SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Enabling Client Model Heterogeneity for Serverless Federated Learning Using Knowledge Distillation

Pulkit Khera

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Enabling Client Model Heterogeneity for Serverless Federated Learning Using Knowledge Distillation

# Unterstützung von Clientmodell-Heterogenität für serverloses föderiertes Lernen mithilfe von Wissensdestillation

| | |
|---|---|
| Author: | Pulkit Khera |
| Supervisor: | Prof. Dr. Michael Gerndt |
| Advisor: | M.Sc. Mohak Chadha |
| Submission Date: | 15.05.2023 |

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.05.2023 Pulkit Khera

# Acknowledgments

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of my master's thesis.

Firstly, I would like to thank my advisor Mohak Chadha for his guidance, invaluable support, and patience throughout the research and implementation process. He provided me with valuable feedback and suggestions at all the various stages and also arranged the adequate computing infrastructure required for the success of this project.

Importantly, I would like to thank my supervisor Prof. Dr. Michael Gerndt for providing me with the opportunity to work on this interesting project and for the assistance and support throughout the whole process.

Finally, I would like to thank my family and friends for their constant support and encouragement which have been invaluable in keeping me inspired throughout my studies.

# Abstract

Recently, Federated Learning(FL) has emerged as a promising approach for training machine learning models on distributed data, while preserving privacy and ownership. In FL, multiple clients can collaboratively train a model without providing access to their sensitive private data sets to a central server. FL clients train their models locally and perform periodic weight updates to the server which are then aggregated and afterward, the global parameters are distributed back to the clients at the end of each round. Common challenges of traditional Infrastructure-as-a-Service(IaaS) based federated learning systems are less resource efficiency and high infrastructure costs/management due to idle resources. The serverless computing paradigm eliminates these issues by automatically provisioning resources and scaling on a pay-per-use basis. Therefore, previous work proposed a serverless federated learning framework that supports multiple commercial and self-hosted Function-as-a-Service (FaaS) providers as clients integrating the benefits of serverless computing into federated machine learning. However, it requires each client to have the same model architecture since it is based on global parameter averaging techniques but this is not possible in practice due to system heterogeneity among the clients resulting in their inability to agree on a global model architecture. Moreover, due to statistical data heterogeneity among clients, they must have the independence to choose model architectures optimized for their private data distribution but also learn from other clients at the same time. Finally, these averaging techniques require client weight transfer to a central server which can lead to privacy issues. Our work mainly focuses on extending the ability of this framework to enable heterogeneous client models using Knowledge Distillation(KD) which addresses these problems while obtaining similar performance results. We analyze existing IaaS-based federated KD algorithms and finally implement two algorithms i.e. FedMD[38] and FedDF[42] in the current FaaS-based framework based on their performance and ease of integration along with other optimization-related enhancements. We evaluate the system in a distributed setting with 100 FaaS clients(having heterogeneous model architectures) based on accuracy performance across different data heterogeneity levels, execution duration, and FaaS costs for various learning tasks. We achieve accuracy levels similar to our predecessor FedLesScan[17] across all learning tasks along with an average **3.5x** speed-up in the FedMD pretraining process and a **76.7%** execution time reduction for FedDF aggregation compared to sequential execution times.

# Contents

**Acknowledgments** **iii**

**Abstract** **iv**

**1 Introduction** **1**
- 1.1 Problem Definition . . . 4
- 1.2 Methodology . . . 5
  - 1.2.1 Research Questions . . . 5
  - 1.2.2 Approach . . . 6
  - 1.2.3 Evaluation and experiments . . . 6

**2 Background** **8**
- 2.1 Federated Learning . . . 8
  - 2.1.1 Limitations of Federated Learning . . . 9
- 2.2 Serverless Computing . . . 10
  - 2.2.1 Function-as-a-Service (FaaS) . . . 10
- 2.3 Knowledge Distillation (KD) . . . 11
  - 2.3.1 Distillation Process . . . 12
  - 2.3.2 Basic Distillation Example . . . 12
  - 2.3.3 Benefits of Knowledge Distillation . . . 12

**3 Related Work** **14**
- 3.1 Serverless Federated Learning . . . 14
- 3.2 Knowledge Distillation in standard FL . . . 18
  - 3.2.1 In-Depth Explanation - FedMD[38] and FedDF[42] . . . 20
- 3.3 Model Architecture Agnostic Serverless Federated Learning . . . 24

**4 Enabling Model Heterogeneity Using Knowledge Distillation** **25**
- 4.1 Central Controller Enhancements . . . 25
- 4.2 Tunable Data Heterogeneity . . . 26
- 4.3 Intelligent Client Selection for Heterogeneous Architectures . . . 26
- 4.4 FedMD Integration . . . 29
  - 4.4.1 Training Workflow . . . 29

