setup does not work for Windows distributions. The only requirement is installing a Python 3.8 virtual environment and then following the tutorial.

  One major problem encountered after following the exact setup instructions was that the example applications tested (the Pytorch MNIST tutorial and the Keras Mnist tutorials) could not be run just by following the respective Jupyter notebooks. In order to properly run the experiments the source code had to be modified by modifying several methods.

- **Quality of code documentation**

  The documentation for the main components of the framework is good (namely the collaborator and aggregator APIs) however, there still exist TODO comments as well as comments that specify that an error might be encountered by the user. Also, there are comment snippets that mention that a fix needs to be implemented for the corresponding code fragment. The codebase documentation as well as the rather simple example application documentation suggests that the framework is not that well established and still under development before reaching a complete version.

- **Frameworks capabilities**

  The main focus of OpenFL is deep learning, which can be seen from the wide variety of deep-learning example tutorials. However, the framework's main advantage is being agnostic with regards to the underlying machine learning tool used, providing wrappers for both PyTorch and Tensorflow, 2 of the most widely used machine learning platforms.

  Even though the documentation does not provide a clear distinction between simulated and real-world federated learning workflows, the director-based worflow presents different roles of the federation. These are the experiment manager(a person or group who uses OpenFl), the director manager (the machine learning model creator) and the collaborator manager (which represents a singular client). On the other hand the aggregator-based workflow is mainly used for one-time simulated experiments and testing the framework.

  The available aggregation algorithms are FedAvg (for all frameworks), FedProx (for PyTorch and TensorFlow experiments), FedOpt(for all frameworks), FedCurv (for PyTorch experiments).

4.2.2 Performance evaluation

The results of the CIFAR10 experiment can be found in the graphs presented in Figure 2.

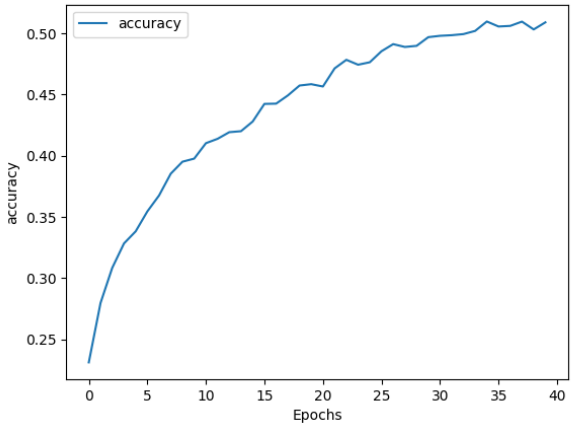

Figure 2: Accuracy over 40 training rounds

The experiment only includes 40 rounds since after that the machine ran out of memory and killing the process.

4.3 Flower benchmarking

4.3.1 Heuristics evaluation

- **Documentation**

  The Flower documentation is extensive and has a really good transition from simpler concepts to more complex ones. The very first section starts with an introduction to federated learning which puts an emphasis on the fact that the framework tries to welcome users of any experience level. Also, after the introduction to Federated Learning, a 4 step tutorial shows how the most important components of a federated setting are implemented using Flower.

  After this short but pragmatic on-boarding guide, the documentation contains quick-start guides for some of the most popular machine-learning frameworks which help users from the machine-learning community adapt their workflows to a federated setting.

  The documentation continues with more complex subjects, from which the most important ones are setup guides for using and configuring Flower, API, and CLI documentation as well as a dedicated section for future contributors to the Framework's progress.

  Overall, the documentation is really well organized and excels at helping users of various expertise levels get onboarded to Flower. The versatility of the framework is definitely reflected in the number of on-boarding guides for the machine learning frameworks which facilitates the transition to Federated Learning.

- **Available examples**

  Flower's GitHub repository [2] contains an extensive list of example applications. The main category represents quick-start guides for different machine-learning frameworks and technologies. However, the repository contains experiments that cover different facets of the framework such as using the simulation environment or advanced tutorials which use more clients and use more complex features of the framework.

  Besides the examples which are part of the GitHub repository, the main documentation page already contains most of the quick-start examples as well as the initial suite of onboarding tutorials, which might be easier to follow for users unfamiliar with Federated Learning.

  The most important aspect of the available examples is the abundance of explanations that follow each block of code (for introductory tutorials) or each CLI command (for advanced tutorials). Especially for the quick-start tutorials the explanations do not just stop at just the codes' functionality, but even include advantages of the approaches taken and even complex information about the API or the architecture of the framework.

- **Setup and learning curve**

  The setup guide is present in the official Flower documentation and it is easy to follow. The only restriction is using Python 3.7 or above and using a virtual environment is highly recommended. If the user skips the setup guide, all of the quickstart tutorials include an introduction to setting up the framework, which makes it really hard to get stuck.

