This project has received funding from the European Union's Horizon 2020 - Research and Innovation Framework Programme, H2020 SU-FCT-2019, under grant agreement no 883341.

Global Response Against Child Exploitation

Instrument:	Research and Innovation Action proposal
Thematic Priority:	FCT-02-2019

Federated Learning Strategies

Deliverable number	D5.3	
Version:	1.1	
Delivery date:	May 2022	
Dissemination level:	PU	
Classification level:	Non classified	
Status	Final	
Nature:	Other	
Main author(s):	Athanasios Psaltis Kassiani Zafeirouli	CERTH
Contributor(s):	Stavroula Bourou, Ioannis Kontopidis	SYN VICOM

DOCUMENT CONTROL

Version	Date	Author(s)	Change(s)
0.1	08/04/2022	Athanasios Psaltis, Cassie Zafeirouli (CERTH)	TOC
0.2	03/05/2022	Cassie Zafeirouli	Section 2 added
0.3	09/05/2022	Athanasios Psaltis	Section 3 added
0.4	18/05/2022	Cassie Zafeirouli Stavroula Bourou (SYN) Ioannis Kontopidis (VICOM)	Partners Input
0.5	20/05/2022	Cassie Zafeirouli	Section 4 added
0.6	30/05/2022	Athanasios Psaltis, Cassie Zafeirouli	1st version for SAB & peer review
0.7	06/06/2022	Salvatore Vicari (ENG)	Peer review
0.8	07/06/2022	Athanasios Psaltis	Integration of review suggestions
0.9	08/06/2022	Philip Engström	Approved by SAB
1.0	09/06/2022	Athanasios Psaltis, Cassie Zafeirouli	Final proof reading
1.1	15/03/2024	Peter Leskovsky	Correcting the dissemination level to PU

DISCLAIMER

Table of Contents

Tables

Figures

Executive Summary

This deliverable presents the results of Task 5.3 Federated Learning Strategies. Within this task a set of criteria/requirements related to the Federated Learning paradigm have been defined, taking into account the established network topology, type of participants, data distribution and privacy, and security needs as reported in D5.1 and D5.4. In addition, the most functional and commonly used frameworks have been investigated to identify their advantages and limitations concerning the project's needs and requirements. Furthermore, a tool for experimentation, reproducibility, deployment and a central model registry has been integrated. Moreover, this task included a thorough study of the Federated Learning aggregation strategies and more specifically the optimization algorithms that take place on both the server and client-side during the training process. In this deliverable, we carry out a complete convergence analysis to evaluate some of the broader aggregation strategies in the selected Federated Learning framework. This task also included the hyper tuning of several parameters during the model exchange process, the local training as well as the optimization algorithm. Validation tests have been also executed to evaluate multiple Federated Learning strategies under different scenarios for different tasks using different modalities in both the IID and Non-IID settings.

1. Introduction

1.1. Overview

This deliverable is reporting the output of the Task 5.3 - Federated Learning Strategies that is described in the Description of Action (DoA) as:

"The objective of T5.3 is to design, develop and implement the strategies of DL model building under the Federated Learning paradigm, including the parameters of model exchange among nodes, local training, model updates based upon secure aggregation and decision on the weighted average. Blending of online/offline DL processing and adaptation of the existing lambda architecture are handled by this task, as well as the selection of DL categories and algorithms, including supervised learning, semi-supervised and unsupervised learning. The selection of technologies and processes for FL takes place in this task."

The main objectives of this document are the following:

- To report and comprehensively describe general GRACE's requirements related to the Federated Learning paradigm focusing on the network topology, type of participants, data distribution and privacy, security needs
- To investigate the available open-source Federated Learning frameworks and libraries to select the more suitable one for the GRACE Federated Learning ecosystem
- To design and develop the GRACE FL system based on the selected framework
- To research, implement and evaluate multiple Federated Learning strategies under different Federated Learning scenarios for different tasks.

Performance evaluation will be completed in the last task of WP5 Task 5.5 Federated Learning System Analysis, which will include an overall assessment of the GRACE Federated Learning system.

1.2. Relation to other deliverables

This deliverable is related to the following other GRACE deliverables:

Receives inputs from:

Deliv. #	Deliverable title	How the two deliverables are related
D2.1, D2.2	Use Cases, Process and Data Flows Refinement	D2.1 should be considered in order to design the FL system that will accommodate the defined use cases, processes and data flows.
D2.4, D2.5	User Requirements	D2.4 should be considered to ensure that the designed FL architecture and the proposed FL strategies will satisfy the user requirements.
D2.10, D2.11	Technical Specifications and Architecture	The architecture considered in D2.10 should be considered in the GRACE FL ecosystem.
D2.14, D2.15	Security and auditing mechanisms report	The mechanisms defined in D2.14 must be integrated in the FL architecture in an efficient but definitive way.
D5.1	Federated Learning Infrastructure	The selected FL framework and the proposed FL

	and processes	strategies should be aligned with the GRACE FL architecture and the system topology.
D5.2	Federated data annotation	The design and development of the tools for annotation of Federated data of D5.2 must be taken into account to ensure that the specifications are met.
D5.4	Secure data exchange mechanism	The design and implementation of the Federated Learning privacy-preserving mechanisms described in D5.4 should be considered for the development of the GRACE FL strategies and FL ecosystem. The creation of D5.3 and D5.4 are performed in parallel and in close collaboration.

Table 1: Relation to other deliverables – receives inputs from

Provides outputs to:

Deliv. #	Deliverable title	How the two deliverables are related
D5.5	Federated Learning system analysis	The provided FL framework and the proposed FL strategies will be used in the analysis of the Federated Learning system.
WP3 deliverables	Data Acquisition and Handling	The GRACE FL framework interacts with the components of WP3, and specifically with the Unified Data Lake that is on the premises of each LEA and EUROPOL's.
WP4 deliverables	visual, audio and text processing	D5.3 will provide guidelines to WP4 tools, related to the selected FL framework and the proposed FL strategies.
WP8 deliverables	Pilots Definition, Preparation, Planning, Execution and Evaluation	The FL framework and strategies described in D5.3 shall be integrated in the WP8 context.
WP9 deliverables	Social, Ethical, Legal and Data Protection Framework	The requirements for the GRACE FL ecosystem are prepared considering the ethical aspects, coordinated with the business needs to have a performant GRACE platform.

Table 2: Relation to other deliverables – provides outputs to

1.3. Structure of the deliverable

The document includes the following sections:

- Section 1: An overview of this document is provided, setting the objectives of the deliverable and highlighting the dependencies among the current and other deliverables.
- Section 2: Reporting of the GRACE specific requirements and needs related to Federated Learning paradigm and comprehensive description and evaluation of the existing Federated Learning frameworks.
- Section 3: Analysis of multiple state-of-the-art Federated Learning algorithms including FedSGD,

FedAvg, FedProx, FedOpt methods.

- Section 4: Performance evaluation methodology description and results analysis for different modalities and FL parameters.
- Section 5: Document summarization and future work discussion.

2. Federated Learning Frameworks

Standard machine learning approaches require aggregating data from several edge devices, like mobiles, IoTs, servers and centralizing them on one central datacenter or on a cloud for model training and evaluation. This approach can achieve excellent results with respect to model's performance but raises security concerns related to privacy violation and personal data leakage and requires large software and hardware resources due to transmission and storage of high-volume, high-velocity and high-variety data.

Federated Learning (FL) is a new era in AI that aims to handle the limitations of centralized ML, providing a collaborative, fully decentralized ML approach by exploiting both distributed data and distributed resources to build accurate, robust models in a privacy-preserving manner. Unlike centralized ML approaches that gather distributed local data to a central datacenter, FL solution transfer only the local-trained models, without data exchange, to a centrally located server to build the shared global model. FL inherently ensures privacy and security as the data resides in owner's premises and never accessed or processed by other parties.

Fig. 1 presents in more detail how FL works in practise, within the GRACE project, and how orchestrates the ML training process between different entities, in our case EUROPOL and MS LEAs. GRACE's FL architecture follows a star-like topology, where every local node (MS LEAs' devices) connects to a central device (EUROPOL's server) and establishes a one-to-one communication. Once the communication has been established, the distributed training process starts. Firstly, the server distributes the initial version of the model to each node for training on local data. This first version can either be randomly initialized or pre-trained on a predefined dataset. Afterwards, the updated local model is sent back to the central server to be averaged with other nodes' updates, utilizing secure aggregation methods. In the final step, the updated global model is forwarded to the local nodes for another round of training. The process is continuing until the global aggregated model is fully trained and achieves the required performance (see D5.1).

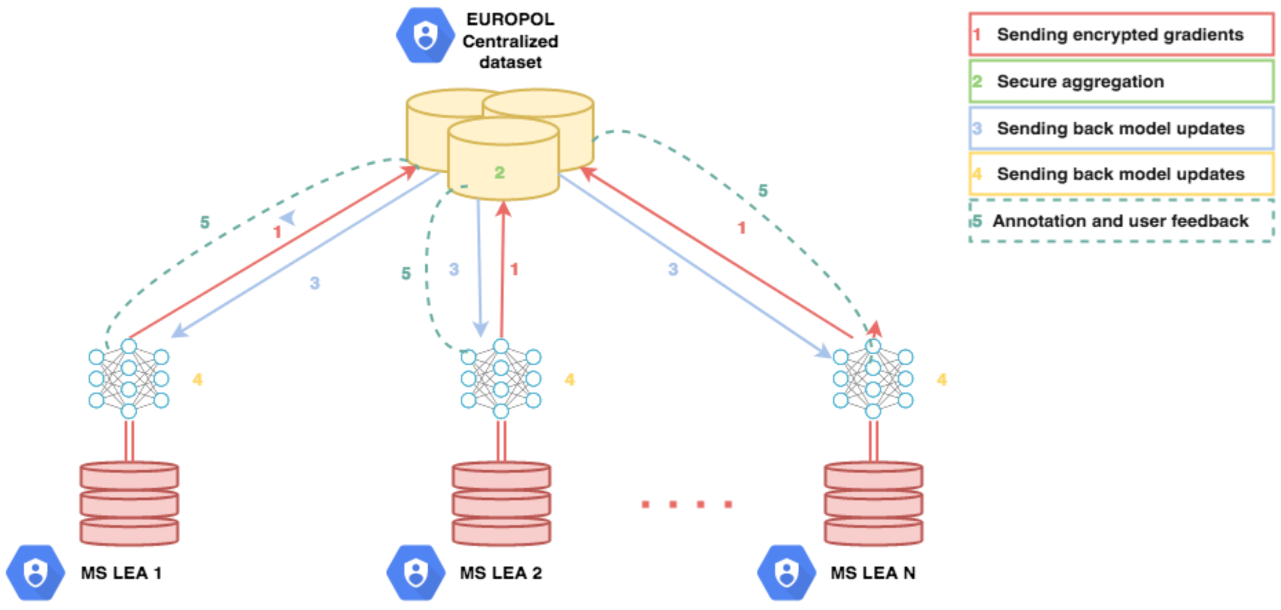

EUROPOL Centralized dataset
1 Sending encrypted gradients
2 Secure aggregation
3 Sending back model updates
4 Sending back model updates
5 Annotation and user feedback
MS LEA 1
MS LEA 2
MS LEA N

Figure 1: The adopted Star-like topology. GRACE utilizes a centralized server (EUROPOL) to create and share the global model, while several MS LEA nodes participate asynchronously in the training process.

FL solution provides the methodological framework to collaboratively train a single global ML model by only sharing the parameters of the models separately trained on local nodes. Many open-source libraries and frameworks have been developed to connect the FL principals with existing ML frameworks to enable the implementation of FL – ML scenarios. These frameworks provide the basic fabric for the FL training process, combined with advanced features including multiple learning algorithms, various security and privacy mechanisms and different topologies. Since FL is a new concept, introduced in 2017 by Google[1], the majority of the existing FL frameworks are still under development. In the context of GRACE, the most functional and commonly used frameworks have been investigated to identify their advantages and limitations with respect to project's needs and requirements.

2.1. General Requirements

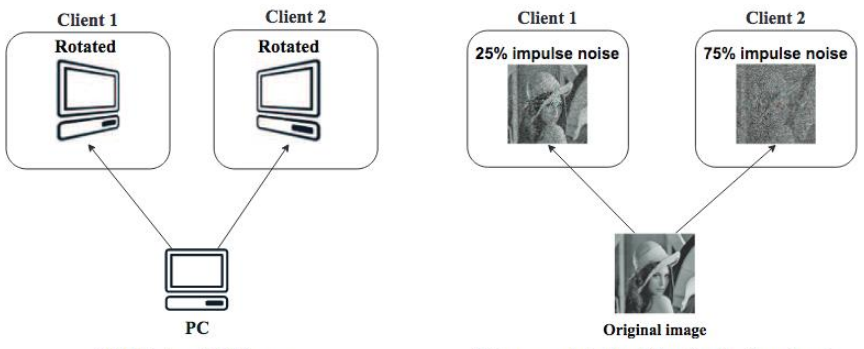
Different FL frameworks provide different capabilities related to type of infrastructure topology, type of clients, learning algorithms, and security mechanisms, covering different needs. GRACE has specific requirements that should be taken into consideration for the selection of the most suitable FL framework.

As mentioned above and comprehensively described in D5.1, GRACE network architecture follows a start-like topology for cross-silo FL, where, in this case, silos are the EUROPOL and few MS LEAs that each represent a large repository of data. The main challenge in cross-silo FL is the heterogeneity of the data distribution among parties. The private local databases are usually non-independently and identically distributed (non-IID) and they may be different in size, and show differnet label and feature distributions. For example, different MS LEAs can have very different investigation-related objects distribution depending on their country (e.g., different types of power sockets or cans/bottles). Global collaboration without considering individual private data specifications could lead to low model performance and poor generalization. Therefore, the selected FL framework should provide multiple FL learning algorithms to address the learning effectiveness under non-IID data settings, while also supporting the development of custom algorithms to cover specific needs.

The main categories for non-IID data can be summarized as follows (see Figure 3 to Figure 7):

- Covariate shift: local nodes may store examples that have different statistical distributions compared to other nodes.
- Prior probability shift: local nodes may store labels that have different statistical distributions compared to other nodes. This can happen if datasets are regional and/or demographically partitioned.
- Concept drift (same label, different features): local nodes may share the same labels but some of them correspond to different features at different local nodes.
- Concept shift (same features, different labels): local nodes may share the same features but some of them correspond to different labels at different local nodes.
- Unbalancedness: the amount of data available at the local nodes may vary significantly in size.

[1] https://ai.googleblog.com/2017/04/federated-learning-collaborative.html



(a) Multi-view of a PC image. (b) Images perturbed by different levels of impulse noise.

Figure 2: Two examples of attribute distribution skew

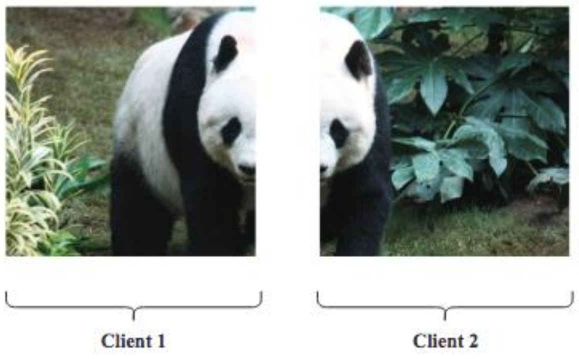
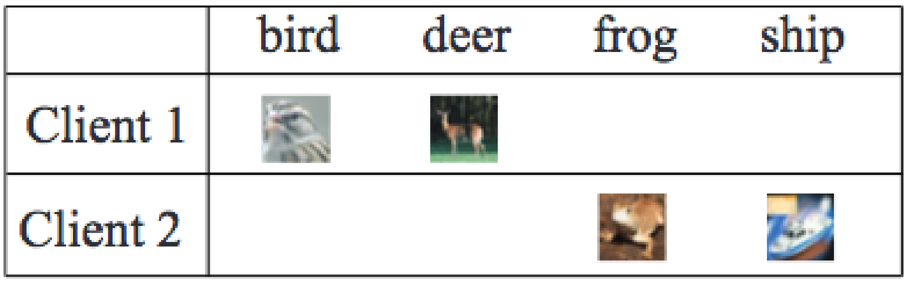


Figure 3: An example of non-overlapping attribute skew for image datasets, the left half of the image data is stored on Client 1, while the right half of the image is on Client 2



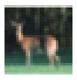
	bird	deer	frog	ship
Client 1				
Client 2				

Figure 4: An illustrative example of label size imbalance with two clients and class_number = 2 using CIFAR10 as an example

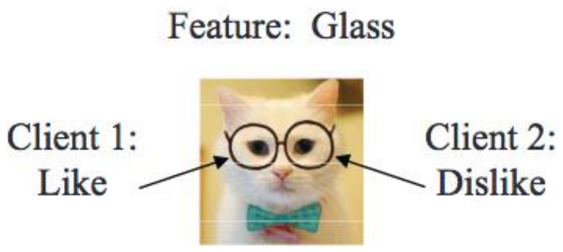


Figure 5: An example of label preference skew. For the same input feature (a cat with glass), Client 1 may label 'like' while Client 2 may label 'dislike' instead.