4.5 FedDF Integration . . . . . 32
4.5.1 Ensemble Distillation for Model Fusion . . . . . 32
4.5.2 Training Workflow . . . . . 34
4.6 Overall System Design Improvements . . . . . 35
4.6.1 Optimized FedMD Pre-Training using Ray . . . . . 35
4.6.2 FedDF Parallel Aggregation using FaaS . . . . . 37

**5 Experiments and Evaluation** **38**
5.1 Experiment Setup . . . . . 38
5.1.1 Overall Evaluation Procedure . . . . . 38
5.1.2 Datasets and Heterogeneous Client Data Distribution . . . . . 39
5.1.3 Client Model Architectures . . . . . 43
5.1.4 Hyperparameters . . . . . 45
5.1.5 Infrastructure and Experiment Setup . . . . . 48
5.2 Learning Task Performance and Accuracy . . . . . 48
5.2.1 FedMD . . . . . 49
5.2.2 FedDF . . . . . 54
5.3 Time and Cost Analysis . . . . . 56
5.3.1 FedMD . . . . . 58
5.3.2 FedDF . . . . . 60
5.4 Quantitative comparison - FedMD and FedDF . . . . . 63

**6 Conclusion and Future Work** **66**

**Abbreviations** **68**

**List of Figures** **69**

**List of Tables** **71**

**Bibliography** **72**

# 1 Introduction

In recent years, with the availability of increased amounts of data and computing power, Machine Learning (ML) has become ubiquitous with applications in nearly every field and it continues to be on a similar growth trajectory at least for the near future.ML has revolutionized the way we process, analyze and interpret large amounts of data. It has led to breakthroughs in the fields of image recognition, speech recognition, natural language processing, and many other applications. Overall, recent progress in ML has led to significant advances in fields including healthcare, finance, transportation, and many others [51]. Currently, the most common method to train these ML models is to train a single centralized model with access to the complete data set to be used for the training process. This centralization allows for a more efficient and streamlined approach to ML due to relatively simple algorithms and central resource management. However, there are several issues associated with centralized ML such as data privacy concerns, security risks, scalability limitations, bias, and lack of transparency [4]. Data is often dispersed among multiple sources, such as individual devices, institutions, or organizations. Federated Learning (FL)[33] is a relatively new ML model training technique that tackles these issues and provides similar results as compared to the centralized training methods. In contrast to centralized ML, FL enables distributed data to be used for training machine learning models without any need of centralizing the data.

Federated Learning(FL) is a new paradigm that allows for distributed training of ML models across multiple devices while maintaining data privacy and security. The general idea is multiple parties train local models on their private data sets and exchange parameters to train a global model shared by all the parties with robustness to non-independent and identically distributed (i.i.d.) data samples among the participating clients. This approach has numerous advantages, including reduced communication costs, improved scalability, and increased data privacy[57]. In FL, clients can be mobile, edge devices, or virtual machines managed by Infrastructure-as-a-Service (IaaS) providers however, all these traditional FL systems can immensely benefit from a new computing paradigm called serverless computing as proposed by Chadha, Jindal, and Gerndt in FedKeeper[11].

As the technology sector is migrating towards the cloud due to high availability and easy resource access and management, serverless computing is one of the most widely used cloud computing models [52] in which the cloud provider manages and dynamically provisions the resources required for running processes completely abstracting away the back end infrastructure-related details from the user. It is offered by most of the big cloud providers in the form of Function-as-a-Service (FaaS) platforms such as AWS Lambda [6], Google Cloud Functions [14], and Microsoft Azure Cloud Functions [7] on a pay-per-use basis. Open-source frameworks such as OpenFaaS [43] and OpenWhisk [48] are also available to deploy such functions on managed clusters.

Using FaaS technologies for FL as proposed in FedKeeper [11] leads to improvements in resource efficiency and cost since FaaS clients can scale down to zero instead of waiting for other clients to complete their local training process in contrast to IaaS based systems that stay idle incurring costs and resources during the complete process. These idle times are further increased due to differences in clients' computing capabilities and data heterogeneity.

Grafberger et. al [23] proposed FedLess as an evolution of FedKeeper in which more FaaS providers were added to the existing system along with other features such as client authentication and authorization, Local Differential Privacy(LDP) for model weights, and a lightweight process controller for faster convergence.