  Flower examples use Poetry [14] to manage dependencies, but there exist tutorials for using other popular virtual environments such as Pyenv [1] or Anaconda [5]as well. After following the Poetry setup guide and following the flower installation tutorial, there were no errors and basic examples could be run smoothly from the first try.

- **Quality of code documentation**

  The quality of the code documentation is very good, especially for the example applications, where references to the system architecture and possible code customization options are explained to the user.

  Another great aspect of the code documentation is the API reference in the official documentation, which contains all of the information regarding Client and Server method utilization (which represent the main working parts of an experiment) as well as an extra section regarding helper function. Lastly, the documentation is completed by a short CLI guide for using Flower commands which are explained in depth.

- **Frameworks capabilities**

  The main advantage of using Flower is the ease of adaptability from a regular Machine Learning setting to a Federated one. Also, being able to adapt so many regular machine-learning frameworks leads to an even greater flexibility with regards to model selection, making the framework a great multi-purpose tool.

  Another key aspect of the tool is the ease of setup and the on-boarding experience which helps new inexperienced users get used to Federated Learning really fast. Also, the code structure is vastly designed around two major components which are the Client and the Server. This once again makes it much easier for new users to navigate the codebase and start developing custom solutions.

  The list of available aggregation algorithms is extensive, including FedAvg (alongside 3 variations of it), FedOpt, FedProx, FedAdagrad, FedAdam and FedYogi. [15]

4.3.2 Performance evaluation

The results of the CIFAR10 experiment can be found in the graphs presented in Figure 3.

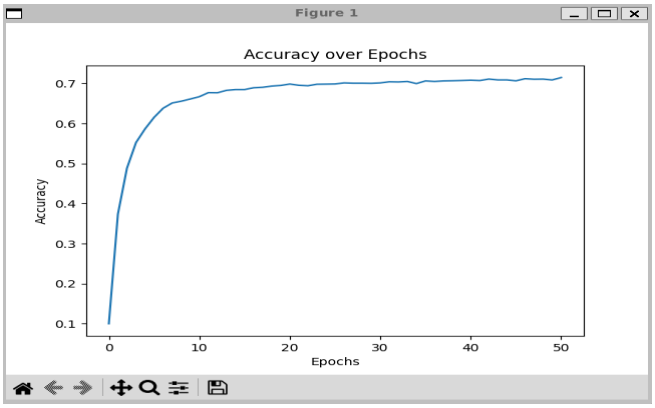

Figure 3: Accuracy over 50 training rounds.

5 Results

Each framework presented in the benchmarking section includes unique advantages and depending on the expertise and experience of the user one platform might be more suitable than the other 2. Since the key features and main functionalities have been already discussed, this section will rank the 3 frameworks based on the components discussed in the benchmarking section and then include a flow diagram that helps each type of user decide what framework to choose.

5.1 Heuristics ranking

- **Documentation**

  1. Flower
  2. Nvidia Flare
  3. OpenFL

  In terms of documentation, Flower provides the most user-friendly one, guiding the user from simple to complex concepts. Nvidia Flare also offers great documentation and the layering from simple to complex concepts is also present, but it is much harder to follow for an inexperienced user who is new to Federated Learning. OpenFL comes on last place, the documentation being more chaotic and harder to follow than the other 2 frameworks.

- **Available Examples**

  1. Nvidia Flare
  2. Flower
  3. OpenFL

  Nvidia Flare provides the largest variety of examples, ranging from introductory and complex examples. Also, each example is well explained which makes them easy to follow and implement by the user. While Flower also offers a large range of examples the majority of them focus on setting up different Machine Learning workflows to a federated setting, which makes the examples really specialised and sometimes even niche. OpenFL provides a good amount of great quality examples, but it specializes mainly on using PyTorch and TensorFlow workflows, since it specializes on neural networks so the variety does not compare to the other 2 platforms.

- **Setup and learning curve**

  1. Flower
  2. OpenFL
  3. NvidiaFlare

  The setup process of Flower is by far the easiest one out of the 3 platforms, with no errors being shown after following the installation and setup guides. In terms of setup, OpenFL is also easy to use but the learning curve is quite steep even when using the most introductory tutorials. This is because some features do not work as expected from the start and the user sometimes needs to adapt the template code in order to run the tutorials. NvidiaFlare comes last in the ranking because the setup process is the hardest to follow and the encountered errors were solved by browsing the issues page of the GitHub repository which is a hard workaround to find. While setting up the simulation environment is relatively easy, using the NvidiaFlare Proof-of-Concept mode, which is a more realistic environment is harder to learn how to use and properly set up.