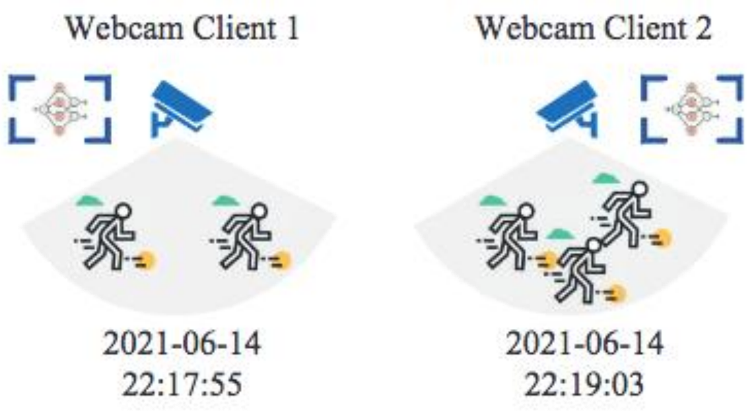


Figure 6: An example of temporal skew for webcam recordings in FL

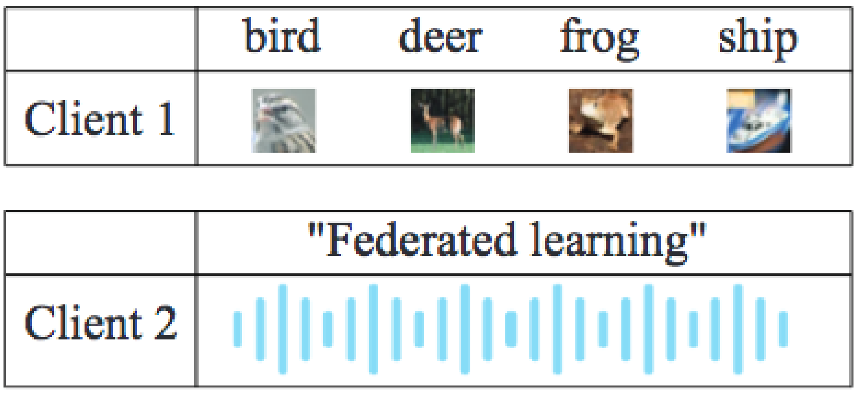


Figure 7: An example of different features, different labels Non-IID. Client 1 and Client 2 may hold different types of data (image and audio).

Another significant parameter that should be considered is the data protection and the alignment with the GDPR regulation and restrictions[2] during the FL process. The natural advantage of FL is the ability to reassure data privacy because sensitive data are stored locally and only the model's parameters are exchanged. However, without any other privacy-preserving and security mechanism, sensitive information can leak from local training parameters. GRACE handles very sensitive data related to child exploitation, which must not be revealed, stolen or reconstructed, and therefore a holistic secure FL environment is developed, enhanced by additional privacy and security solutions to mitigate any risk.

Most of the existing FL frameworks are not yet fully functional and fully developed, and they cannot support large scale real-world distributed training scenarios. However, GRACE wants to exploit both the distributed data and the distributed resources of multiple entities in a realistic context. Therefore, GRACE FL approach will enable the establishment of a secure connection and communication channel between multiple external clients, the usage of the distributed resources including both CPUs and GPUs and the development of a data handling tool to unify clients' management system to ease the training process.

In the next subsections, the FL frameworks that have been investigated in the context of GRACE project are presented, to report their advantages and their limitations with respect to project's requirements and needs.

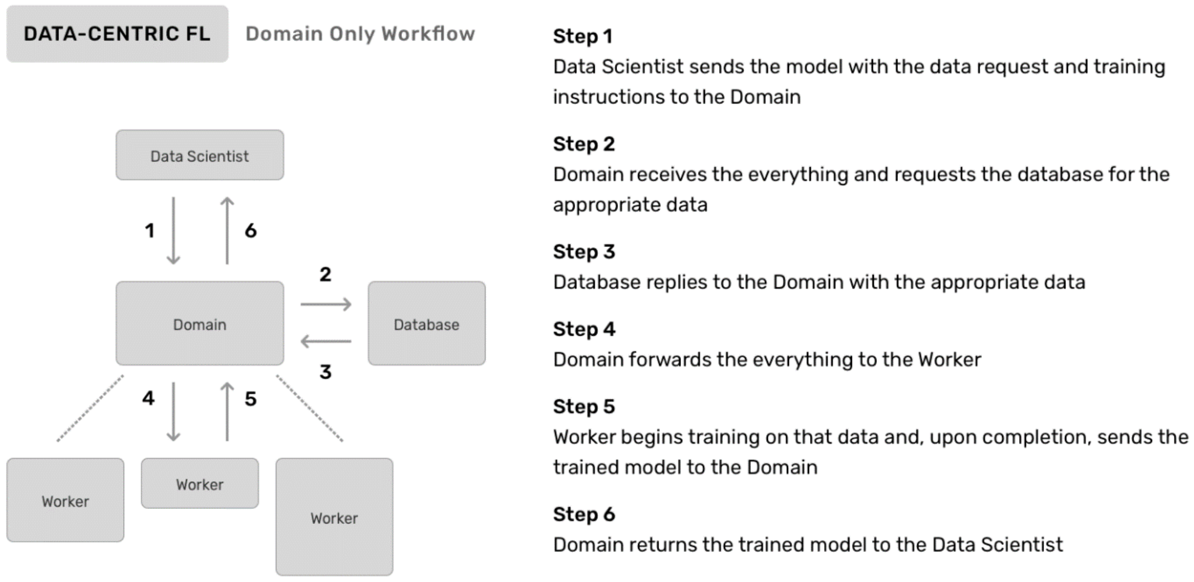
2.2. PySyft

[2] http://data.europa.eu/eli/reg/2016/679/oj

PySyft is an open-source numPy-based library, introduced by OpenMinded[3], that enables secure and private ML by extending frameworks like Pytorch and Tensorflow to adopt the FL principals in a transparent, lightweight manner. PySyft is supported by ecosystems and command line tools such as PyGrid and HAGrid to perform data science on data hosted in distributed resources.

PySyft library provides the basic functions and methods to perform FL, including aggregation algorithms and privacy and security mechanisms. However, PySyft, as standalone tool, can only be used for proof-of-concept FL scenarios by utilizing "virtual" nodes hosted in the same machine, and therefore it is not suitable for GRACE FL approach. To make it possible to exploit PySyft for real-world FL scenarios, it should be combined with PyGrid, a server for private data. PyGrid is designed to host private data in order to allow data scientists and tools developers to exploit distributed data without being able to process or download them.

Fig. 8 describes in more detail the whole FL process utilizing the Syft ecosystem. Firstly, the data owner (EUROPOL, MS LEAs) should deploy a domain node, a server to load the private data. Once the data have been successfully loaded, a data scientist can request access to the domain to perform remote training leveraging the existing data. The data owner can control how much a user can change and tinker with domain node by adding permissions related to data requests, domain settings etc. After the local training is completed, the model is sent back to data scientist to be aggregated with the other updated distributed models.



Step 1
Data Scientist sends the model with the data request and training instructions to the Domain

Step 2
Domain receives the everything and requests the database for the appropriate data

Step 3
Database replies to the Domain with the appropriate data

Step 4
Domain forwards the everything to the Worker

Step 5
Worker begins training on that data and, upon completion, sends the trained model to the Domain

Step 6
Domain returns the trained model to the Data Scientist

Figure 8: Syft ecosystem workflow

Although Syft is a framework that provides a secure environment for multiple entities to be connected and train ML models collaboratively, it has some limitations that make it unsuitable for GRACE's FL approach. A main disadvantage is that Syft supports only data in NumPy format and does not allow data owners to load to their domain other types of data. This is very restrictive for GRACE as it handles multiple types of data including image, audio, and text. Moreover, at the time of writing, Syft does not support the usage of GPUs in distributed machines and therefore the training of large models with extensive datasets will be time and resources consuming.

[3] Ziller, Alexander, et al. "Pysyft: A library for easy federated learning." *Federated Learning Systems*. Springer, Cham, 2021. 111-139.

2.3. Tensoflow Federated (TFF)

TFF[4] is a federated learning framework, which is extensively used for federated learning research by simulating federated computations on realistic datasets. TFF offers two APIs, which are the federated learning and the federated core. The federated learning API contains a set of high-level interfaces, while the federated core API provides a low-level interface with the ability to customise federated methods. Fig. 9 presents the architecture overview of TFF.

As aggregation mechanisms, the TTF implements FedAvg and Federated Stochastic Gradient Descent (FedSGD) algorithms. Specifically, the FedAvg utilises three main aggregation operators:

- Sum, in which the clients' values are summed and the results are published at the central server
- Mean, in which the weighted mean of clients' values is calculated and the result are published at the central server
- Differentially private, in which a Gaussian or Laplacian noise is computed based on the clients' values and it is added to the data. Then the results are published at the central server

Regarding the security and privacy aspects, a secure aggregation protocol can be built into TensorFlow Federated, based on encryption properties from TF Encryption. Additionally, the TFF framework contains TensorFlow privacy, a python library for applying privacy techniques on machine learning model training.

However, the TFF has some drawbacks. Firstly, this FL framework can only be used for simulation mode, which means that it cannot run a real-life experiment or perform commercial-like analysis. Secondly, vertical and hybrid data splitting is not supported. Lastly, TFF does not support homomorphic encryption nor multi-party computation as privacy preserving mechanisms.

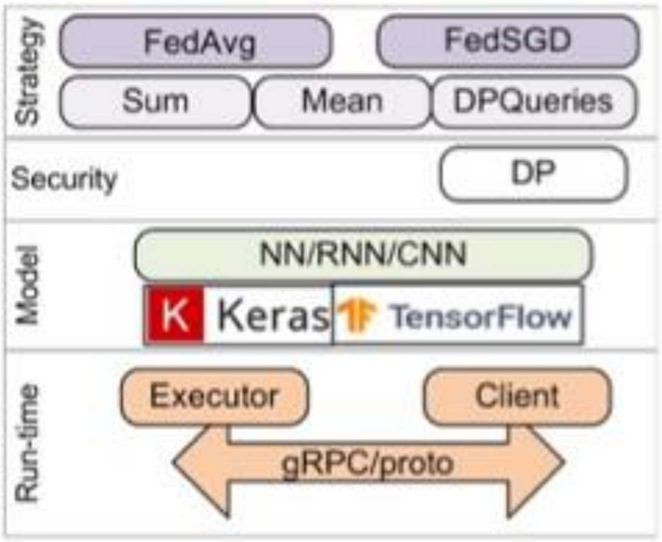


Figure 9: Architecture Overview of TensorFlow Federated (TFF)

2.4. NVIDIA Federated Learning Application Runtime Environment (NVFLARE)

[4] https://www.tensorflow.org/federated

NVFLARE[5] is a domain-agnostic, open-source FL platform, developed by NVIDIA, used by researchers and data scientists to adapt existing ML/DL workflow (Pytorch, Tensorflow) to a federated paradigm and enables developers to build a secure, privacy-preserving offering for a distributed multi-party collaboration.

As shown in Fig. 10, NVFLARE provides an extensive set of tools and features to build a custom FL ecosystem. Extensible management tools enable secure provisioning using SSL certifications, orchestration through an admin console, and monitoring of FL process with multiple visualization methods. Built-in workflow strategies can be used for training and evaluation, combined with learning algorithms for model aggregation. Moreover, NVFLARE provides privacy-preserving algorithms that guarantee the protection of sensitive information and prevent reverse-engineering of ML models.

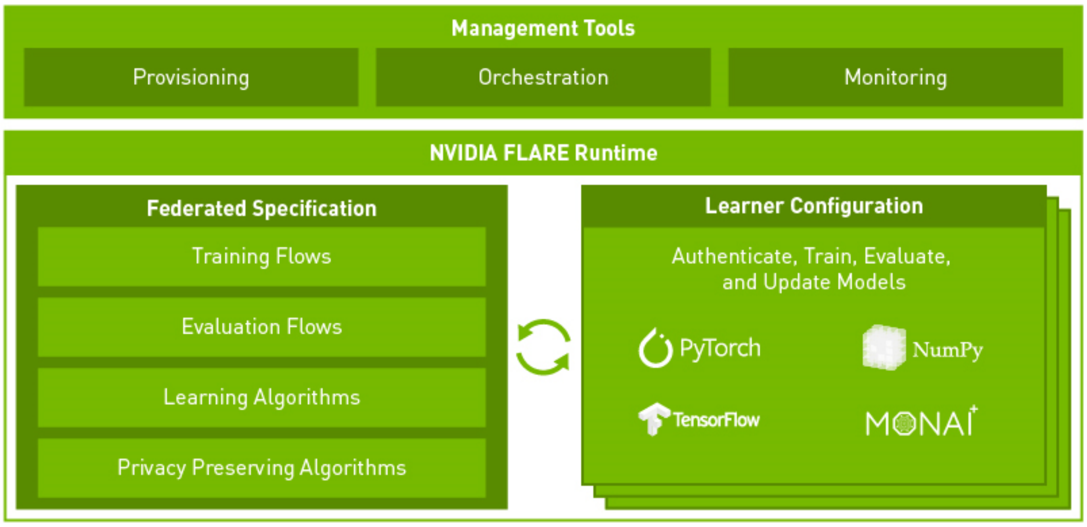


Figure 10: NVIDIA Federated Learning Platform

In more technical details, the purpose of provisioning in NVFLARE is to generate mutual-trusted system-wide configurations for all participants, thus allowing them to join the FL process across different locations. The configuration files usually include information related to network discoveries (domain names, IP address, ports) and credentials for authentication (certificates of participants). The data scientists/tool developers, referred as administrators in NVFLARE, who wants to collaboratively train a model, should create these files to fit their own requirements, and distribute them to participants (EUROPOL, MS LEAs) to establish a secure connection between them. Once the connection has been established the administrator is responsible for the orchestration of the FL process, by starting, aborting, or restarting the distributed training, and more.

Fig. 11 presents the training workflow that GRACE follows within the NVFLARE environment, in high and low-level analysis, with one Server (EUROPOL) assigning tasks to Clients (MS LEAs) and aggregating the produced results. The Controller is a function that controls or coordinates the Workers to get a job done. The controller runs on the FL server and Workers run on FL clients. The relationship between these components is shown in the diagram below, where the server controller defines task assignments (local training, model evaluation) that are broadcasted and executed on the client worker. The results of the client task execution are then returned to the server for aggregation. Both task assignment and task result submission can be applied on both the server and client side, together with filtering and including filters related to privacy and security mechanisms.

[5] https://developer.nvidia.com/flare

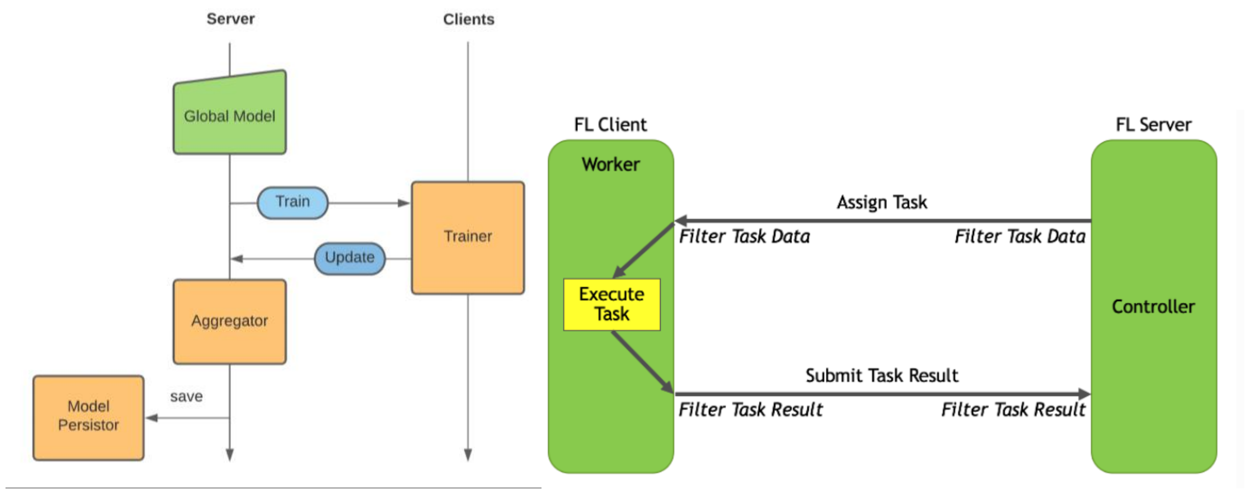


Figure 11: High-level FL training workflow (left), Low-level FL training analysis withing NVFLARE (right)

NVFLARE framework has been selected for the GRACE FL platform, since it covers all project's requirements. NVFLARE is a domain-agnostic ecosystem capable of handling any type of data and any type of ML/DL models. It is suitable for large-scale real-world FL scenarios by providing advanced provisioning and orchestration tools to set-up and coordinate an FL project in a user-friendly manner. It also supports GPU-based training on the distributed machines for time-efficient training and evaluation of large models and datasets.

3. Federated Learning Strategies

Since the term federated learning was initially introduced with an emphasis on mobile and edge device applications, interest in applying FL to other applications has greatly increased, including some which might involve only a small number of relatively reliable clients. GRACE project is a typical example of such application where multiple organizations need to acquire knowledge through the analysis of their data, but cannot share their data directly. In this case a group of LEAs collaborating to train an ML model based on the whole set of their data.

3.1. Data Partitioning

Based on data distribution over the sample and feature spaces, Federated Learning Strategies (FLS) can be typically categorized in horizontal, vertical, and hybrid schemes[6]. In short, Horizontal federated learning uses datasets with the same feature space across all devices, Vertical federated learning uses different datasets of different feature space to jointly train a global model, while Hybrid federated learning is a combination of the first two (in terms of feature and sample distribution).