In the existing FedLess[23] framework, there is an implicit assumption that all the clients need to have the same ML model architecture to train a global model through weight averaging however a practical FL system is affected by different types of client level issues that are difficult to address with this particular assumption:

- **Computational Heterogeneity**: In the real-world federated learning setting, FL clients can vary from edge devices to fully-fledged GPU-enabled systems and each of these clients may have different memory, computing, and storage capacities[40]. As a result, it is not possible for each participating client to agree on a single global model architecture. Even if a client model architecture is really simple due to computational constraints, it might contain crucial private data that can be useful for the convergence of the global model.

- **Statistical Data Heterogeneity**: In a distributed FL system, it is a common scenario for clients to have non-IID data distributions which greatly impact the accuracy of the global model as demonstrated in figure 1.1 which shows the negative impact of non-i.i.d. data distribution on the FedMD[38] federated learning algorithm for the MNIST and CIFAR-10 image classification task. A smaller value of Dirichlet alpha indicates a higher level of non-i.i.d. distribution among the clients as also demonstrated in section 5.1.2. We observe an accuracy drop of about **50%** for MNIST and **10%** for the CIFAR task between alpha values of *100* and *0.1*. This happens mostly because the final models do not generalize well after the model aggregation process due to variance among the trained local client models. Also, it is important for each participating client to have the independence to choose their own model architecture specifically tailored to their private distribution but also benefit from other collaborating clients at the same time.

- **Privacy**: The current system involves federated averaging of the client model weights on a central aggregator. However, federated learning clients can be distributed across various different organizations and every organization might not be willing to share personal model architecture details and parameters specific to their private datasets as they can be used for inferring the original training data[25, 12, 38]. Although the existing FedLess system provides Local Differential Privacy(LDP) mechanism to prevent such attacks, it can not fully guarantee protection as long as client parameters are being shared with a centralized server. Even though this issue does not directly demand client model heterogeneity, it comes as an added benefit with the knowledge distillation techniques that we will be using to enable model heterogeneity.

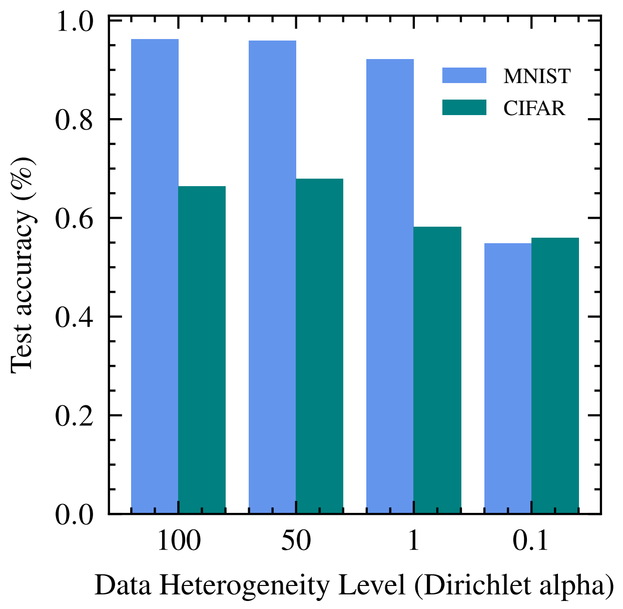


Figure 1.1: Impact of increasing data heterogeneity on the test accuracy of the FedMD distributed learning algorithm for MNIST and CIFAR image classification tasks

Therefore this assumption of homogeneous architectures is not always possible in practical scenarios. To allow all the devices to contribute as well as benefit from the training process, we need the ability for clients to have heterogeneous model architectures and this brings us to an active area of research known as Knowledge Distillation (KD) first introduced by Hinton et. al [27]. We choose knowledge distillation to enable model heterogeneity since it allows a high level of flexibility in the choice of client model architectures in contrast to other approaches like parameter decoupling [3] that only allows flexibility in particular layers of the overall architecture. Additionally, knowledge distillation-based approaches are more communication efficient in federated learning since they usually involve the transfer of only the prediction logits to the aggregation server instead of complete model parameters.

KD is a technique used in ML to transfer knowledge from a large, complex model (teacher model) to a smaller and more efficient model (student model). The goal is to improve the performance of the student model by allowing it to learn from the teacher model's knowledge. These distillation techniques can be applied in the FL setting and enable client model heterogeneity since it does not include a direct exchange of model weights among the client and teacher models.

Most KD techniques such as FedMD[38], DS-FL[29], MHAT[28] comprise a series of steps such as private training, prediction on public datasets, and aligning local models to these logits. Due to the synchronous nature of these algorithms, the central controller cannot move to subsequent steps until all the participating clients finish a particular step which results in a lot of resources being idle and incurring unnecessary costs for every round. Additionally, due to system heterogeneity and data imbalance, the variation in step execution times for each client increases further leading to even more idle time and resource wastage. However, implementing these algorithms using FaaS eliminates this issue because of the auto-scaling capabilities which can scale down resources to zero during these idle times and bring the execution costs significantly down for such algorithms through its pay-per-use billing model.

