**OpenFL evaluation**

* **General information**

*Setting up the experiment*

There are two ways to set up FL experiment with OpenFL. First one is based on TaskRunner API where Aggregator and Collaborator is terminated at the end of the experiments. It also supports communication channels and TEE-based computing environments. The second way is Workflow API allowing for local experimentation. Those experiments can be easily scaled up to deployment phase.

*Flow*

Every workflow begins with the start task and ends with end task. Aggregator starts the process with the start task and extracts collaborators from the runtime. They will create a list of participants to run tasks and take the model, optimizer, and any other specified functions. Each definition of the task is marked by placement decorator in the code to distinguish between aggregator and collaborator. Once the given task is completed, each collaborator passes their state to the train task. Then it can be either locally validated or directly passed back to the aggregator, where all the results are joint and averaged. Once the whole loop is completed, the next training round can start.

A diagram of a model

AI-generated content may be incorrect.

* **How is privacy preserved?**

There are multiple ways in which privacy is secured. Firstly, it follows all principles of federated learning, so the data never leaves the client side. Secondly, it is possible to use certified authority: “*OpenFL supports mTLS, which ensures secure communication between the collaborators and the aggregator. This step generates a certificate authority (CA) that will be used to sign the certificates of the collaborators.*” Thirdly, we can generate certificates and private keys for aggregators. Those certificates have to be signed by already mentioned CA. There are also privacy measurements for collaborator, it requires certificate signing, generation and verification. Once the request of the collaborator is sent and signed, it has to go back to the collaborator.

* **Ease of use**

There are many tutorials explaining how to start with the package. After completing them, it is relatively easy to come up with your own model. Once you understand the structure and the flow of this approach, you can choose which components you want to replace and create your own modifications. Dependencies are easy to resolve most of the time.

* **What works well**
* There are many templates and tutorials provided.
* Each tutorial demonstrates how OpenFL can be used in specific situations (various models adopted) making it easy to adopt to your project.
* Easy to install (compatible many current new versions)
* Compatible with many other tools and packages.
* Easy to experiment with.
* The structure is easy to understand and concise: decorators are used to distinguish which part of code is executed by which party; moreover the .next() call makes the whole structure flow nicely.
* Each step is clearly separated from the others allowing for modifying the whole block without interrupting the whole structure
* In some notebooks there are specified what versions of packages were used for running them.
* Can be run manually or via Docker.
* **What could be improved**
  + Tutorials are good but it is not that well documented. User has to go through the code on their own and count on only on the explanations provided, while the code has little to no annotations.
  + Documentation could be improved; however, the simplicity of use compromises this downside.
  + Automating some parts would speed up the whole process (all of the steps in the flow have to be defined manually, would be nice if some more templets were available to fill in) – in contrast to Flower or FEDn.
  + In comparison to other packages, it is well developed but the functionality is not well demonstrated. Only once user starts to go deeper, they can discover the possibilities of this package, it is not exposed straight away.
* **Deployment**

OpenFL provides tutorial how to use it in combination with Docker. However, the provided instructions are very fuzzy, and user has to figure out on their own; also a very proficient level of Docker knowledge is assumed. After setting up a base image, there is a big gap in explanation on how to set up workspace image. There is only a reference to their help method.

The command that worked in my case: *fx workspace create --template 'vertical\_fl\_two\_party' --prefix template*

There is an option to run the scripts of TEE or without TEE. Besides setting aggregator and collaborators, there is not much explanation. There is an example of how to use it in production, but it seems very vague.

Conclusion: strong Docker basis is required to be able to use it since the official tutorial does not provide much explanation. Assuming you are experienced, deploying your projects is quick: there are only a few commands you need to run.

* **Additional functionality/implementations** 
  + Authors of the package provide an example of cyclic institution incremental learning. They demonstrate that depending on the case, it can either improve or worsen the performance of the model. OpenFL supports its implementation.
  + One of the issues that are hard to detect in FL is *backdoor attack*. Averaging algorithms such as FedAvg, blindly trust all the updates sent by clients, therefore, there is a risk that some malicious clients might try to alter the results performed by global model. OpenFL offers a solution for that type of attacks; an example can be found in their repository. It is called CrowdGuard. In the future, implementations requiring TEE is going to be published. The approached used:
    - Local validation – each client verifies the received model to see if it suspicious
    - Multi-layer clustering scheme performed by aggregator – clustering responses from clients; anomalous patterns might signify the attack
    - Suspicious models are pruned from the aggregation process; global model is updated only when no suspicious models are detected.

There are four files involved in the implementation of such a structure. The client has its own file where all the necessary components are added. Moreover, the experiment itself is also set up in a separate file. There are also two examples of attacks provided: intense one and less invasive. They test how efficient the set up structure is. The proposed framework can be reused in your projects, however some adaptation might be needed, depending on your application/model. Mostly, only the main file should be adapted, as well as the proposed NN structure. If you would like to plug it in to other files, it is best to keep it as a separate file implementing additional logic to the experiment.

Some of other common ways of dealing with this issue are:

* + - Reducing the influence of outlier updates;
    - Detecting poisoned updates using update patterns;
    - Adding differential privacy to reduce the influence of each client as an individual;
    - Track suspicious clients.
  + Example of fine-tuning Phi-4 model
    - A lot of preparations and pre-processing has to be done before starting anything (it goes for every complex model)
    - Those files can be used as an example how to approach tasks where we have text data. We only need to adapt the model and tokenizer to our needs.
* **Requirements**

Python 3.10 and above is required to use OpenFL (supported until at least April 2025). Can be used via Docker but pure Python commands are enough. For deployment, Docker required. Docker images are supported only on linux/x86\_64.

* **Sources**

Official website: <https://openfl.io>

Official repository: <https://github.com/securefederatedai/openfl/tree/develop>

How to use Opacus with OpenFL: <https://github.com/securefederatedai/openfl/tree/develop/openfl-tutorials/experimental/workflow/Global_DP>

How to start with Azure and OpenFL: <https://www.intel.com/content/www/us/en/developer/articles/technical/try-federated-learning-with-openfl.html>

Overview of a task runner: <https://openfl.readthedocs.io/en/latest/about/features_index/taskrunner.html>

Interactive API: <https://openfl.readthedocs.io/en/v1.6/about/features_index/interactive.html>

Documentation: <https://openfl.readthedocs.io/en/latest/openfl.html>