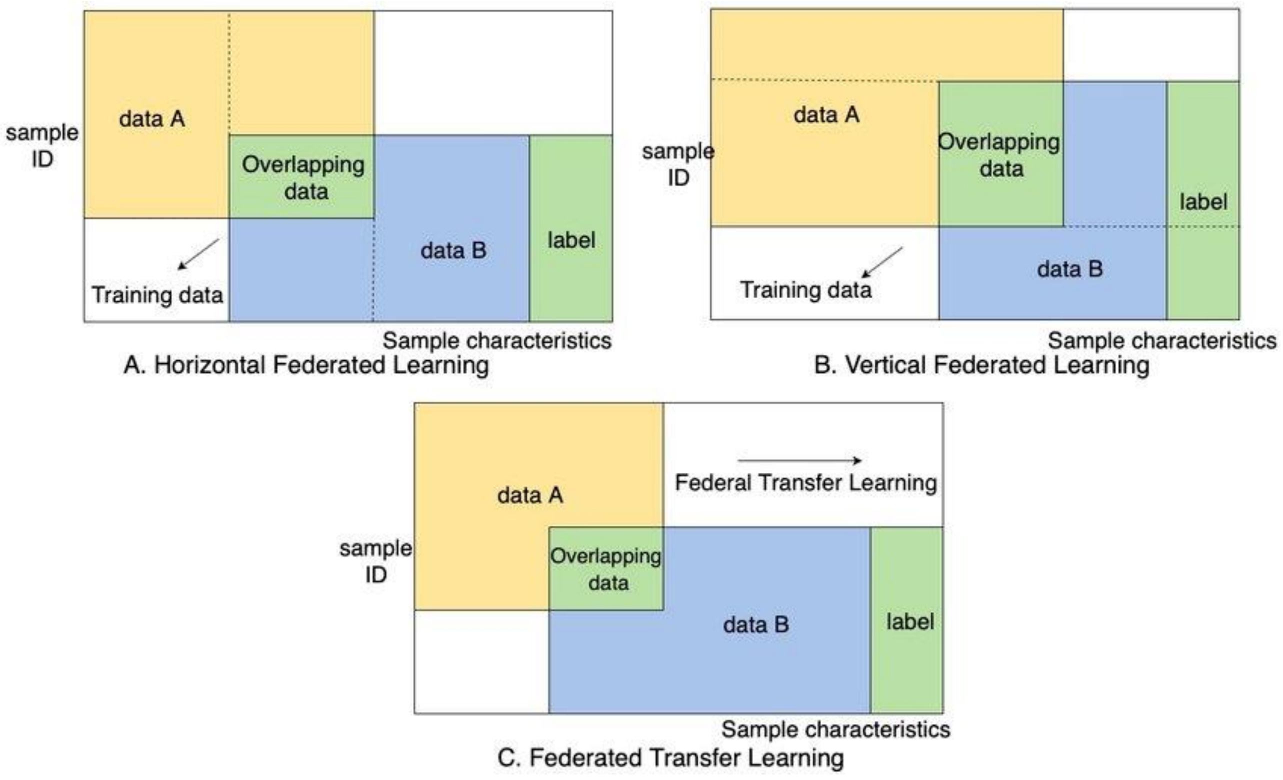


Figure 12: Federated Learning schemes based on data distribution.

Horizontal FL: In this scenario (Fig. 12A), the datasets of different parties have the same feature space but little intersection on the sample space. This is a natural data partitioning especially for the cross-device setting,

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

where different users try to improve their model performance on the same task using FL. Also, the majority of FL studies adopt horizontal partitioning (i.e. partitioning by example). Since the local data is in the same feature space, the parties can train the local models using their local data with the same model architecture. The global model can simply be updated by averaging all the local models. A basic and popular framework of horizontal federated learning is FedAvg[7]. Partitioning by examples is usually relevant for cross-silo FL approaches (e.g., GRACE FL architecture), when a single company cannot centralize their data due to legal constraints, or when organizations with similar objectives want to collaboratively improve their models. An example of horizontal partitioning is shown in Fig. 13.

Vertical FL: In this setup (Fig. 12B), the datasets of different parties have the same or similar sample space but differ in the feature space. The majority of literature approaches that lay in this category usually adopt entity alignment techniques[8] to collect the overlapped samples of the parties. Then the overlapped data are used to train the ML model using encryption methods. In a recent study, Cheng et al.[9] propose a vertical approach to enable parties to collaboratively train ML models, by utilizing privacy-preserving entity alignment to find common users among two parties, whose gradients are used to jointly train the decision trees. It is important to note that, cooperation among different companies usually can be treated as a situation of vertical partition. An example of vertival partitioning is shown in Fig. 14.

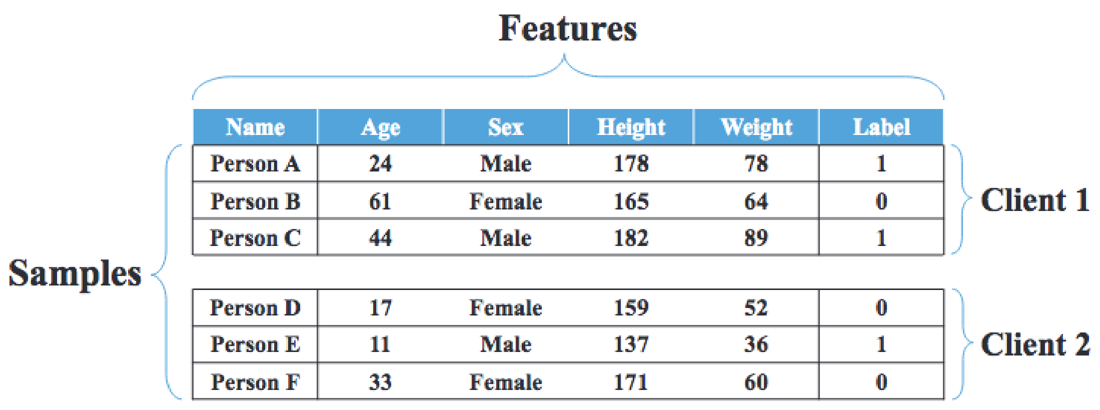
Hybrid FL: The majority of the existing schemes are mainly covered by one of the above two categories, however, the partition of data among specific parties may be a hybrid of horizontal partition and vertical partition. In this case (Fig. 12C), more advanced concepts have been proposed to consider the challenging scenarios in which data parties share only a partial overlap in the user space or the feature space, and leverage existing traditional ML transfer learning techniques to build models collaboratively. Federated transfer learning can be regarded as vertical federated learning adopting a pre-trained model that is trained on a similar dataset for solving a different problem. Liu et al.[10] propose a secure federated transfer learning system which can learn a representation among the features of parties using common instances. However, the existing formulation is limited to the case of 2 clients and cannot be applied to the GRACE project.

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

[8] Zhuang Yan, Li Guoliang, and Feng Jianhua. A survey on entity alignment of knowledge base. Journal of Computer Research and Development, 1:165–192, 2016.

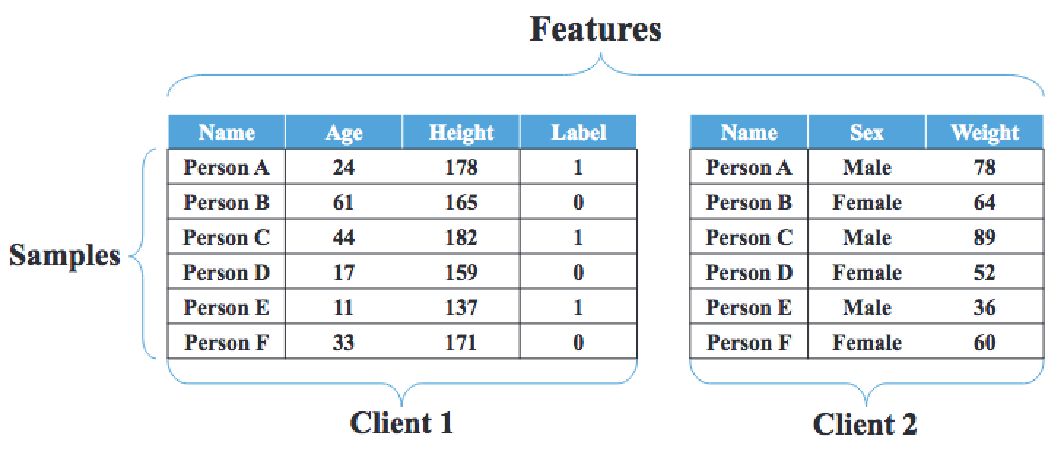
[9] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. arXiv preprint arXiv:1901.08755, 2019.

[10] Yang Liu, Tianjian Chen, and Qiang Yang. Secure federated transfer learning. arXiv preprint arXiv:1812.03337, 2018.


Features
Samples
Client 1
Client 2

Name	Age	Sex	Height	Weight	Label
Person A	24	Male	178	78	1
Person B	61	Female	165	64	0
Person C	44	Male	182	89	1
Person D	17	Female	159	52	0
Person E	11	Male	137	36	1
Person F	33	Female	171	60	0

Figure 13: An example of Horizontally partitioned data: data frames are partitioned horizontally into rows, each of which having the same features


Features
Samples
Client 1
Client 2

Name	Age	Height	Label
Person A	24	178	1
Person B	61	165	0
Person C	44	182	1
Person D	17	159	0
Person E	11	137	1
Person F	33	171	0

Name	Sex	Weight
Person A	Male	78
Person B	Female	64
Person C	Male	89
Person D	Female	52
Person E	Male	36
Person F	Female	60

Figure 14: An example of Vertically partitioned data: partition data frames into columns, with each column holding the same feature.

3.2. Strategies for Dealing with Federated Systems and Data

Compared with distributed learning, federated learning has some unique attributes:(a) The communication of federated learning is relatively slow and unstable. (b) Participants in federated learning have heterogeneous devices, and different devices have different computing capabilities. (c) Federated learning pays more attention to privacy and security. At present, most studies assume that the participants and the server are trustworthy. However, in real life, they may be untrustworthy. In implementing federated learning, it is necessary to consider how to optimize the federated learning algorithm to solve the existing practical problems. In terms of optimization, the major issues currently faced by researchers are: high **communication** cost, **statistical** and **structural** **heterogeneity** (Fig. 15).

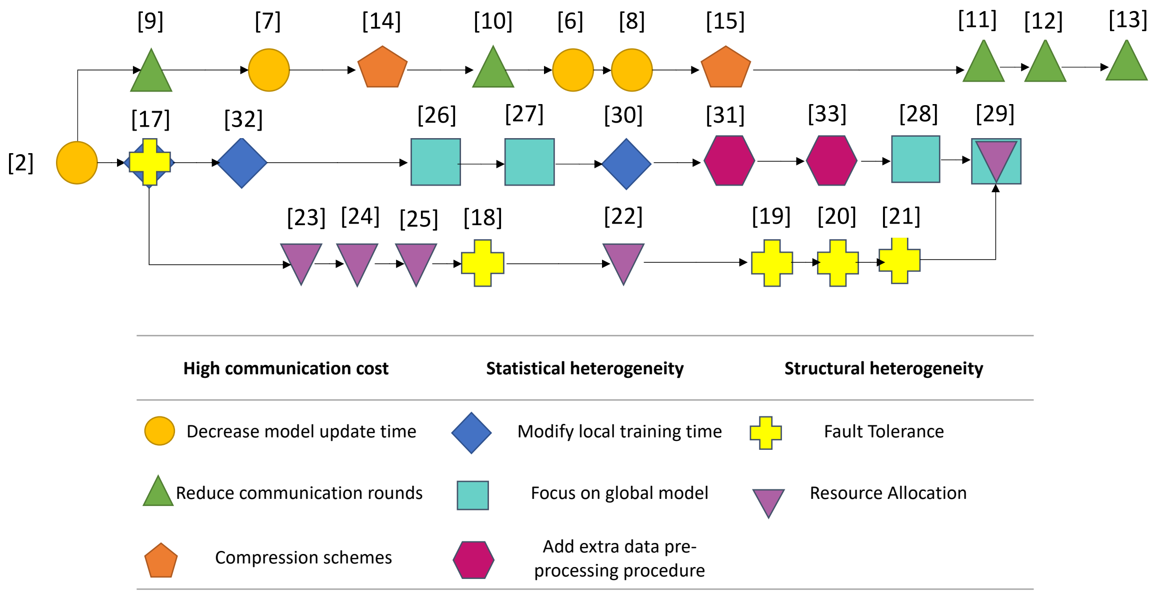


Figure 15: The first branch denotes the studies to deal with high communication costs. The second one represents the evolution of overcoming the challenge of statistical heterogeneity, while the third denotes structural heterogeneity. In the same branch, different symbols represent different ways to tackle the issue.

Communication overhead is a key bottleneck to consider when developing methods for federated networks. While it is beyond the scope of this deliverable to provide a self-contained review of communication-efficient distributed learning methods, we point out several general directions, which can be roughly divided into (a) local updating methods, and (b) compression schemes. To tackle large masses of data and make FL flexible against the explosive increasing of datasets size, the reduction of the communication overhead should be a top priority. Meanwhile, effective efforts have been made including the reduction of communication rounds and the improvement of model upload speed, to further reduce the update time.

In a typical machine learning system, an optimization algorithm like Stochastic Gradient Descent (SGD) runs on a large dataset partitioned homogeneously across servers in the cloud. Such highly iterative algorithms require low-latency, high-throughput connections to the training data. But in the Federated Learning setting, the data is distributed across several devices in a highly uneven fashion. Performing local updates and communicating less frequently with the central server addresses the main core challenges: complying with data locality constraints and considering the limited communication capabilities of edge device clients.

Several recent methods have been proposed to improve communication-efficiency. Communication between server and clients is willing to be as little as possible to reduce upload time. This is achieved by allowing, in distributed settings, for a variable number of local updates to be applied on each machine in parallel at each communication round, thus making the amount of computation versus communication substantially more flexible. These methods drastically improve performance in practice, and have been shown to achieve significant speedups over traditional distributed approaches. In federated settings, optimization methods that allow for flexible local updating and low client participation have become widely accepted.

The research of McMahan et al.[7] is considered the pioneering work on FL to make communication more efficient by increasing the calculated quantity of each client between each communication round. This method is based on averaging local stochastic gradient descent (SGD) updates for the primary problem. They also pointed out that increasing the parallelism, which means motivating more clients to join training on each round, is effective. Inspired by the latter, Nishio and Yonetani[11] built the FedCs framework to integrate the available clients to the utmost extent in each training round to make it efficiently in practice. A maximum mean discrepancy was inserted into the FL algorithm to enforce the local model to acquire more knowledge from others in training devices, thus speeding up convergence[12]. Yurochkin et al.[13] designed the Bayesian Nonparametric FL framework, which is the state of the art since it can aggregate local models into a federated model without extra parameters, thus avoiding unwanted communication rounds. The experiment shows that they can obtain a satisfactory accuracy rating with only one communication round.

Even if the communication rounds are optimized, how to accelerate model update is a remained problem. Initially, McMahan et al. proposed two strategies to **reduce model-update time[14]**. One is a structured update, which means transmitting only part of the updated model by means of a low-rank model or in a random mask way. Likewise, an end-to-end neural network is a kind of structured update mode which maps update information into a lower-dimension space thus relieving the pressure of communication[15]. The other is sketched update, which refers to making use of a compressed update model. Zhu and Jin[16] optimized sparse evolutionary training (SET) thus conveying only a piece of parameters to the server, which resembles the sketched update.

Since in each round, each client manipulates fixed epochs, Jiang and Ying[17] designed an adaptive method for local training. The local training epochs are decided by the server according to training time and training loss, thus it will reduce local training time when the loss is getting small. The above-mentioned algorithms are all based on stochastic gradient descent (SGD), but this method could be inefficient if the function is anisotropic. Therefore, Liu, Chen, Chen, and Zhang[18] utilized momentum gradient descent to consider previous gradient information in each local training epoch to accelerate convergence speed. These algorithms are not fully suitable for all federal settings. Therefore, a more flexible communication-efficient method needs to be explored for high efficiency demand in digital forencics.

While local updating methods can reduce the total number of communication rounds, model **compression schemes** (e.g., sparsification, subsampling, and quantization) can significantly reduce the size of messages communicated at each round. These methods have been extensively studied, both empirically and theoretically, in previous literature for distributed training in datacenter environments. In federated environments, the low participation of devices, non-iid local data, and local updating schemes pose novel challenges to these model compression approaches. Several works have provided practical strategies in federated settings, such as forcing the updating models to be sparse and low-rank; performing quantization with structured random rotation[19]; using lossy compression and dropout to reduce server-to-device

[11] Nishio, T., & Yonetani, R. (2019, May). Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE international conference on communications (ICC)* (pp. 1-7). IEEE.

[12] Yao, X., Huang, C., & Sun, L. (2018, December). Two-stream federated learning: Reduce the communication costs. In *2018 IEEE Visual Communications and Image Processing (VCIP)* (pp. 1-4). IEEE.

[13] Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., & Khazaeni, Y. (2019, May). Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning* (pp. 7252-7261). PMLR.

[14] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

[15] H. Li and T. Han, "An End-to-End Encrypted Neural Network for Gradient Updates Transmission in Federated Learning," 2019 Data Compression Conference (DCC), 2019, pp. 589-589, doi: 10.1109/DCC.2019.00101.

[16] Zhu, H., & Jin, Y. (2019). Multi-objective evolutionary federated learning. *IEEE transactions on neural networks and learning systems*, *31*(4), 1310-1322.