Thus, the focus throughout this work would be to build upon the existing FedLess[23] FaaS based FL framework and extend it to enable client model heterogeneity while obtaining similar training results and execution costs compared to the existing framework. We explore various KD techniques in the federated learning setting based on their ability to support client model heterogeneity and data heterogeneity among the participating clients and enhance the current FedLess architecture to integrate them.

## 1.1 Problem Definition

In server-based standard FL techniques, the same model architecture is used by the FL clients and server which is usually difficult in practice because of computation capacity limitations on the clients as explained in the previous section. Knowledge

Distillation in federated learning tackles this issue by providing flexibility to accommodate heterogeneous model architectures for the clients by following a student-teacher paradigm.

We evaluate these existing state-of-the-art KD algorithms based on their performance as well as the ease of integration into the current FedLess system architecture. We want to make use of some of these approaches and develop an extension of FedLess that allows for client model heterogeneity with at least similar or better performance than the current FedLess system as well as the IaaS-based counterparts of these approaches in terms of accuracy, convergence time, and execution cost. Also, statistical data heterogeneity among the heterogeneous client architectures is a common scenario that occurs in real-world distributed learning settings which has been addressed in this work.

Out of all the KD techniques mentioned in table 3.1, we only consider those which allow client model heterogeneity since that forms the basis of the problem we want to address. In addition, we want the technique to be robust towards non-i.i.d. data distribution among the participating clients and able to perform well on all types of tasks and not be limited to a specific domain for better generalization of our implementation.

## 1.2 Methodology

In this section, we provide an overview of the research objectives, structure, and milestones throughout the project. We also provide a summary of the evaluation criteria that we use.

### 1.2.1 Research Questions

- How can knowledge distillation techniques enable client model heterogeneity in a Serverless FL setting? [Main Question]
- What effect does client model heterogeneity have on the accuracy of such a system?
- What effect do different levels of client data heterogeneity have on the accuracy of such a system?
- How much FaaS execution time and cost is required for such a system?

### 1.2.2 Approach

Firstly, we conduct an in-depth analysis of the currently published IaaS-based approaches and their implementations. We finally boil down to two best-suited IaaS approaches FedMD[38] and FedDF[42] based on their performance and the ease of integration into the current system. We implement the shortlisted approaches from the previous step along with further optimization of the existing processes by making use of serverless computing demonstrated in the upcoming chapter 4. In terms of changes to the overall FedLess system, we incorporate the following enhancements:

- Support for heterogeneous client model architectures across all strategies.
- Tunable data heterogeneity to create synthetic non-i.i.d. data distributions across participating clients for experimentation.
- Extended version of the FedLesScan[17] client selection algorithm that incorporates heterogeneous client model architectures in the clustering process.
- Most importantly, the optimized implementation of **FedMD**[38] and **FedDF**[42] algorithms in our serverless federated model training framework with the following main optimizations:
  - FedMD pre-training using the Ray[49] serverless distributed training framework.
  - Fast parallel aggregation for FedDF using multiple FaaS aggregators.

### 1.2.3 Evaluation and experiments

We evaluate the system on a variety of computer vision and Natural Language Processing (NLP) tasks, heterogeneous client architectures, and varying levels of data heterogeneity among the clients. First, we conduct an in-depth accuracy performance analysis for both FedMD and FedDF at various data heterogeneity levels, then we shift our focus towards the FaaS execution time and incurred costs individually for each algorithm. Finally, we perform a quantitative comparison between both algorithms in terms of accuracy, execution time, and FaaS costs.

Overall, our experiments show that we are able to achieve similar accuracy levels and even better for language modeling tasks across both algorithms in a standard i.i.d. data distribution setting when compared to the performance of our predecessor FedLesScan[17]. In the case of extreme non-i.i.d. distribution, FedDF demonstrates much more robustness yielding performance accuracy similar to FedLesScan on both MNIST as well as Shakespeare tasks in contrast to FedMD which results in an average drop of **46.5%** test accuracy on the MNIST dataset for the extreme data heterogeneity case. In terms of system optimizations, we were able to achieve an average speed-up of **3.5x** for the FedMD pre-training process using the Ray platform. We also achieved a decrease of **76.7%** in the aggregation process execution time for FedDF. Finally, in terms of overall experiment execution duration and FaaS costs, FedMD had an average execution time of **193 minutes** costing **$13.24** whereas FedDF had an execution time of **191.5 minutes** costing **$5.59** on average. Consequently, FedDF outperforms FedMD in terms of accuracy performance, especially in extreme cases of data heterogeneity, and has a similar FaaS execution time but is relatively cheaper in terms of invocation costs.