- **Quality of code documentation**

  1. Flower
  2. NvidiaFlare
  3. OpenFL

  In terms of code documentation, both Flower and NvidiaFlare excel, with Flower being slightly better with regard to the introductory tutorials. However, both offer great explanations for the main APIs as well as for the example experiments. The OpenFL documentation is good when considering the example applications, but the codebase contains TODOs as well as comments suggesting future fixes that need to be implemented, which leads to the fact the framework is not yet at the same maturity as the other 2.

- **Framework capabilites**

  Since framework capabilities refer to unique features each platform provides it is hard to create a ranking based on it. Considering this fact, the final flow diagram that helps each user decide what framework to choose should be considered.

  However, in terms of the most standout factors of each framework, we can clearly associate the frameworks with these features:

- Flower - easiest to start working with and adapt from regular Machine Learning workflows.
- Nvidia Flare - most inclined towards experienced users.
- OpenFL - best for security customization.

5.2 Performance ranking

In terms of performance, Nvidia Flare performed best for the CIFAR10 experiment, followed by Flower and OpenFl. In the case of OpenFL, the training stopped midway due to the machine running out of memory.

Figure 4 below shows all of the results from the performance benchmarking:

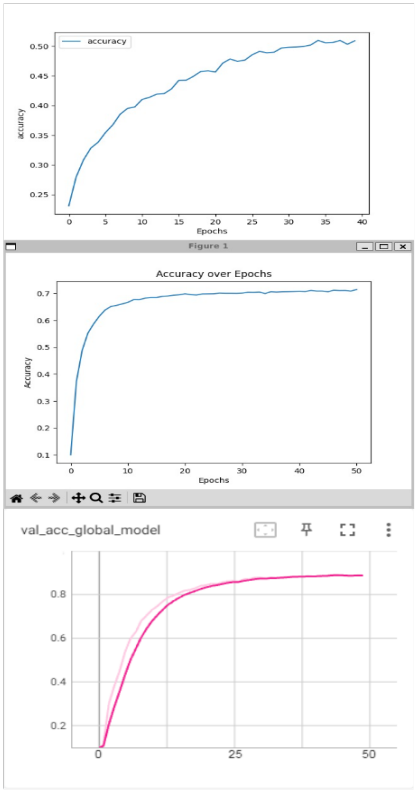

Figure 4: Experiments from top to bottom : OpenFL, Flower, Nvidia Flare

The diagram in Figure 5 (also present in the appendix, for ease of reading) can be used as a starting point for deciding what framework to use considering the bench-marking results and the comparison of each framework's strengths:

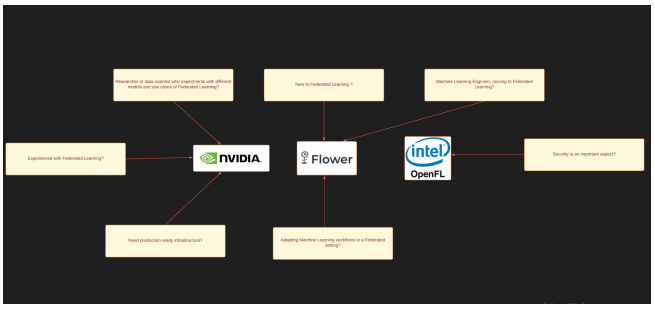

Figure 5: Choosing between the frameworks.

6 Responsible Research

The research conducted did not involve any ethical concerns. This is because most of the experiments that were run are based on the CIFAR10 dataset, a generic dataset that classifies images of different objects or animals. Also, the main focus of the research is based on reviewing the documentation and the code of federated learning platforms which did not introduce any ethical concerns.

7 Discussion and conclusion

In conclusion, Flower, Nvidia Flare, and OpenFL are all 3 viable Federated Learning frameworks that have their unique strengths and weaknesses. The overall goal of the research paper was to help different type of users decide between one of the 3 frameworks, which can be decided based on the results section. The main focus of the benchmarking was evaluating a couple of heuristics that can be derived from using the platforms. Also, a small experiment was replicated across all 3 platforms in order to have a brief idea about performance comparisons. The only limitation found was for the OpenFL experiment, where the same machine that was used for all experiments ran out of memory. This could be classified be a machine limitation, but considering that the nature of the experiment was not extremely computational intensive it might point out to some framework limitations as well.

For future work, the benchmarking can continue with a more thorough performance analysis, including more experiments. Then, additional frameworks could be added to the comparison in order to have a better overview of the Federated Learning tools available.

References

[1] Pyenv. https://github.com/pyenv/pyenv.

[2] Flower examples. https://github.com/adap/flower/tree/main/examples, 2020.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore,

Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[5] Conda. Conda installation guide. https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html.

[6] Sai Praneeth Karimireddy, Chaitanya Hegde, and Satyen Kale. SCAFFOLD: Stochastic controlled averaging for federated learning. *arXiv preprint arXiv:2102.03279*, 2021.