[17] Jiang, P., & Ying, L. (2020, March). An optimal stopping approach for iterative training in federated learning. In *2020 54th Annual Conference on Information Sciences and Systems (CISS)* (pp. 1-6). IEEE.

[18] Liu, W., Chen, L., Chen, Y., & Zhang, W. (2020). Accelerating federated learning via momentum gradient descent. *IEEE Transactions on Parallel and Distributed Systems*, *31*(8), 1754-1766.

[19] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

communication[20]; applying Golomb lossless encoding[21].

On the other hand, when dealing with system **heterogeneity** in federated networks, there is significant variability in the characteristics of the system across the federation, as devices may differ in terms of hardware, network connectivity, and battery power, thereby causing unbalanced training time. These systems characteristics introduce issues such as stragglers, significantly more prevalent than in the typical centralized case. Up to now, methods to deal with system heterogeneity mainly focus on resource allocation for heterogeneous devices and fault tolerance for devices prone to being offline.

When learning over remote devices, **fault tolerance** becomes more critical as it is common for some participating devices to drop out at some point before the completion of the given training iteration. To address the issue of stragglers Smith et al.[22] considered the influence of low participation in the training process to resist device drop out. Another practical strategy is to simply ignore such device failure[23], which may introduce bias into the device sampling scheme if the failed devices have specific data characteristics. For instance, devices from remote areas may be more likely to drop due to poor network connections and thus the trained federated model will be biased towards devices with favorable network conditions. While several recent works have investigated convergence guarantees of variants of federated learning methods, few analyses allow for low participation or study directly the effect of dropped devices. To enable FL system to be robust to dropped participants, scholars also designed a secure aggregation protocol[24] which is tolerant against arbitrary dropouts as long as surviving users are enough to join federate updates. Lib et al.[25] take stragglers into account and allow these devices to spend different locally update computation times. Wu et al.[26] also fully considered the device straggling phenomenon in a heterogeneous network. They made use of a cache structure to store those unreliable user update thus alleviating their trustless impact on the global model.

For the sake of **resource constraint**, most foregoing works devote to allocating resources properly to heterogeneous devices. For instance, Kang et al.[27] took overhead in heterogeneous clients into consideration to motivate more high-quality devices to participate to the training process. In a similar way, Nishio and Yonetani[28] explore novel device sampling policies based on systems resources, where the server aims to aggregate as many device updates as possible within a pre-defined time window. And Tran et al.[29] studied training accuracy and convergence time with the influence of heterogeneous power constraints. Meanwhile,

[20]Caldas, S., Konečny, J., McMahan, H. B., & Talwalkar, A. (2018). Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*.

[21] Sattler, F., Wiedemann, S., Müller, K. R., & Samek, W. (2019). Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, *31*(9), 3400-3413.

[22] Smith, V., Chiang, C. K., Sanjabi, M., & Talwalkar, A. S. (2017). Federated multi-task learning. *Advances in neural information processing systems*, *30*.

[23] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., ... & Roselander, J. (2019). Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, *1*, 374-388.

[24] Hao, M., Li, H., Luo, X., Xu, G., Yang, H., & Liu, S. (2019). Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, *16*(10), 6532-6542.

[25] Li, S., Cheng, Y., Liu, Y., Wang, W., & Chen, T. (2019). Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*.

[26] Wu, W., He, L., Lin, W., Mao, R., Maple, C., & Jarvis, S. (2020). Safa: a semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, *70*(5), 655-668.

[27] Kang, J., Xiong, Z., Niyato, D., Yu, H., Liang, Y. C., & Kim, D. I. (2019, August). Incentive design for efficient federated learning in mobile networks: A contract theory approach. In *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)* (pp. 1-5). IEEE.

[28] Nishio, T., & Yonetani, R. (2019, May). Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE international conference on communications (ICC)* (pp. 1-7). IEEE.

[29] Tran, N. H., Bao, W., Zomaya, A., Nguyen, M. N., & Hong, C. S. (2019, April). Federated learning over wireless networks: Optimization model design and analysis. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications* (pp. 1387-1395). IEEE.

Chai et al.[30] considered the impact of resources (e.g. CPU, memory, and network resources) heterogeneity on the training time of FL. To address this issue, Li, T. et al.[31] designed fairness metrics to measure loss in devices and a q-Fair optimization goal to impel fair resource allocation in FL. While these methods primarily focus on systems variability to perform active sampling, we note that it is also worth considering actively sampling a set of small but sufficiently representative devices based on the underlying statistical structure.

Challenges arise when training federated models from data that is not identically distributed across devices, both in terms of modelling the data, and in terms of analyzing the convergence behaviour of associated training procedures. The traditional machine learning approach, implicitly or explicitly, assumes that the **data distribution** is identically independent. This scenario is suitable for collecting all data and then training in a distributed way. However, data is collected from various devices or institutions and thereby do not follow Identically Independent Distribution (IID), thus training a single global model on the union of client datasets, becomes harder with non-IID data. To tackle this problem, the general resolution is to focus on a global model, modify the local training mode (e.g., through different hyperparameter choices), or add some extra procedure to the data pre-processing stage.

The first proposed FedAvg algorithm resolves this issue by averaging local upgrades on each device directly. In addition, Mohri et al.[32] noticed that previous work ignore the importance of fairness which may lead to biased centralized model. They improved the global model to cope with any target distribution comprised of a mixture of different clients. As for the aggregation stage, convergence behaviour is another stressed issue. The existence of heterogeneity may lead to the mis-convergence of the global model. Wang, X. et al.[33] discussed the convergence bound of FL based on gradient-descent in Non-IID data background, and they further enhanced an improved adaptive method to reduce loss function within constraints of resource budget. Moreover, the authors[34] gave four kinds of convergence theorems with different parameter settings or premises for FedAvg in Non-IID situations. These studies partially fill the theoretical gap in the research on the convergence speed of an FL algorithm. Besides, they provide the effect of parameter adjustment on the convergence speed for guidance. To understand the performance of FedAvg in statistically heterogeneous settings, FedProx[35] has recently been proposed. FedProx makes a small modification to the FedAvg method to help ensure convergence, both theoretically and in practice. FedProx can also be interpreted as a generalized, re-parameterized version of FedAvg that has practical ramifications in the context of accounting for systems heterogeneity across devices.

For data pre-processing, Huang, Shea et al.[36] introduced clustering thought with FL and constructed a community-based FL method. By separating independent data into different clusters, and then processing federated training on each community, the non-IID problem is thus can be resolved. However, one drawback is that it's not suitable for massively data training due to high parameter conversion overhead. In a hierarchical heterogeneous horizontal framework, the method projects each embedding submanifold into a common

[30] Chai, Z., Fayyaz, H., Fayyaz, Z., Anwar, A., Zhou, Y., Baracaldo, N., ... & Cheng, Y. (2019). Towards taming the resource and data heterogeneity in federated learning. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)* (pp. 19-21).

[31] Li, T., Sanjabi, M., Beirami, A., & Smith, V. (2019). Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*.

[32] Mohri, M., Sivek, G., & Suresh, A. T. (2019, May). Agnostic federated learning. In *International Conference on Machine Learning* (pp. 4615-4625). PMLR.

[33] Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., & Chen, M. (2019). In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, *33*(5), 156-165.

[34] Li, X., Huang, K., Yang, W., Wang, S., & Zhang, Z. (2019). On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*.

[35] Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, *37*(3), 50-60.

[36] Huang, L., Shea, A. L., Qian, H., Masurkar, A., Deng, H., & Liu, D. (2019). Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records. *Journal of biomedical informatics*, *99*, 103291.

embedding space to overcome data heterogeneity[37].

For some applications, it may be possible to augment data in order to make the data across clients more similar. One approach is to create a small dataset which can be shared globally. This dataset may originate from a publicly available proxy data source, a separate dataset from the clients' data which is not privacy sensitive, or perhaps a distillation of the raw data. Another idea is to optimize modelling to achieve personalization for individual devices such as MOCHA, which introduced multi-task learning to make utilization of shared representation[Error! Bookmark not defined.]. In a similar work[38], the authors considered a solution to deal with n on-iid data by sharing a small set of data among each local model. Huang, Yin et al.[39] also gained a good deal of enlightenment from the previous data sharing ideology to overcome the Non-IID problem. They put cross-entropy loss into the transmission process and assign different local update times for each client in each round.

3.3. Optimization algorithms followed in GRACE

Optimization strategies and in particular *aggregation* algorithms play an important role in Federated Learning as they are responsible for combining the knowledge from all devices/nodes without knowing about users'/organization's private data. Federated learning revolves around the Federated *averaging* algorithm described in[7] , called "FedAvg". FedAvg is the very first vanilla FL algorithm formulated by Google for solving distributed training problems. Since then, many variants of FedAvg algorithms have been proposed, such as "FedProx", "FedOpt", etc, to address the main challenges of the domain. Looking in to the details of the FL process, many optimization methods use local client updates, in which clients update their models multiple times before communicating with the server. This can greatly reduce the amount of communication required to train a model.

Within GRACE, three main aggregation mechanisms have been followed, examined in detail and applied to specific scenarios, namely, the FedAvg, FedProx and FedOpt.

Optimization parameters: Once the topology of the node network is chosen, one can control different parameters of the federated learning process to optimize learning: a) Number of federated learning rounds, b) Total number of nodes used in the process, c) Fraction of nodes used at each iteration for each node, d) Local batch size used at each learning iteration, e) Number of iterations for local training before pooling, f) Local learning rate. These parameters have to be optimized depending on the constraints of the machine learning application (e.g., available computing power, available memory, bandwidth).

Formally, FL is a communication-training protocol acting as per Algorithm 1 (defined in[35]). The framework involves a group of devices named clients and a server coordinating the learning process. Each client has a local training dataset which is never uploaded to the server. The goal is to train a global model by aggregating the results of the local training clients. Algorithm 1 encodes the training procedure described below. There is a fixed set of $I = \{1, \ldots, N\}$ clients (each with a local dataset), before every communication $round\ t \in \{0, E, \ldots, (T-1)E\}$ the server randomly selects a set I_t of $C \cdot N$ clients. Parameters, fixed by server, include: a set I grouping N clients, the ratio of clients C selected at each round, the number of communication rounds T and a number of local epochs E. The server sends to the clients the current global algorithm state, then it asks the clients to perform local computations based on the global state and their local dataset, successively it requests the clients to send back an update; at the end, the server updates the weights of the model by aggregating clients' updates and the process repeats. The model is defined by its weights: at the

[37] Gao, D., Ju, C., Wei, X., Liu, Y., Chen, T., & Yang, Q. (2019). Hierarchical Heterogeneous Horizontal Federated Learning for EEG.
[38] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., & Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.
[39] Huang, L., Yin, Y., Fu, Z., Zhang, S., Deng, H., & Liu, D. (2020). LoAdaBoost: Loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data. *Plos one*, *15*(4), e0230706.

end of each epoch, w_{t+1}^{i} defines the weight of the client $i \in I$. For each communication round, w_t is the global model detained by the server and w_{TE} is the final weight.

Algorithm 1 Generic Federated Learning Algorithm

```
Input: $N, C, T, E$
Output: $w_{TE}$
 1: Initialize $w_0$.
 2: for each round $t \in \{0, E, 2E, \dots, (T-1)E\}$ do
 3:     $m \leftarrow \max(C \cdot N, 1)$
 4:     $I_t \leftarrow$ (random set of $m$ clients)
 5:     for each client $i \in I_t$ in parallel do
 6:         $w_{t+E}^{i} \leftarrow \text{CLIENT-UPDATE}(w_t)$
 7:     end for
 8:     $w_{t+E} \leftarrow \text{AGGREGATION}(w_{t+E}^{1}, \dots, w_{t+E}^{N})$
 9: end for
10: return $w_{TE}$
```

3.3.1. Federated stochastic gradient descent (FedSGD)

Deep learning training mainly relies on variants of stochastic gradient descent (SGD), where gradients are computed on a random subset of the total dataset and then used to make one step of the gradient descent. **Stochastic gradient descent:** is a drastic simplification of the gradient descent algorithm which computes the gradient over an extremely small subset (mini-batch) of the whole dataset. In the simplest case, corresponding to maximum stochasticity, one data sample is selected at random in each optimization step.

Let F be the loss function, i.e., the difference between the true value of the objective function and the output computed by the network. The back-propagation algorithm computes the partial derivative of F concerning each parameter w and updates the parameter according to the gradient of F. $F_i(w)$ is the value of the loss function at the i-th example.

The update rule of stochastic gradient descent for a parameter w is:

$$w := w - lr\nabla F_i(w)$$

where lr is the learning rate and each loss function F_i is typically associated with the i-th observation in the dataset. We refer to one full iteration over all available input data as an epoch.

Federated stochastic gradient descent[40] is the direct transposition of this algorithm to the federated setting, but it uses a random fraction C of the nodes and considers all the data on this node. The gradients are averaged by the server proportionally to the number of training samples on each node and used to make a gradient descent step. In particular, the selective SGD chooses a fraction of parameters to be updated at each iteration. This selection can be completely random, but a smart strategy is to select those that have a larger gradient $\nabla F_i(w)$, according to a selection rate θ.

[40] Shokri, R., & Shmatikov, V. (2015, October). Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security* (pp. 1310-1321).

3.3.2. FedAvg

Federated averaging (FedAvg) is a generalization of FedSGD, which allows local nodes to perform more than one batch update on local data and it exchanges the **updated weights** rather than the gradients. The rationale behind this generalization is that in FedSGD, if all local nodes start from the same initialization, averaging the gradients is strictly equivalent to averaging the weights themselves. Further, averaging tuned weights coming from the same initialization does not necessarily hurt the resulting averaged model's performance.

FedAvg can be viewed as a communication-light implementation of the standard FedSGD and it is currently the most commonly used federated learning optimization algorithm. Unlike the conventional SGD, wherein the updates at different clients are aggregated right after every local step, its essential idea is to compute the gradients locally optimizing the local model on the local data, and updates are only aggregated on a centralised device after every $s-th$ local step, where $s \geq 1$ is a small constant.

In particular, at each iteration, FedAvg first locally performs s epochs of stochastic gradient descent (SGD) on K devices—where K is a small fraction of the total devices in the network. The devices then communicate their model updates to a central server, where they are averaged. At each round of FedAvg, the aim is to minimize the objective of the global model w which is just the sum of the weighted average of the local device loss.

The objective function is defined as follows:

$$\min_{w} f(w) = \sum_{k=1}^{N} p_k \underbrace{F_k(w)}_{\text{loss on device } k}$$

where N is the number of devices, $p_k \geq 0, \sum_k p_k = 1$, and F_k is the local objective function for the kth device. The user-defined term p_k specifies the relative impact of each device. The round starts with the random selection of a subset of N clients. Then the server broadcasts its global model w to each client. In parallel, the clients run SGD on their loss function F_k and sent the resulting model w_k to the server for aggregation. The server then updates its global model as the average of these local models. The process is then repeated for n such communications rounds.

The number of local epochs in FedAvg plays an important role in convergence. On one hand, performing more local epochs allows for more local computation and potentially reduced communication, which can greatly improve the overall convergence speed in communication-constrained networks. On the other hand, with dissimilar (heterogeneous) local objectives F_k, a larger number of local epochs may lead each device towards the optima of its local objective as opposed to the global objective—potentially hurting convergence or even causing the method to diverge.

Setting the number of local epochs to be high may increase the risk that devices do not complete training within a given communication round and must therefore drop out of the procedure

The FedAvg algorithm is a relatively basic federated optimization algorithm, while its deployment is relatively simple, and its application field is vast.

3.3.3. FedProx

In practice, it is therefore important to find a way to set the local epochs to be high (to reduce communication) while also allowing for robust convergence. We note that the optimal setting for the number of local epochs is likely to change at each iteration and on each device—as a function of both the local data and available systems resources. Indeed, a more natural approach than mandating a fixed number of local epochs is to allow the epochs to vary according to the characteristics of the network and to carefully merge solutions by accounting for this heterogeneity. FedProx is a generalization of FedAvg, which allows for variable amounts of work to be performed locally across devices based on their available systems resources, and then aggregate the partial solutions sent from the stragglers (as compared to dropping these devices). A proximal term is proposed for the local subproblem to effectively limit the impact of variable local updates.

In particular, instead of just minimizing the local function $F_k(\cdot)$, device k uses its local solver of choice to approximately minimize the following objective h_k:

$$\min_w h_k(w;\ w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

The proposed proximal term addresses the issue of statistical heterogeneity by restricting the local updates to be closer to the initial (global) model without any need to manually set the number of local epochs, while it allows for safely incorporating variable amounts of local work resulting from systems heterogeneity. In general, proximal terms such as the one above are a popular tool utilized throughout the optimization literature. FedProx makes only lightweight modifications to FedAvg, this enables easy integration of FedProx into existing packages/systems, such as the NVIDIA Flare that we have selected. We should also note that FedAvg is a special case of FedProx with $\mu = 0$.

3.3.4. FedOpt

One could also utilize optimizers other than SGD on the clients, or use an alternative update rule on the server. This family of algorithms, which are referred [41]to as $FEDOPT$, is formalized in Algorithm 2. In Algorithm 2, *CLIENTOPT* and *SERVEROPT* are gradient-based optimizers with learning rates ηl and η respectively. Intuitively, *CLIENTOPT* aims to minimize the same objective function as in FedAVG based on each client's local data, while SERVEROPT optimizes from a global perspective. FEDOPT naturally allows the use of adaptive optimizers (eg. ADAM, YOGI, etc.), as well as techniques such as server-side momentum. In its most general form, FEDOPT uses a CLIENTOPT whose updates can depend on globally aggregated statistics (e.g. server updates in the previous iterations). *FEDOPT* also allows learning rates η and ηl to depend on the round t to encompass learning rate schedules. Worth noting that FEDOPT is already integrated into the IA FLARE framework. By using adaptive methods, namely, by including ADAGRAD, ADAM, and YOGI, (which generally require maintaining state) on the server and SGD on the clients, FEDOPT ensures the same communication cost as FEDAVG while also working cross-device settings. It has been demonstrated that adaptive optimizers can be powerful tools in improving the convergence of FL, a conclusion that we have also drawn based on the experiments that we have conducted.