# 2 Background

This section introduces some key concepts and technologies that are being used throughout this project. Section 2.1 defines federated learning, its benefits, and its limitations. Then we move on to section 2.3 which dives into Knowledge Distillation that enables the main objective of this project which is to client model heterogeneity in a serverless federated learning setting.

## 2.1 Federated Learning

Federated learning is an emerging machine learning paradigm that enables training ML models without requiring the centralization of data. It allows various data owners such as institutions and organizations to collaboratively learn a shared model by exchanging model parameters, instead of raw data. This approach has the potential to address the privacy and security concerns associated with centralized machine learning systems.

It was first introduced by McMahan et al. [33] at Google and was integrated into GBoard (Google Keyboard)[45] on the Android mobile operating system across millions of heterogeneous mobile phones where federated learning processes the typing history on-device to suggest improvements to the next iterations of the query suggestions model.

Typical federated machine learning algorithms mainly comprise the following steps as shown in Figure 2.1:

1. **Model initialization:** The central server initializes the machine learning model with some pre-defined parameters.

2. **Local model training:** Client devices perform local training on their private data. The local training updates the model's parameters using the private data that is available on the device. The device then sends the updated parameters back to the centralized server. Usually, a fraction of clients are selected for each local training round and this can be done using selection schemes like FedLesScan [17] which uses a clustering-based approach for client selection.

3. **Model Aggregation:** The central server aggregates the updated parameters from all the client devices and calculates a new set of parameters for the model.

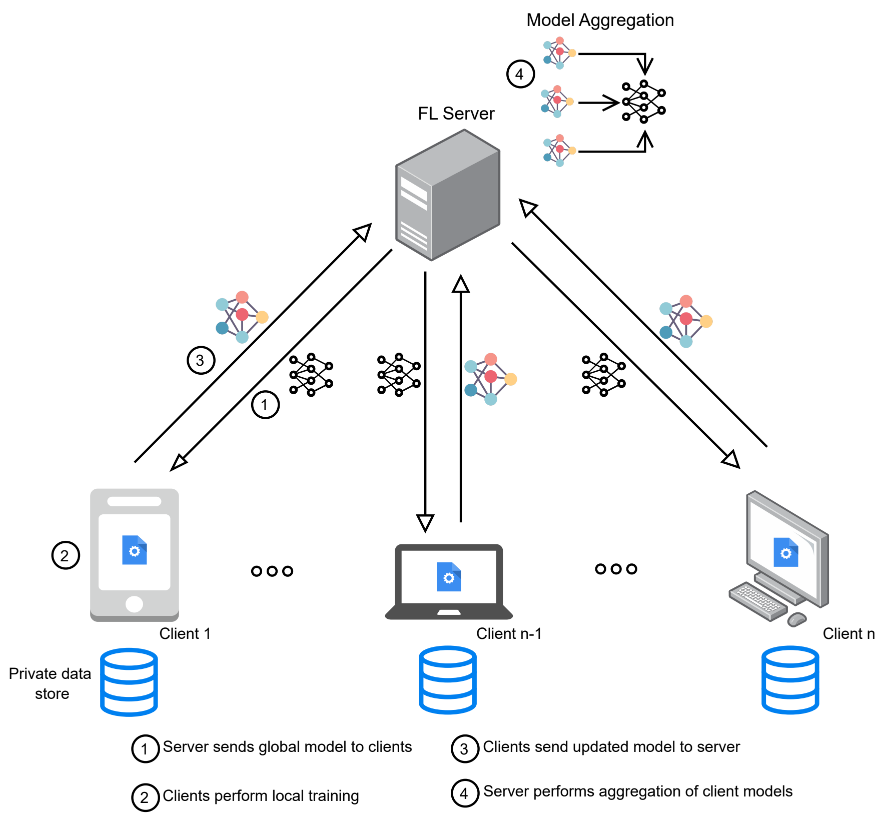

Model Aggregation
4
FL Server
3
1
2
Client 1
Client n-1
Client n
Private data store
1 Server sends global model to clients
2 Clients perform local training
3 Clients send updated model to server
4 Server performs aggregation of client models

Figure 2.1: Standard FL setup

This process is done using aggregation algorithms such as Federated Averaging(FedAvg)[53].

4. **Client Model Update:** The updated parameters are then sent back to each client device for further local training, and the process repeats until the desired accuracy is achieved.

### 2.1.1 Limitations of Federated Learning

Two major challenges relevant to this project in federated learning are device and data heterogeneity[39]. Since the training is distributed across a set of devices, each device has different computing capabilities and access to different data samples which usually results in a non-i.i.d. distribution of samples among clients. For instance, a smartphone may have very limited processing power compared to a desktop which makes it challenging to train the same model on both these devices.