[7] Jakub Konečnỳ, H. Brendan McMahan, Peter Richtárik, Ananda Theertha Suresh, and Felix Xu. Federated optimization in heterogeneous networks. In *Advances in Neural Information Processing Systems*, pages 317–325, 2016.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[9] Xiaoyuan Liu, Tianneng Shi, Chulin Xie, Qinbin Li, Kangping Hu, Haoyu Kim, Xiaojun Xu, Bo Li, and Dawn Song. Unifed: A benchmark for federated learning frameworks, 2022.

[10] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[11] NVIDIA. NVIDIA Flare Documentation. https://developer.nvidia.com/flare-rhino, 2023.

[12] OpenFL Community. OpenFL Documentation. https://openfl.readthedocs.io/en/latest/index.html.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[14] Python Poetry. Python Poetry documentation.

[15] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Amr Alex, David Li, Virginia Lee, and Suvrit Sra. Adaptive federated optimization. In *International Conference on Machine Learning*, pages 8586–8597, 2021.

[16] Secure and Private AI Collaborative. Openfl. https://github.com/securefederatedai/openfl, Year of retrieval.

[17] The Jupyter Development Team. Jupyter notebook, 2015.

[18] Wikipedia. Public Key Infrastructure. https://en.wikipedia.org/wiki/Public_key_infrastructure.

[19] Wikipedia. Transport Layer Security. https://en.wikipedia.org/wiki/Transport_Layer_Security.

A Convolutional Neural Network code

The code below was used to create a Convolutional Neural network which was used for the 3 performance experiments :

```
model = Sequential()
model.add(Conv2D(32, kernel_size=3, padding='same', activation='relu', input_shape= (32,32,1)))
model.add(Conv2D(64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2, strides=2))
model.add(Conv2D(128, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(128, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2, strides=2))
model.add(Dropout(0.05))
model.add(Conv2D(256, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(256, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2, strides=2))
model.add(Flatten())
model.add(Dropout(0.1))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
return model
```

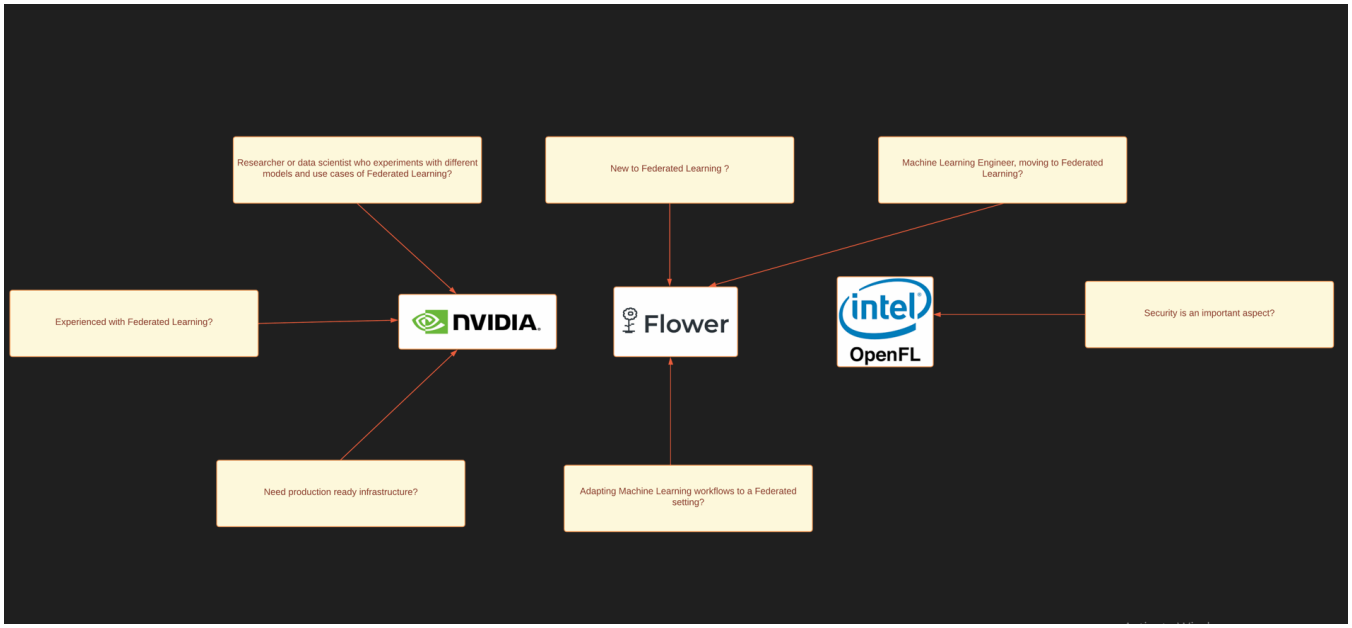

Figure 6: Choosing between the frameworks.