[41] Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., ... & McMahan, H. B. (2020). Adaptive federated optimization. arXiv preprint arXiv:2003.00295.

```
1: Input: $x_0$, ClientOpt, ServerOpt
2: for $t = 0, \cdots, T-1$ do
3:     Sample a subset $\mathcal{S}$ of clients
4:     $x_{i,0}^t = x_t$
5:     for each client $i \in \mathcal{S}$ in parallel do
6:         for $k = 0, \cdots, K-1$ do
7:             Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$
8:             $x_{i,k+1}^t = \text{ClientOpt}(x_{i,k}^t, g_{i,k}^t, \eta_l, t)$
9:         $\Delta_i^t = x_{i,K}^t - x_t$
10:    $\Delta_t = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \Delta_i^t$
11:    $x_{t+1} = \text{ServerOpt}(x_t, -\Delta_t, \eta, t)$
```

4. Technical & Performance analysis

Section 4 presents the technical analysis of the FL framework and reports the results and the performance of the distributed trained models under different scenarios, using the NVFLARE ecosystem. Multiple experiments are conducted in the context of image classification and named entity recognition, to investigate how FL affects the whole training and evaluation process. The experiments highlight the impact of different learning algorithms and data distribution variations on global model's performance.

4.1 Experiments details

Image classification

Image classification aims to assign an input image one label from a predefined set of categories. The DL model developed for this task is trained and evaluated on the Cifar-10 dataset[42], both for FL and centralised learning. Cifar-10 is one of the most widely used datasets for computer vision tasks that contains 60000 RGB images in 10 different, diverse classes (see Fig. 16). We choose Cifar-10 for our experiments since the provided images are low resolution (32x32) and therefore it is feasible to perform multiple FL scenarios to extract valuable information related to different learning strategies. The DL model that is used is a CNN, which has 3 convolutional layers blocks, each consisting of two 3x3 layers and one max-pooling layer, followed by three fully connected layers. This model is not the state-of-the-art on the CIFAR-10 dataset but is sufficient to show the variation in performance for our investigation. Finally, the classification accuracy is used as the evaluation metric.

[42] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

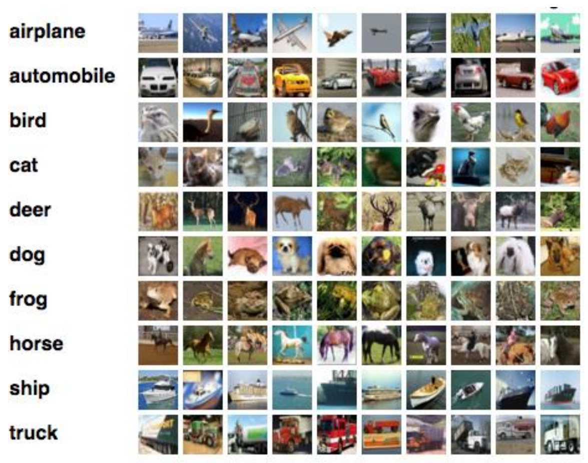


Figure 16: Benchmark dataset: CIFAR10 dataset (an example of an IID type)

For the centralised training, the whole dataset is centralised to a single node, where 50000 images are used as training dataset and the remaining for testing. The performance of this model will be used as a benchmark in a thorough evaluation against FL algorithms.

For the FL experiments, we start by analysing the different learning strategies from Section 3, assuming of identically distributed data among participants. Next, we analyse the classification performance as a function of non-identical data distribution, highlighting how proposed aggregation methods can improve performance in these less identical cases.

CIFAR-10 is not an FL related dataset, since it has been built for traditional centralised learning algorithms. Therefore, dataset partition required to make Cifar-10 suitable for FL application, utilising strategies, which ensure that the data splits follow a close to real-world distribution. We follow the strategy described by Wang[43] to simulate both homogeneous and heterogeneous partitions. This approach is based on a Dirichlet sampling algorithm, where a parameter α controls the amount of heterogeneity with respect to data size and classes' distribution. We use the original CIFAR-10 test set (10000 images) as the global test set for fair comparison among different set ups.

[43] Wang, Hongyi, et al. "Federated learning with matched averaging." *arXiv preprint arXiv:2002.06440* (2020).

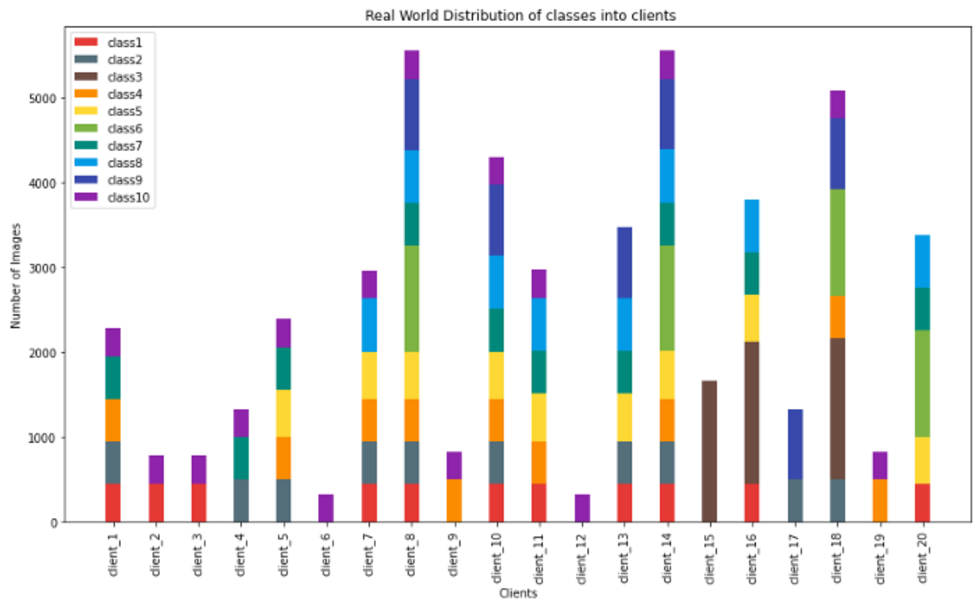


Figure 17: Cifar10 real-world data partition among clients

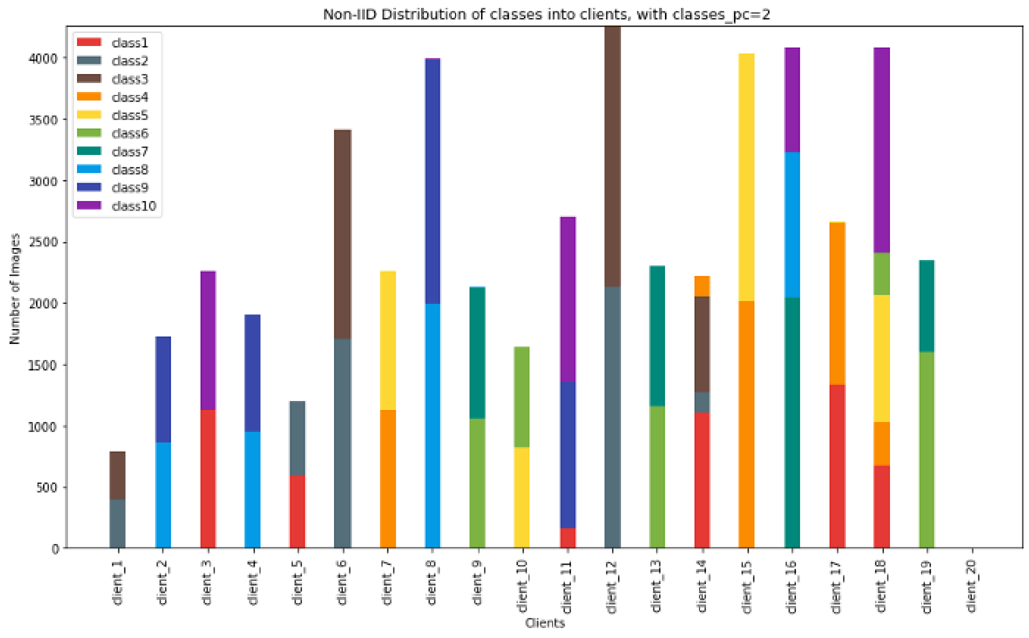


Figure 18: An extreme highly heterogenous dataset partition.

Now the question is how to prepare a non-iid/real-world dataset for the image classification task. One can follow the steps indicated below:

1. **Setting Hyper-parameters:** classes_per_client, number_of_clients, batch_size,
2. **Creating the distribution:** download/acquisition of the original dataset, data splitting for training and testing purposes (80/20, or 70/30 may be sufficient)
   a) **Data Transformation:** data transformations and creation of a random distribution for the clients, such that every client has an arbitrary number of images.
   b) **Splits the given images into n clients**: creation of a split, which is further used to create the

real-world dataset

c) a similar split may be created for the non-IID dataset

d) **Data Shuffling**: shuffling the images of each client respectively

e) **Data Loading**: conversion of the split into a data loader (image augmentation is done in this part) to be provided as an input to the model for training.

f) **FL Training:** datasets can be further used by federated learning to develop state of the art models.

g) **Baseline retraining:** a set of images to be saved on the global server to retrain the client's model before aggregation. This technique of retraining all the models on the global server deals with non-IID/real-world datasets.

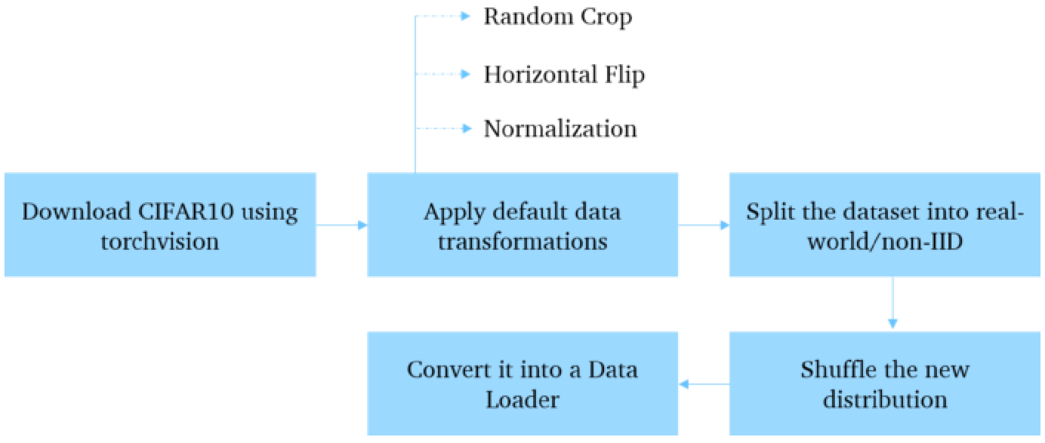

Random Crop
Horizontal Flip
Normalization
Download CIFAR10 using torchvision
Apply default data transformations
Split the dataset into real-world/non-IID
Shuffle the new distribution
Convert it into a Data Loader

Figure 19: Data flow diagram

Named entity recognition

Named Entity Recognition (NER) can be considered as subtasks of information extraction, aiming to classify and locate entities listed in unstructured text data into predefined categories. The DL model is trained at CoNLL-2003[44], a well-known open-source dataset for the task of NER. The CoNlLL-2003 dataset consists of 20745 sentences of 301415 labelled tokens among 9 classes. The dataset is split to training and test set, containing 20345 and 400 sentences respectively.

For the task of NER, a Bidirectional Long Short-Term Memory (LSTM) is created. The model firstly contains an embedding layer, then a bidirectional LSTM layer is added, while a fully connected layer transforms the output of the LSTM. The total number of trained parameters of the NER model is 236809. The model is trained using Adam stochastic gradient descent as an optimizer and sparse categorical cross-entropy loss function. The F1 Score is used to evaluate the performance of the trained models under the FL set up.

Training and experiments are conducted both for FL and centralised learning. For the centralised training, the training dataset is centralised to a single node. The performance of the NER model trained in a centralised manner represents the benchmarking training and it is used as a baseline to compare the performance of the next experiments.

The first experiment regards the training of the NER model in an FL environment of 10 clients in a homogeneous way since the training dataset is equally split to each participant. For the second experiment, the training dataset is split into 10 clients in an unequal way, representing a heterogeneous splitting.

[44] Sang, Erik F., and Fien De Meulder. "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition." *arXiv preprint cs/0306050* (2003).

Specifically, the training set is split among clients in three different ways, which are mild, moderate and intense.

4.2 MLFlow in distributed learning scenarios

4.2.1 Deployment and Architecture

MLFlow is an open-source platform for managing the machine learning lifecycle. In the GRACE project, our goal is to integrate MLFlow within the distributed learning functionalities for experimentation, reproducibility, deployment, and a central model registry. The MLFlow system architecture consists of the four main components presented in the following image:

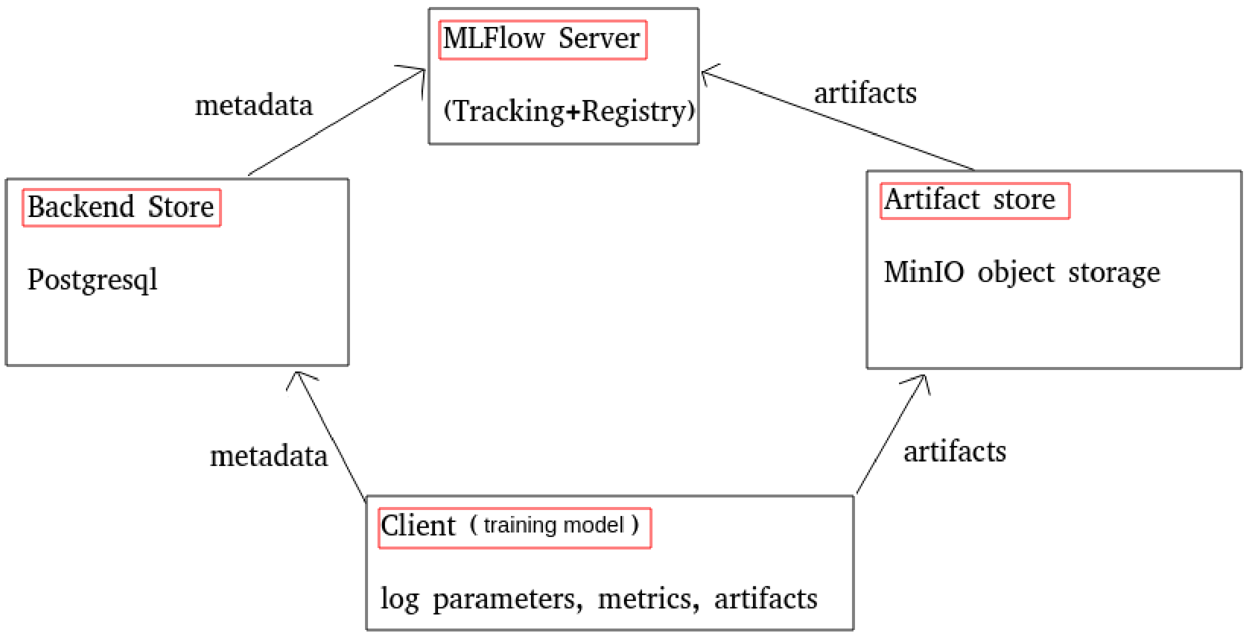


Figure 20: MLFlow system architecture

MLFlow is deployed on a separate Virtual Machine. The specifications of the system can be found in the table below:

VM specifications	
IP	10.41.41.211
RAM	2GB
CPU	1 core
Storage	1TB

Table 3: VM specifications

The service was deployed using Docker. The following tables contain important information about all MLFlow's sub-services.

MLFlow Server	
Description	Monitoring Web UI

URI	http://10.41.41.211:5000
MinIO	
Description	Storage of the models as artifacts
URI	http://10.41.41.211:9000
Username	graceminio
Password	gracepass
Postgres	
Description	Storage of the training hyperparameters and evaluation data
Username	gracepost
Password	grace

Table 4: MLFlow's sub-services details

The user interacts with the system through the MLFlow Server, using the URI found in the in the corresponding table above. The port can be reached only from inside the GRACE network. Postgres and MinIO services work in the background, interacting with the Web UI and the Model Registry respectively. More specifically, Postgres is responsible for sending training data to the Web UI and MinIO for storing the resulted model at the end of the training process. Apart from the Model Registry, where you can ingest, fetch and mark the production stage of a model, the trained weights are also saved in a local folder within the VM named *"models"*. This directory is mapped to the rest of the GRACE servers and it can be reached through the following absolute path:

/DATA/MLflow/models

The list with the trained models can also be found in the MinIO server, inside the directory named *"models"*, by connecting to the URI and using the credentials mentioned earlier.

4.2.2 Usage Guidelines

For the remaining part of the section, we will present an example of the MLflow use, demonstrating the extra lines of code that were added in the train and evaluation scripts of a Face Detection model. For integrating mlflow to our Python code the following libraries must be imported:

- mlflow
- Mlflow.pyfunc

In the beginning of the training script the user needs to define a few environment variables that are responsible for the connection with the MinIO and Web UI services. The user can define tags that can be attached to a run. These tags are defined priorly within the Federated Learning configuration file and then referenced within the training script. A name of the experiment where the run will be attached must also be provided. If the name of this experiment does not exist, a new experiment with the defined name will be created. The following code block summarizes the aforementioned information.