In the case of data heterogeneity, clients have access to different distributions of data as well as varying numbers of training data samples depending on the storage capacity. This can lead to biased models that do not generalize well to the entire data population. There has been a lot of research on this side to tackle the problem of data heterogeneity in federated learning such as [44] where they use data augmentation to tackle data heterogeneity for inducing equal or greater performance.

## 2.2 Serverless Computing

A relatively new cloud computing paradigm is *serverless computing* which has gained a lot of popularity. It is a cloud computing application execution model in which the complete underlying infrastructure and resource management is abstracted away from developers by the cloud providers so that they can focus on the main application logic and code without having to deal with any server management and other operational tasks. Another major benefit of the serverless model is that the developers pay only for the resources that are actually used. The cloud provider automatically scales up and provisions more computing resources when required and can scale them back to zero when demand decreases. This means that developers are only billed according to the exact execution time and the computing resources used during that period of time which has significant cost benefits. All the major cloud providers such as AWS, GCP, and Azure are moving their services toward the serverless model with function-as-a-service (FaaS) being at the core of this paradigm shift.

### 2.2.1 Function-as-a-Service (FaaS)

One of the most popular and widely used ways to implement a serverless computing model is Function-as-a-Service (FaaS). It allows developers to write code in the form of a stateless event-based microservice function and then these individual functions can be deployed in the cloud with the help of containers. Once deployed, these functions can be triggered via HTTP requests and also through other connected cloud services that the provider offers such as message queues both in a synchronous and asynchronous way. Since this is an event-driven function execution model, the incoming events are monitored and these functions are auto-scaled based on certain triggers like the frequency of incoming requests, CPU usage, and many more depending on the use case. Below are some of the most popular FaaS providers that are being widely used in the industry:

- Amazon - AWS Lambda [6]
- Google - Google Cloud Functions(GCF) [14]

- Microsoft - Azure Functions [7]
- OpenFaaS *(Open Source)* [43]
- Apache OpenWhisk *(Open Source)* [48]

In terms of applications, FaaS is suitable for any cloud application that has repetitive event-driven behaviors such as microservices for web applications and in Internet of Things (IoT) for streaming data processing tasks since several individual functions can also be combined to create complex processing workflows.

## 2.3 Knowledge Distillation (KD)

KD is a technique used in ML to transfer knowledge from a larger, more complex teacher model to a smaller, simpler student model without losing validity. It was first introduced by Hinton et. al [27] for neural networks. Since then, it has been used for many applications such as improving generalization performance through self-distillation[58], learning with noisy data[2], and transfer learning[56]. For this work, the key motivation behind using KD is its ability and flexibility to accommodate personalized heterogeneous model architectures. Based on whether both the teacher and student models are updated or not, distillation techniques can mainly be placed into 3 categories as mentioned in [22]:

1. **Offline Distillation:** This is the most common type of knowledge distillation[27, 50] in which knowledge is transferred from a pre-trained model to a student model. Firstly, the complex teacher model is trained on some training samples and then knowledge is extracted from this model in the form of logits or intermediate features which is in turn used to guide the training process of smaller student models. Therefore, only the student models are updated during the distillation process. However, one drawback of this distillation type is that we need a large-capacity high performing complex teacher model.
2. **Online Distillation:** In this type, both the teacher and the student models are updated simultaneously and therefore it overcomes the issues of offline distillation. The training in this case is completely end-to-end and can be efficiently parallelized. Some interesting works in this area are [24, 1, 59].
3. **Self-Distillation:** In this category, the same networks are used as student and teacher models. The main idea is to use the predictions of the trained model as target values for the retraining process iteratively. Some interesting works are [20, 46].

### 2.3.1 Distillation Process

A lot of research has been performed on various ways to distill knowledge between models as shown in table 3.1. A very basic idea as mentioned in [27] to transfer the generalization ability of the complex teacher model to a small model is to use class probabilities (from the softmax layer) produced by the teacher model as "soft targets" for training the smaller model. For this transfer, the same training set can be used or it can be done using a separate transfer set that may be labeled or unlabelled depending on the algorithm. Since probabilities close to zero have very less impact on the cross entropy loss function, alternatively class logits (output before softmax) from the teacher model can be used as the "soft" targets with a standard L2 loss for optimizing the smaller models.