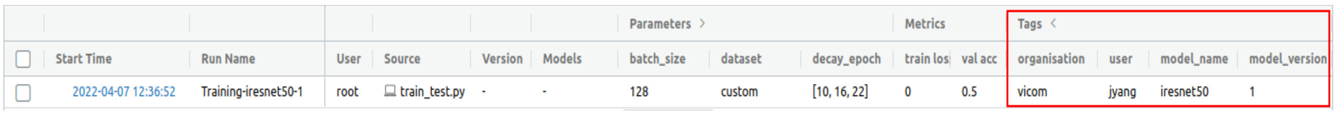
```
os.environ['MLFLOW_S3_ENDPOINT_URL'] = 'http://10.41.41.211:9000'  # Definition of MinIO URI
os.environ['AWS_ACCESS_KEY_ID'] = 'graceminio'   # Definition of MinIO username
os.environ['AWS_SECRET_ACCESS_KEY'] = 'gracepass'   # Definition of MinIO password

mlflow.set_tracking_uri("http://10.41.41.211:5000")  # Definition of MLflow Server URI for tracking
tracking_uri = mlflow.get_tracking_uri()   # Fetch tracking_uri
print("Current tracking uri: {}".format(tracking_uri))  # Print tracking_uri

tags = {
    "organisation": self.org_name   # vicom -> Definition of the organisation who is implementing the run
    "user": self.user   # jyang -> Definition of the user who is implementing the run
    "model_name": "iresnet50"   # Definition of the model name
    "model_version": self.model_version   # 1 -> Definition of the model version
}
mlflow.set_experiment("vicom-face-recognition") # Define the name of the experiment where the run will be
attached
```

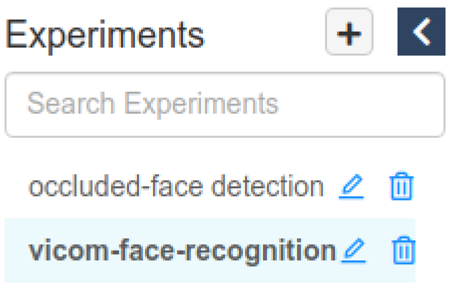
A neat use of the MLflow within the GRACE project is crucial, since the platform is intended to accommodate a large number of models for various implementations, as well as different versions of each model. For this reason, we recommend the following:

- Every new attempted run should carry the following tags:
    - ➢ Organisation name
    - ➢ Developer Name & Surname: (***NSurname***)
    - ➢ Model name
    - ➢ Model version



							Parameters >			Metrics		Tags <			
☐	Start Time	Run Name	User	Source	Version	Models	batch_size	dataset	decay_epoch	train los:	val acc	organisation	user	model_name	model_version
☐	2022-04-07 12:36:52	Training-iresnet50-1	root	train_test.py	-	-	128	custom	[10, 16, 22]	0	0.5	vicom	jyang	iresnet50	1

- Different experiments will correspond to different modalities of the project, such as face detection, person detection, language identification etc. Thus, whenever a partner wants to create a run for a model that corresponds to a specific modality is asked to define the name of the experiment accordingly.
    - ➢ Naming convention: ***partner_name-modality_name*** (*vicomtech-face-detection)*



- Different runs within each experiment will represent different model versions or training attempts of the same modality, and should be named appropriately in a clear manner to avoid potential confusions between the variants.
    - ➢ Naming convention: ***task-model-version*** (*training-iresnet50-1*)

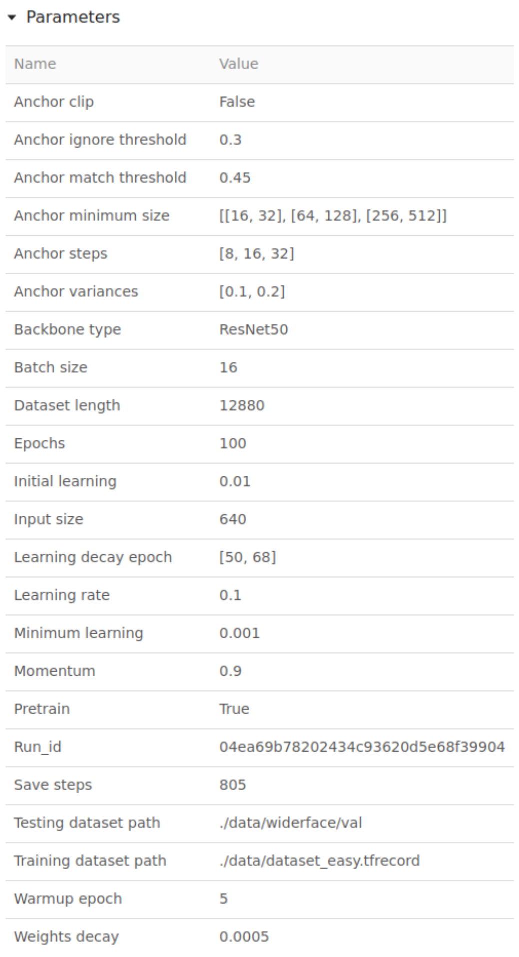
Having defined the URIs, tags and experiment name, the user must nest all the training code below the command that starts the tracking of the run. Additionally, they need to ingest the tags they defined earlier and specify the training parameters to be logged (see following code block).

```
with mlflow.start_run(run_name="Training-iresnet50-1") as run: # Start tracking and define the run name
    mlflow.set_tags(tags)  # Attach the tags defined earlier

    cfg = load_yaml(FLAGS.cfg_path)
    cfg_val = load_yaml("./configs/retinaface_res50_val.yaml")
    # Definition of the training parameters
    params = {"Batch size": cfg["batch_size"], "Input size": cfg["input_size"],
"Backbone type": cfg["backbone_type"], "Training dataset path": cfg["dataset_path"],
"Learning decay epoch": cfg["lr_decay_epoch"], "Weights decay": cfg["weights_decay"],
"Dataset length": cfg["dataset_len"], "Testing dataset path": cfg["testing_dataset_path"],
"Anchor minimum size": cfg["min_sizes"], "Anchor steps": cfg["steps"],
"Anchor match threshold": cfg["match_thresh"], "Anchor ignore threshold": cfg["ignore_thresh"],
"Anchor variances": cfg["variances"], "Anchor clip": cfg["clip"], "Epochs": cfg["epoch"],
"Initial learning": cfg["init_lr"], "Learning rate": cfg["lr_rate"], "Warmup epoch": cfg["warmup_epoch"],
"Minimum learning": cfg["min_lr"], "Momentum": cfg["momentum"], "Pretrain": cfg["pretrain"],
"Save steps": cfg["save_steps"]}

    mlflow.log_params(params) # Logging of the training parameters
```

The hyperparameters will be attached to the training run and will be displayed on the dashboard as shown in Fig. 21 In this way, the user can easily identify the parameters of each training, when searching the history for previous versions of the model.


▾ Parameters

Name	Value
Anchor clip	False
Anchor ignore threshold	0.3
Anchor match threshold	0.45
Anchor minimum size	[[16, 32], [64, 128], [256, 512]]
Anchor steps	[8, 16, 32]
Anchor variances	[0.1, 0.2]
Backbone type	ResNet50
Batch size	16
Dataset length	12880
Epochs	100
Initial learning	0.01
Input size	640
Learning decay epoch	[50, 68]
Learning rate	0.1
Minimum learning	0.001
Momentum	0.9
Pretrain	True
Run_id	04ea69b78202434c93620d5e68f39904
Save steps	805
Testing dataset path	./data/widerface/val
Training dataset path	./data/dataset_easy.tfrecord
Warmup epoch	5
Weights decay	0.0005

Figure 21: Training parameters as displayed on the dashboard

Subsequently, loading the weights of the pre-trained model from the MLFlow Model Registry would require the following lines of code. Before loading a model for training, it is important to have the model registered via the MLFlow server.

```
model_name = "face-recognition"
model_version = 1
backbone = mlflow.pyfunc.load_model(
    model_uri = f"models:/{model_name}/model_version")
backbone = backbone._model_impl.pytorch_model
```

Keras and Pytorch offer pre-built functions (i.e. ***mlflow.keras.autolog()***) for automatically logging the training parameters without having to explicitly define them one by one. In the given example the training parameters are taken from the training configuration file. Towards the end of the script and after the loss metrics on the training and validation sets (or any other metrics) are calculated, the user needs to log them, as in the example below.

```
losses['val loc loss'] = val_loss
losses['train loc loss'] = train_loss
mlflow.log_metrics(losses, steps) # Logging of the train and validation loss metrics
```

The final step is to log the parameter run_ID, add an image of the network architecture (optional), save the model as artifacts and include a text file named ”*dataset_version.txt”*. An example is provided in the following code block.

```
mlflow.log_param("Run_id", run.info.run_id)  # Logging of the parameter run_ID
mlflow.log_artifact("./photo/R50-symbol.png")   # Ingesting the image of the network architecture
mlflow.keras.log_model(keras_model=model, artifact_path="model") # Logging the model as artifacts
mlflow.log_text("dataset_version.txt")  # Logging a txt file containing information about the dataset
```

The text file will contain important information, such as source & type, name (if any) and content category, of the data that was used for the training. This file will be registered as artifact along with the model and will assist in the quick identification of the correspondence between the training data and the model version. The run_ID is a serial number that is uniquely assigned to each attempted run. Similar to the run ID, MLflow also provides a unique experiment ID. Both these IDs will help the user to map a specific run with the corresponding resulted model.

MLFlow offers practical features that assist the user to keep track of the training process. It is possible to visualise tracking data in the form of a line-graph and monitor it in real-time. Below we see an example with the training and validation losses taken from the Face Recognition modality, that we ingested earlier at the end of each training step.

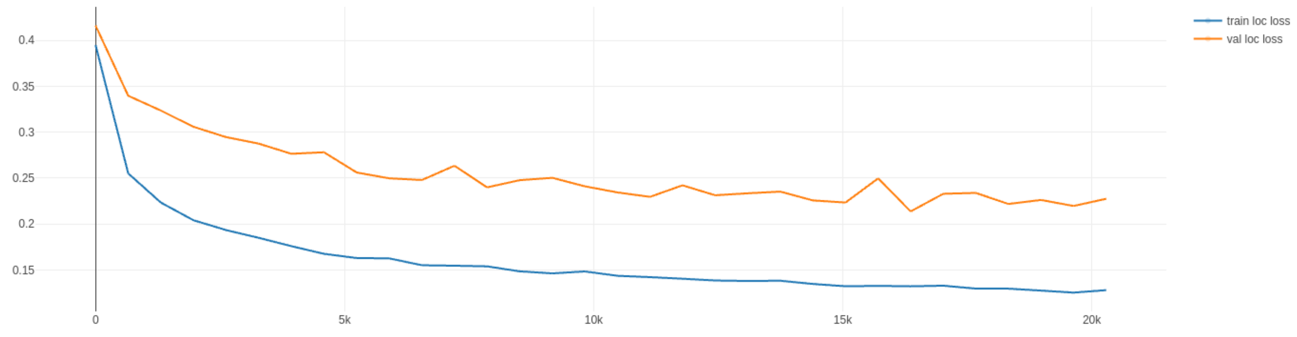
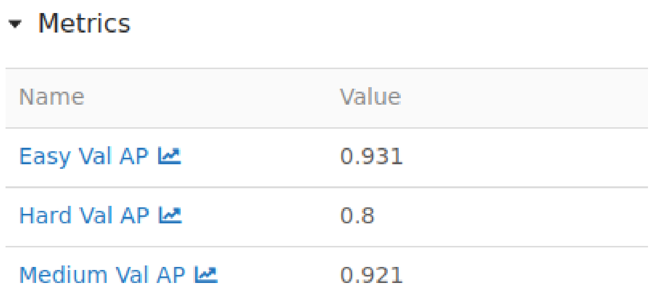


Figure 22: Training and evaluation loss tracking during training using the MLFlow

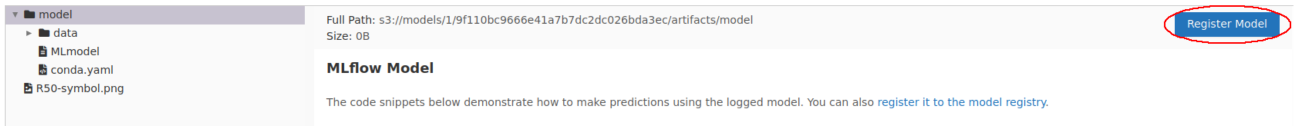
Once the training is completed, the evaluation metrics can be found on the MLFlow server. In this example we assess the resulted model using a separate evaluation script. The metrics were calculated on the test set and they are logged into the server.



Metrics

Name	Value
Easy Val AP	0.931
Hard Val AP	0.8
Medium Val AP	0.921

Finally, if the performance of the trained model is sufficient, the user has the option to register it with a tag, indicating its phase/stage (i.e., staging, production, etc.). The following image shows how to insert a trained model into the MLFlow Model Registry.

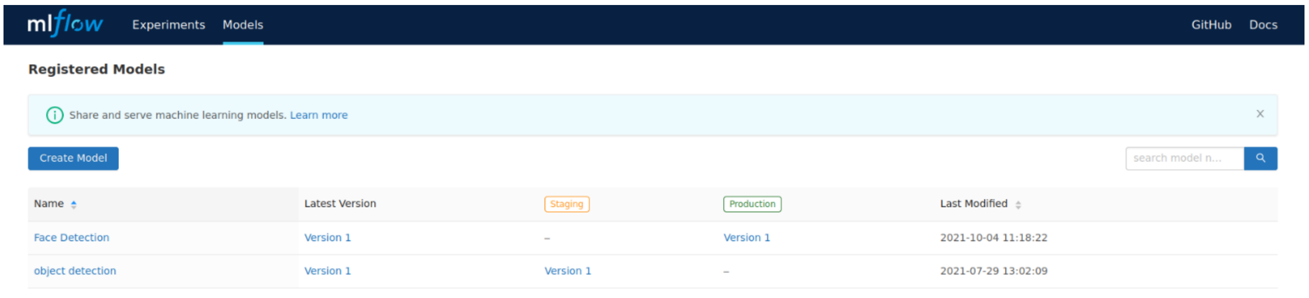


Full Path: s3://models/1/9f110bc9666e41a7b7dc2dc026bda3ec/artifacts/model
Size: 0B

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry.

Then the user can access the *"Models"* section of the dashboard and see all the registered models along with some details regarding their phase, the current registered version and the date of the last modification (see the following image).



Registered Models

Share and serve machine learning models. Learn more

Name	Latest Version	Staging	Production	Last Modified
Face Detection	Version 1	–	Version 1	2021-10-04 11:18:22
object detection	Version 1	Version 1	–	2021-07-29 13:02:09

Figure 23: MLFlow dashboard

4.3 Results

4.3.1 Image classification

Given the above Cifar-10 preparation, we present the performance of the learning algorithms reported in Section 3 under a range of data distributions from identical to non-identical. The experiments are conducted utilising NVFLARE's virtual nodes in a single machine to reduce models' transmitted time and thus training time.

Central learning

The Cifar-10 dataset is located in a single node and the model is trained for 25 epochs. Fig. 24 reports the classification accuracy of the centralised training model (acc = 0.8275).

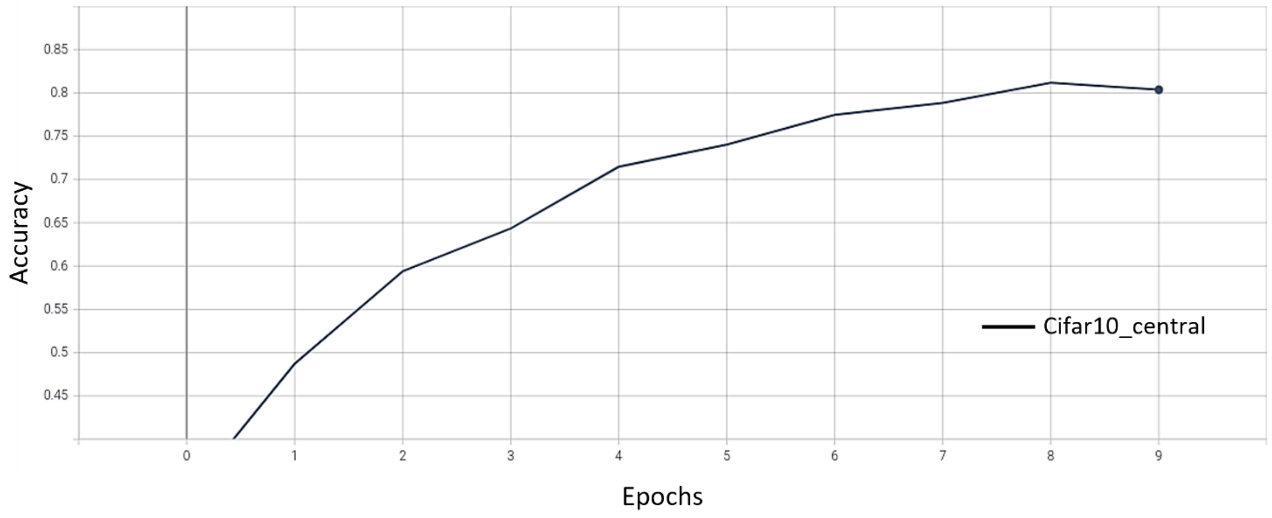


Figure 24: Centralised learning performance

FedAvg Learning with non-identical participants

By running this experiment, we want to report the difference in the global model's performance between the central and the distributed training, utilising the FedAvg, the simplest aggregation method, for different heterogeneity values. As mentioned above, the parameter $\alpha > 0$ controls the identicalness among participants. We tried different α values, where with $\alpha \rightarrow \infty$, all participants have identical distributions and $\alpha \rightarrow 0$, each participant has examples from only one class.

The participants are 10 and the number of the local epochs and the communication/aggregation rounds is also 10, since we want a similar number of iterations across participants as in the central baseline above. We experiment with 4 values for α [1, 0.5, 0.3, 0.1] to generate populations that cover a spectrum of identicalness.

Fig. 25 shows the classification performance as a function of the Dirichlet parameter α (smaller α creates less identical data distributions). The test accuracy is close to central baseline for $\alpha = 1$ and significantly drops for lower α when participants have very diverse data distributions. This result indicates that the simple FedAvg algorithm is not capable of handling non-identically distributed, highly heterogeneous datasets.

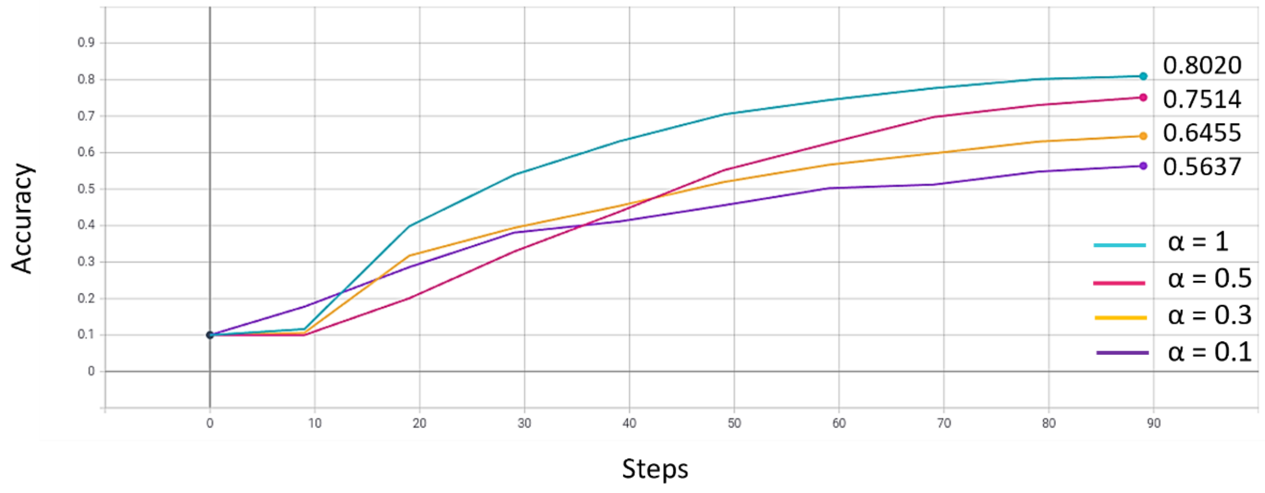


Figure 25: FeAvg learning curves for different α

Advanced Learning Strategies with non-identical participants

As presented above, more heterogeneous data splits require more advanced FL algorithms. We conducted experiments utilizing FedProx and FedOpt learning strategies to investigate their impact on global model's performance and converge under high data distribution heterogeneity. Therefore, we set α to 0.1. The other training parameters are the same as in the FedAvg learning experiment. Fig. 26 shows that FedProx achieves similar performance to FedAvg while FedOpt outperforms FedAvg by a large margin with the same α setting and the same number of training steps. FedOpt achieves that by utilizing Stochastic Gradient Descent algorithm with momentum for global model update, as described in Section 3.

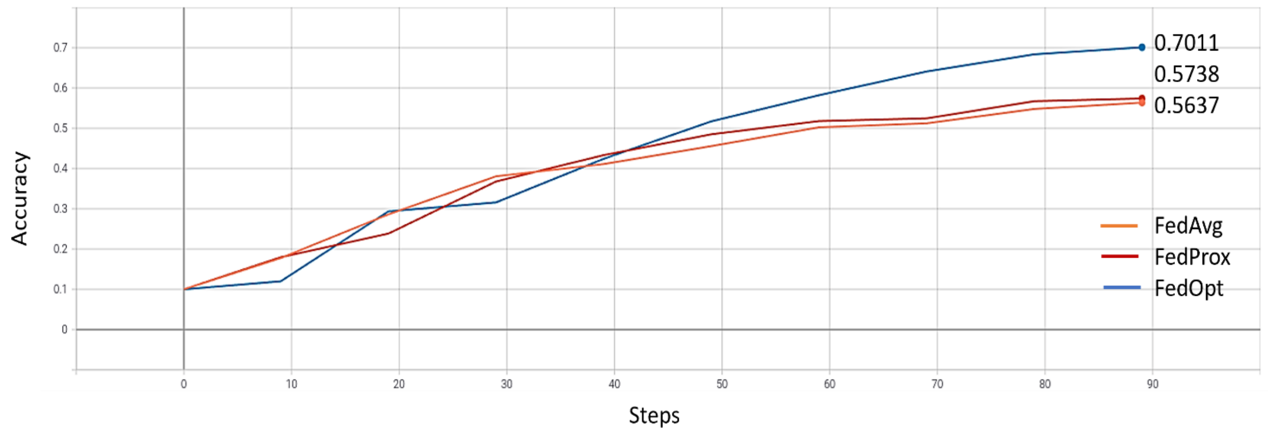


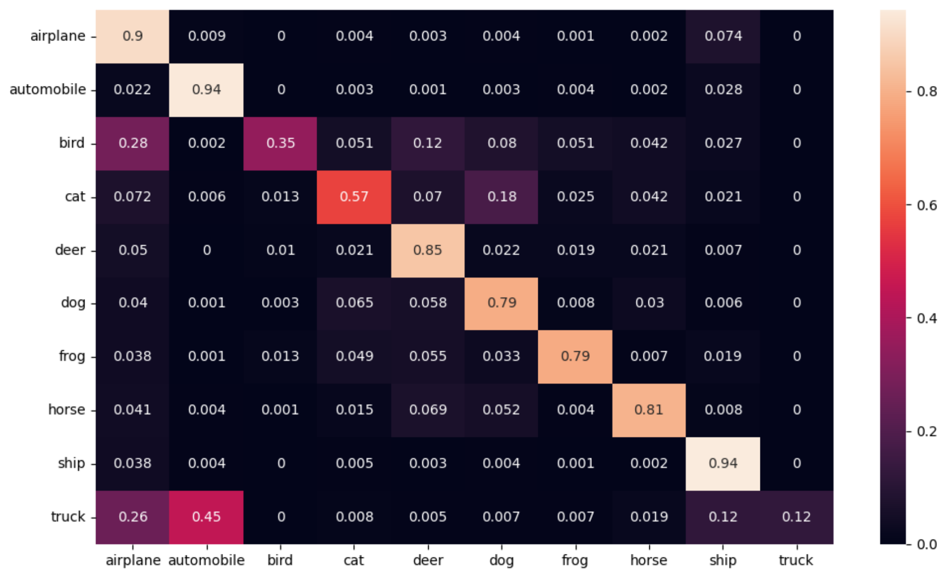
Figure 26: FedAvg, FedProx, FedOpt learning curves for α = 0.1

Utilizing the FedProx algorithm we conducted a set of experiments with multiple α values to highlight the added value of the FL paradigm compared to the isolated trained models.

Table 5 shows the dataset distribution with medium heterogeneity by setting α = 1, for 2 random participants. The participants have samples for most of the classes, with some classes only having few samples. Fig. 27 and 28 present the confusion matrices for participants 0 and 1, evaluating participants' models that are trained only with the local datasets (Table 5). As we can see both participants achieve acceptable performance for most classes, except for sample deprived classes, like truck for participant 0 and airplane for participant 1.

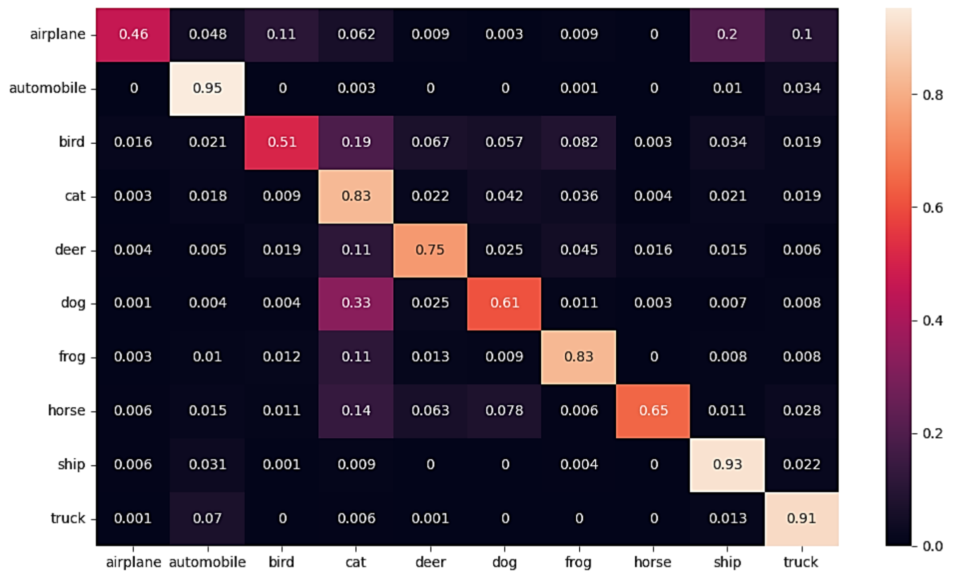
Class/ Participants	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
0	1940	433	126	193	840	191	299	243	1284	**0**
1	**22**	733	**86**	786	494	508	876	154	535	209

Table 5: Closer look at clients' data distribution for medium (a =1) heterogeneity variation



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.9	0.009	0	0.004	0.003	0.004	0.001	0.002	0.074	0
automobile	0.022	0.94	0	0.003	0.001	0.003	0.004	0.002	0.028	0
bird	0.28	0.002	0.35	0.051	0.12	0.08	0.051	0.042	0.027	0
cat	0.072	0.006	0.013	0.57	0.07	0.18	0.025	0.042	0.021	0
deer	0.05	0	0.01	0.021	0.85	0.022	0.019	0.021	0.007	0
dog	0.04	0.001	0.003	0.065	0.058	0.79	0.008	0.03	0.006	0
frog	0.038	0.001	0.013	0.049	0.055	0.033	0.79	0.007	0.019	0
horse	0.041	0.004	0.001	0.015	0.069	0.052	0.004	0.81	0.008	0
ship	0.038	0.004	0	0.005	0.003	0.004	0.001	0.002	0.94	0
truck	0.26	0.45	0	0.008	0.005	0.007	0.007	0.019	0.12	0.12

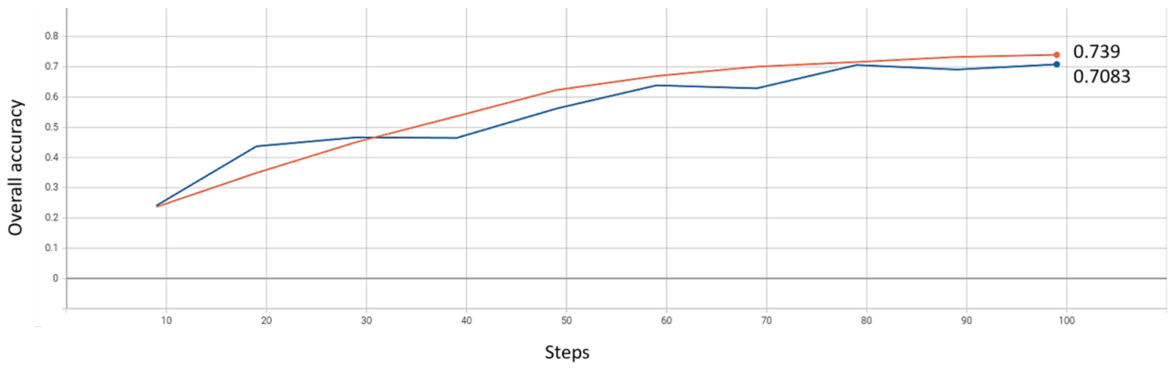
Figure 27: Participant 0 – Confusion matrix



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.46	0.048	0.11	0.062	0.009	0.003	0.009	0	0.2	0.1
automobile	0	0.95	0	0.003	0	0	0.001	0	0.01	0.034
bird	0.016	0.021	0.51	0.19	0.067	0.057	0.082	0.003	0.034	0.019
cat	0.003	0.018	0.009	0.83	0.022	0.042	0.036	0.004	0.021	0.019
deer	0.004	0.005	0.019	0.11	0.75	0.025	0.045	0.016	0.015	0.006
dog	0.001	0.004	0.004	0.33	0.025	0.61	0.011	0.003	0.007	0.008
frog	0.003	0.01	0.012	0.11	0.013	0.009	0.83	0	0.008	0.008
horse	0.006	0.015	0.011	0.14	0.063	0.078	0.006	0.65	0.011	0.028
ship	0.006	0.031	0.001	0.009	0	0	0.004	0	0.93	0.022
truck	0.001	0.07	0	0.006	0.001	0	0	0	0.013	0.91

Figure 28: Participant 1 – Confusion matrix

Fig. 29 shows the performance difference between the locally trained and globally aggregated model for medium heterogeneity variation (α = 1). Although the local modes achieve high classification accuracy, the aggregated model manages to surpass both of them.



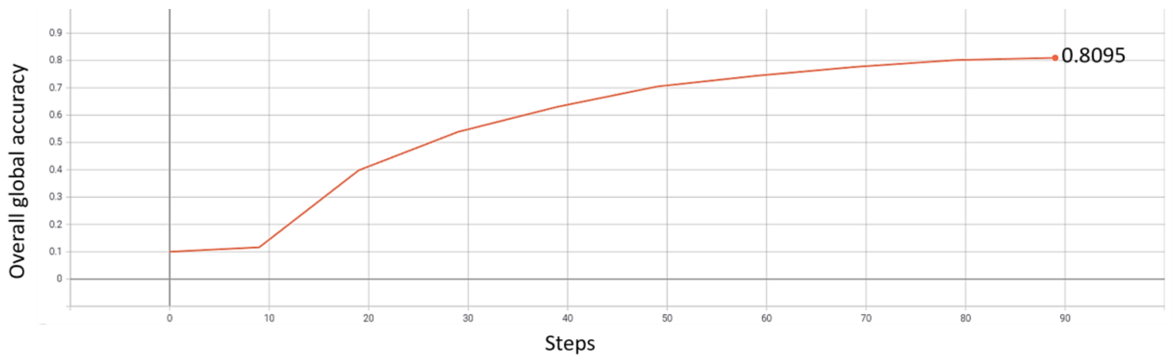


Figure 29: Data heterogeneity has affected the global performance compared with the centralized training (acc = 0.8275) but is significantly better than the local trained model performance.

We repeat the previous experiments with higher level of data heterogeneity by seeting α = 0.1. As shown in Table 6, participants tend to have significant number of samples from one or two classes and few or no samples for the other classes. Fig. 30, 31 indicate that the locally trained models achieve very poor performance for most of the classes and they tend to be biased, since visibly prefer predicting the classes that they presented at training.

Class/Participants	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
0	**0**	814	212	3735	**0**	2737	**0**	**0**	**0**	**0**
1	46	291	**0**	**4**	**0**	**0**	**0**	**0**	401	123

Table 6: Closer look at clients' data distribution for high (a =0.1) heterogeneity variation

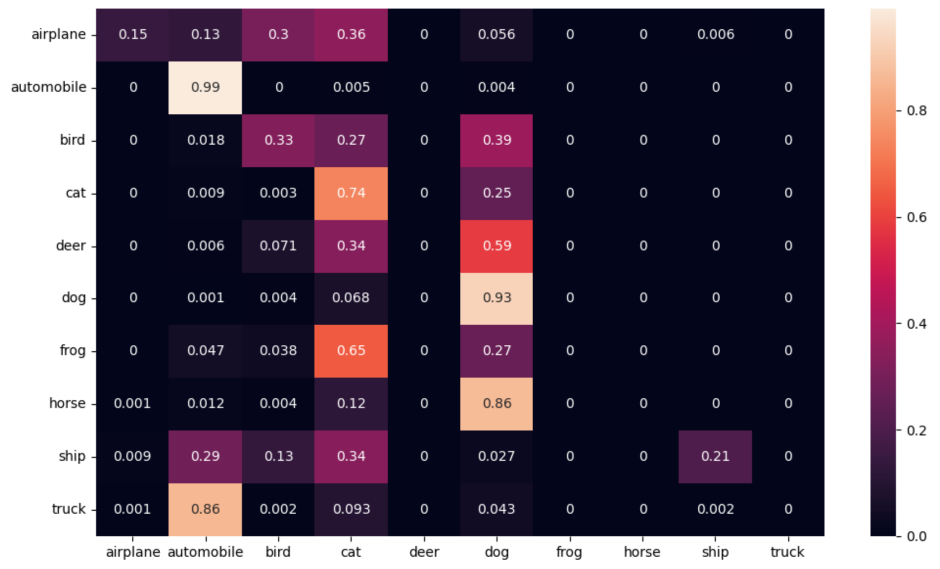


Figure 30: Participant 0 – Confusion matrix

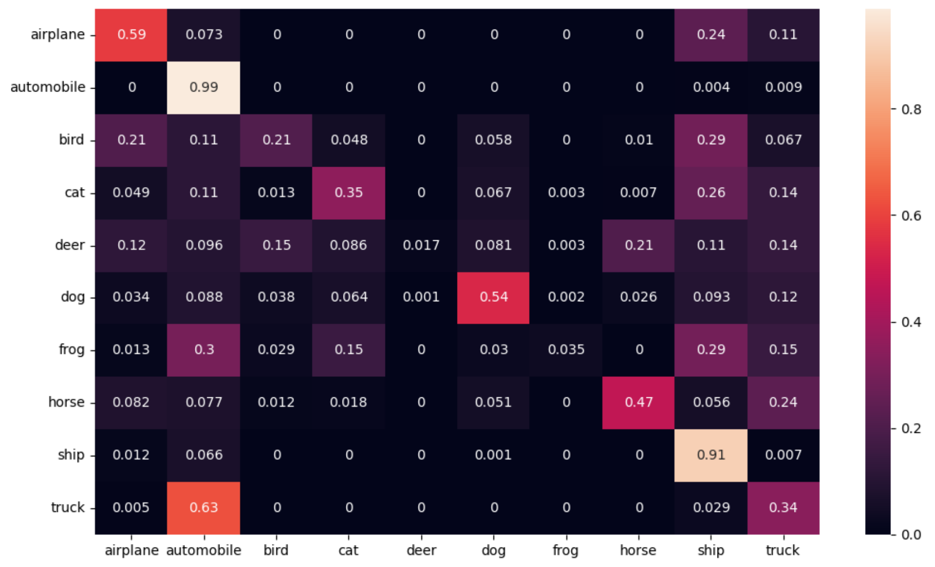


Figure 31: Participant 1 – Confusion matrix

As in the previous experiments we have the comparison between the locally trained and globally aggregated model. As shown in Fig. 32 both local models achieve very low overall local classification accuracy close to 30 % and 50 % respectively. In contrast the global model achieves high accuracy, almost 70 %. The result cleary highlights the added valued of the FL paradigm.