### 2.3.2 Basic Distillation Example

In this section, we take a look at a basic distillation method from [27]. Equation 2.1 represents a "softmax" output layer that converts the output logits $z_i$ into probability $p_i$ where $T$ is known as the temperature. Temperature controls the entropy of the output distribution and higher temperatures lead to softer output distributions which means that bigger logits are penalized more than smaller logits.

$$p_i = \frac{exp(y_i/T)}{\sum_j exp(y_j/T)} \tag{2.1}$$

In a simple form of distillation, the distilled student model is trained on a transfer set and using a soft target distribution for each case in the transfer set that is produced by the larger teacher model with a high softmax temperature. The same high temperature is used when training the distilled model, but after it has been trained it uses a temperature of 1 during inference.

### 2.3.3 Benefits of Knowledge Distillation

Some important uses of KD and its relevance are mentioned below:

1. **Model Compression:** Although deep and complex neural networks might have very high learning capabilities, it is not always the case that these capabilities are being utilized completely. Consequently, it makes sense to transfer this learned knowledge to smaller and simpler student models to reduce computational complexity without a significant performance loss for deployment on simpler and less powerful hardware.

2. **Model Heterogeneity:** Since in KD, we do not exchange weights directly between the teacher and student models to transfer learned knowledge but rather rely on the output class logits. Therefore this enables separate model architectures for the teacher and student models. We will be using this property of KD in this project extensively.
3. **Better Generalisation:** As we distill knowledge from larger teacher models, it can lead to an increased generalization performance in the smaller student models since the larger models are usually trained on larger data sets along with complex features. This improves the student model's accuracy and reduces overfitting.

# 3 Related Work

In this chapter, we first explore the current literature in the field of serverless federated learning in section 3.1 with a major focus on the architecture and in-depth working of FedLess[23], FedKeeper[11] and FedLesScan[17] since it is the baseline of works on which we are building in this work. Then in section 3.2 we provide an overview of client model architecture agnostic federated learning techniques where we first focus on knowledge distillation-based methods and some other approaches. Finally, we look into works combining client model heterogeneity and serverless federated learning which is also the main focus of this project.

## 3.1 Serverless Federated Learning

Serverless computing mainly refers to a cloud computing model where the cloud provider manages the infrastructure and automatically allocates computing resources to event-based processing functions. This saves developers, the hassle of infrastructure management and only pays for the computing resources that they are using while also being highly scalable. A relatively new direction of research in federated learning is the integration of serverless capabilities into FL which mainly helps in reducing idle resource costs and easier infrastructure management.

Kotsehub et al. proposed FLoX[34] which is based on the funcX[19] serverless computing platform where they try to decouple FL model training/inference from infrastructure management and enable users to easily deploy FL models on heterogeneous and distributed compute endpoints. Their workflow involves a "controller" and several participating computing devices. The controller is responsible for selecting eligible training devices from an available device pool, transmitting the training task along with the global model to these devices, and then receiving and aggregating the trained local model weights from each device for the next round using FedAvg[53] method. An interesting aspect of this work is lightweight endpoints with experiments on various low-powered edge devices such as the Raspberry Pi. However, this framework is not generalized and is built specifically on the funcX serverless platform.

Jayaran et. al[30] proposed $\lambda$-FL, which is a serverless aggregation method for federated learning. They aim to reduce resource wastage which is a common bottleneck in most FL processes and have better fault tolerance. They treat the aggregators as

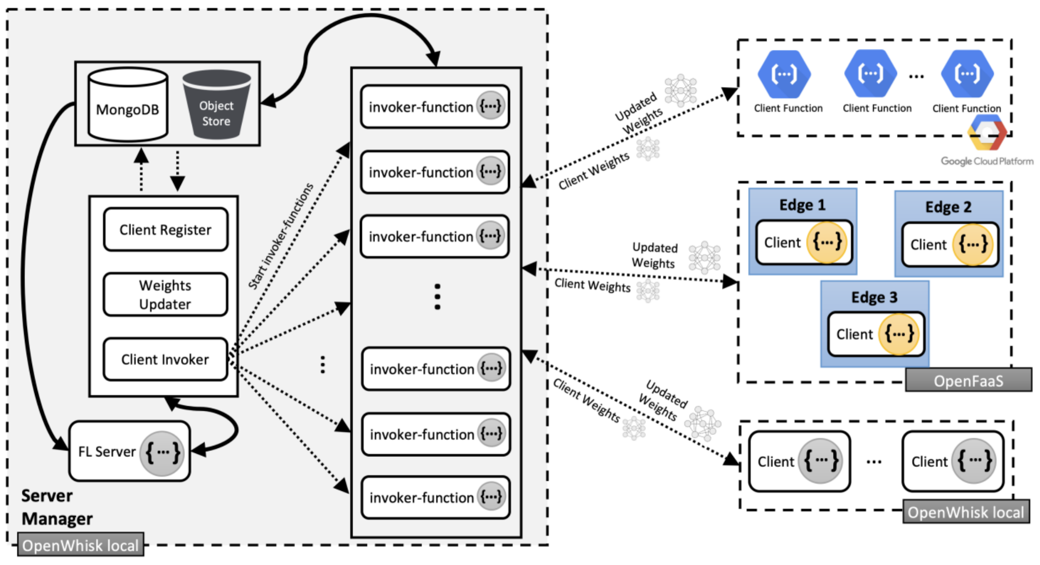


Figure 3.1: Original architecture of FedKeeper [11]