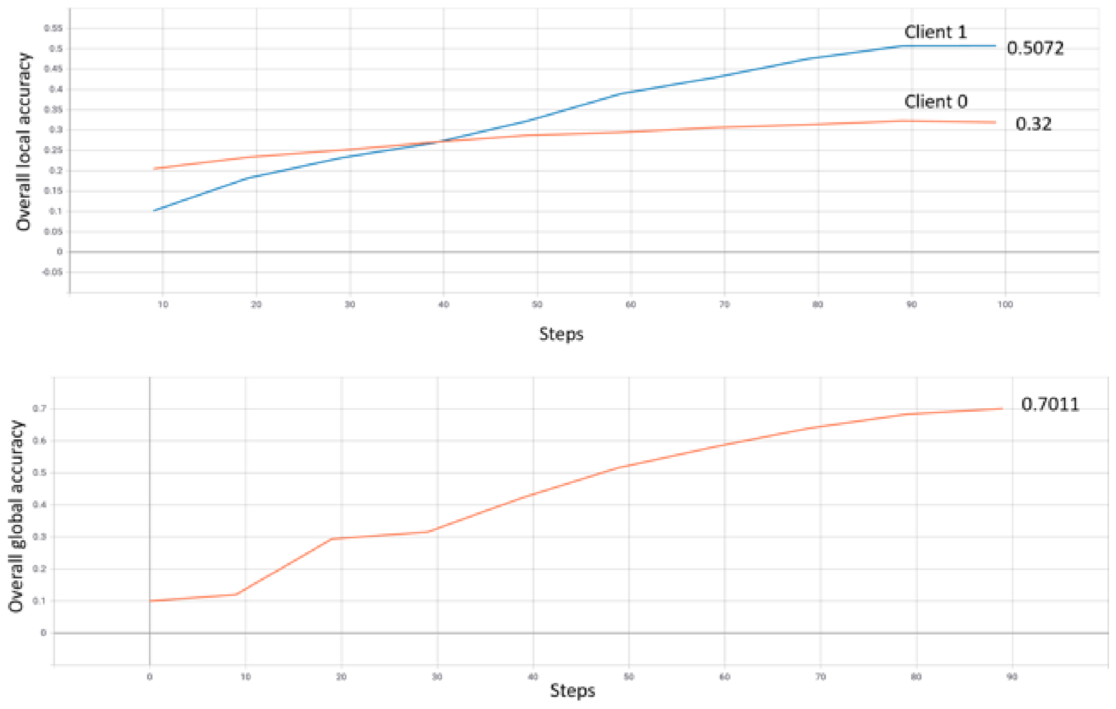


Figure 32: Data heterogeneity has affected the global performance compared with the centralized training (acc = 0.8275) but is significantly better than the local trained model performance.

4.3.2 Named entity recognition

In this subsection, we present the performance of different NER model trainings at the FL setup for different splitting strategies. The experiments are conducted utilising NVFLARE's virtual nodes in a single machine to reduce models' transmitted time and thus training time.

Central learning

In the centralized learning experiments, the dataset is located in a single node and the NER model is trained for 10 epochs. Fig. 33 shows the F1 Score of the centralised training model. The target F1 metric is 85,57%.

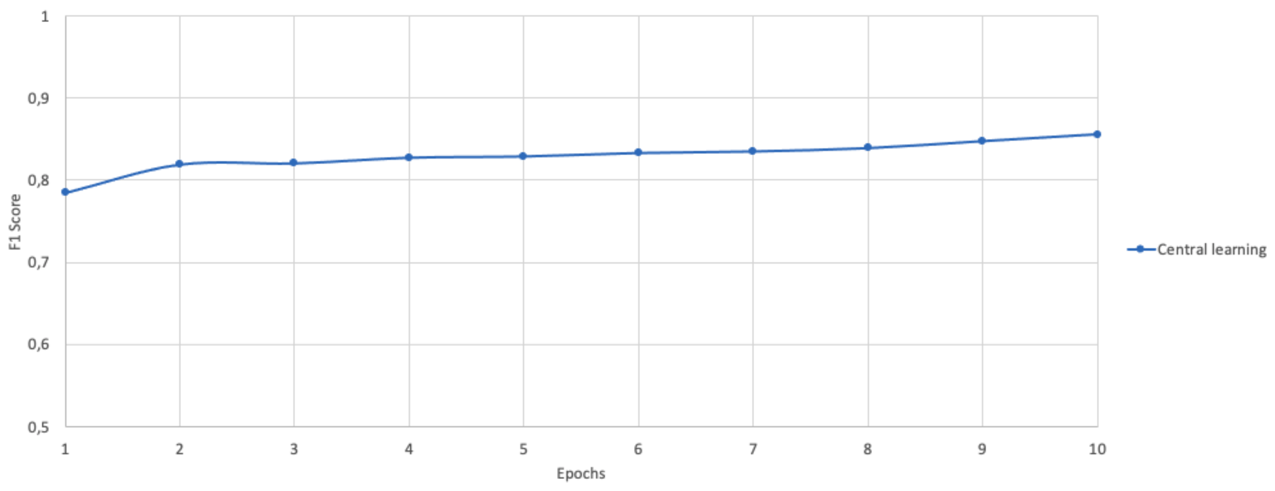


Figure 33: Centralise learning performance

FedAvg Learning with identical and non-identical participants

Initially, the training dataset is equally split among 10 participants, therefore each client contains 2034 training samples.

Afterwards, the training set is unequally split among clients. Specifically, it was split in three different ways, which are mild (split #1), moderate (split #2) and intense (split #3). Table 7 presents the corresponding training dataset of each client for the different splits.

Client	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
split										
Equal split	2034	2034	2034	2034	2034	2034	2034	2034	2034	2034
split #1	1999	2078	2077	1996	2034	1996	2067	2004	2118	1976
split #2	1717	2267	2356	1689	2234	2167	1689	2195	2289	1742
split #3	1035	3080	2075	1569	2895	809	898	3997	2459	1528

Table 7: Number of training samples per client for different data partition strategies

For a better understanding of the distribution of the training sets among clients, the Fig. 34 is given. It is observed that the Split 1 contains almost the same number of training samples as in the case of equal splitting, the split 2 slightly differs, while the split 3 greatly diverge from the original splitting.

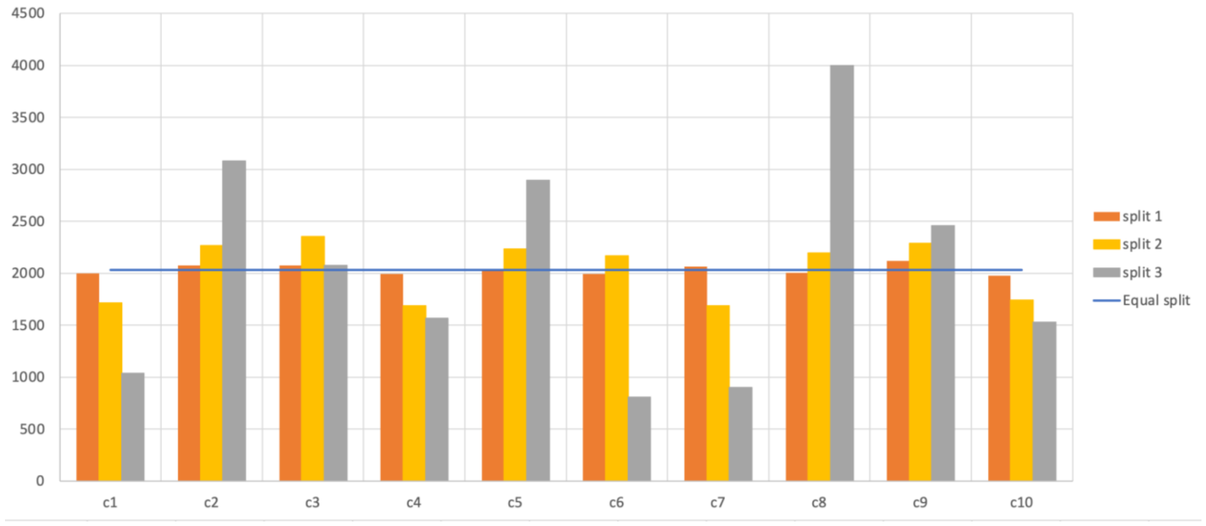


Figure 34: Number of training samples per client for different data partition strategies

Fig. 35 illustrates the F1 Score for the four different experiments, which regard the data splitting. In the case of split 1, the F1 Score is close to the one achieved for the heterogeneous splitting. The performance of federated models gets worse when clients have diverse data distributions. Therefore, it is observed that the FedAvg algorithm is not capable of handling highly heterogeneous datasets, as it is also concluded in Section 4.3.1 (with Fig 25) .

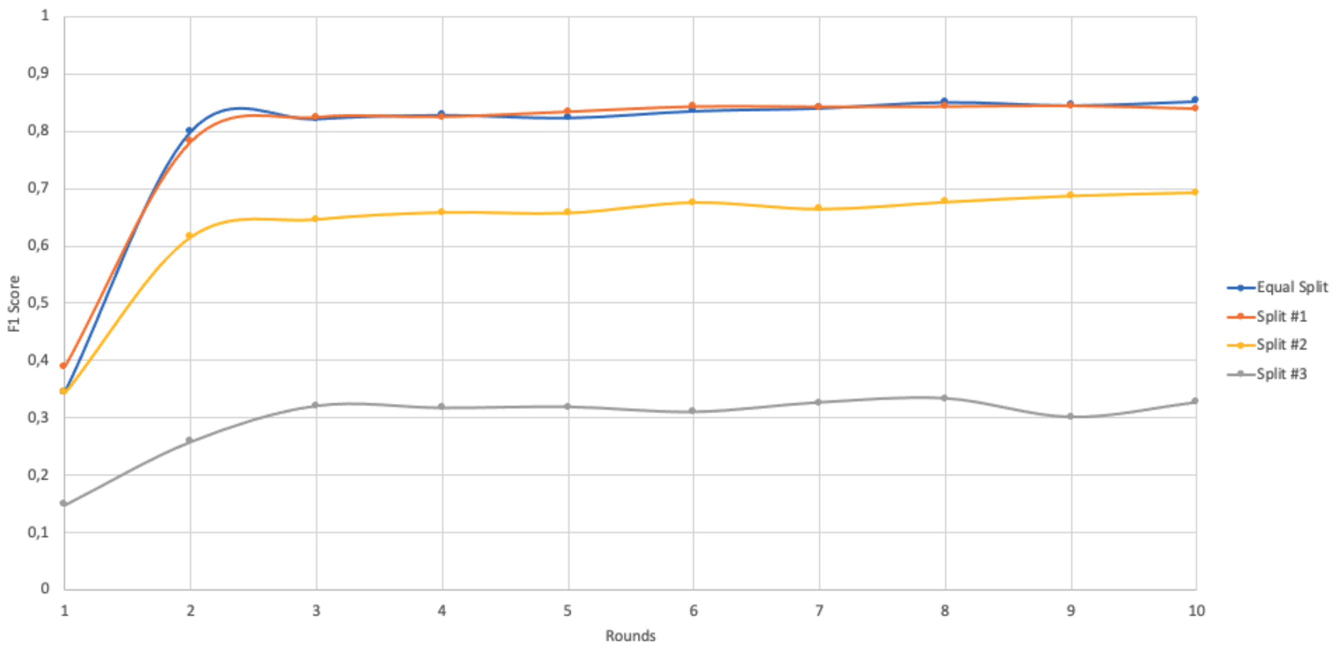


Figure 35: FedAvg F1 score curves for different data partition strategies

Examining the behaviour of the federated model, which is trained at split 2 training dataset, we can compare the overall F1 Metric values of the FL model, that arises from the Secure Aggregation, and the corresponding values of each client. Indicatively, the Fig. 36 shows the performance of the global model as well as of client 1 and client 7 for each federated round. It is observed that the global FL model performs better than the local models at each federated round.

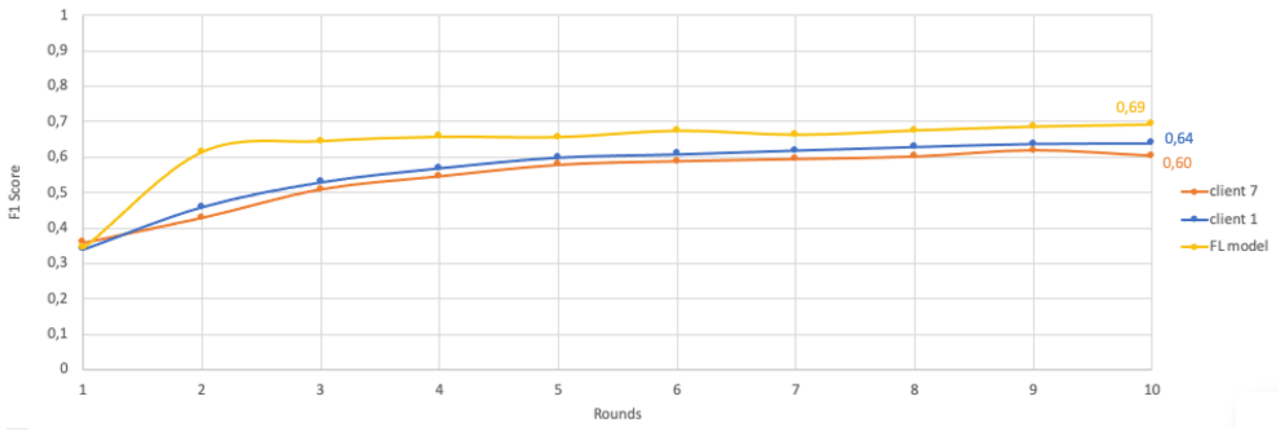


Figure 36: Local vs Global performance for two random clients

5. Conclusion

5.1. Summary

Within this task, we have designed, developed and evaluated a set of Federated Learning strategies under different learning scenarios. We have extended the insightful analysis of Federated Learning frameworks carried out on D5.1, providing requirements, definitions, architectures and training schemes for the federated learning framework, while also providing a comprehensive survey of existing works on this subject. We discuss how the federated learning framework can be applied to various existing datasets successfully followed by a complete evaluation of state-of-the-art aggregation mechanisms. Further to this, a tool for experimentation, reproducibility, deployment and a central model registry has been integrated.

5.2. Evaluation

The proposed aggregation mechanisms have been developed and carefully selected according to the requirements . Preliminary results of the application of the proposed solution to the WP4 tools have been demonstrated to the end-users third co-creation workshop of GRACE. NVIDIA Flare framework has been selected for the GRACE system since it has been shown to cover the majority of the project's requirements related to FL topology, FL type of clients and FL training protocols.

5.3. Future work

Performance evaluation will be completed in the last task of WP5 Task 5.5 Federated Learning System Analysis, which will include an overall assessment of the GRACE Federated Learning system. In addition, we aim to further support the adaptation of WP4 tools to the Federated Learning setting, while also supporting all phases of the pilot preparation and execution.

ANNEX I - GLOSSARY AND ACRONYMS

Term	Definition / Description
AI	Artificial Intelligence
CI/CD	Continuous Integration\ Continuous Delivery
CVAT	Computer Vision Annotation Tool
DL	Deep Learning
DNN	Deep Neural Networks
FL	Federated Learning
GAN	Generative Adversarial Networks
HPC	High-Performance Computing
LEA(s)	Law Enforcement Agency (s)
ML	Machine Learning
MS/s	Member State/s
NER	Named Entity Recognition
NLP	Natural Language Processing
PoC	Proof of Concept
RL	Reinforcement Learning
UI	user interface
VM	Virtual Machine