serverless functions without any state where every aggregator acts on a particular sequence of inputs and produces a single output. Since the approach depends on hierarchical result aggregation by multiple serverless aggregator instances, the aggregation operation has to be associative. They split this associative aggregation operation into two types of serverless functions- Leaf Aggregation Function and Intermediate Aggregation Function. First, they fuse the raw gradient updates from a fixed number of parties to obtain a partially aggregated model update by simple averaging of the gradients using leaf aggregation functions. Then intermediate aggregation functions perform a further aggregation of the partially aggregated model to produce the final aggregated model. In terms of implementation, they use the topic-based message queues, Kafka for party-aggregator communication. Each FL job has a TaskID identifier and two queues TaskID-Agg and TaskID-Parties are created at the start of the process. Once the job starts, the aggregator publishes the initial model on the TaskID-Agg queue which can be downloaded by parties to download and start their training. Once local training is complete, parties push their model to TaskID-Parties at end of each round. After every few updates of TaskID-Parties, an aggregation function gets triggered for the leaf aggregation process and pushes the result back to the Kafka queue triggering other intermediate aggregate functions for the final aggregation result. This work shows a promising research direction toward the use of serverless technology in FL.

One of the first fully serverless federated learning architectures FedKeeper was

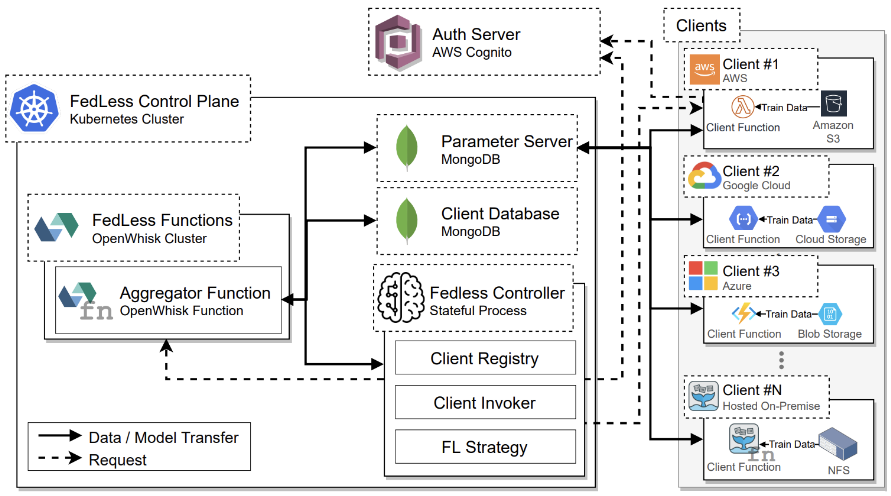


Figure 3.2: Original architecture of FedLess [23]

proposed by Chadha et. al[11] in the form of a Python framework that works over a combination of FaaS platforms for training across heterogeneous devices with unified invocation methods. It runs an OpenWhisk[48] based deployment which is responsible for automatic creation, deletion, and invocation of FL clients for various FaaS platforms and also keeps track of all FaaS functions. All these tasks are achieved by several sub-components i.e. Client Register, Weights-Updater, Client-Invoker, and the FL-Server as individual OpenWhisk functions. Client-Register is a local Mongo database instance for storing client invocation details. FL-Server initializes the model to be trained along with the required training hyperparameters and creates a configuration file for each client. This configuration file is used by Client-Invoker for the invocation of FL clients through their URL with a 1-1 mapping between invoker and client functions. Once clients return updated weights to the Client-Invoker, it triggers the Weights-Updater which aggregates the client weights and stores them in the local object store and then the FL-Server repeats the whole process until convergence for the specific task. System architecture showing all these components can be seen in figure 3.1.

FedLess was proposed by Grafberger et al.[23] which was an evolution of the existing FedKeeper[11] architecture as shown in figure 3.2. It has multiple enhancements in terms of support for more FaaS platforms, authentication, and authorization of clients along with privacy-protecting training mechanisms. From the architectural perspective, FedLess supports four commercial and two open-source FaaS platforms. Its main component is the control pane which runs on a Kubernetes cluster consisting of a Parameter Server(MongoDB), a Client Database(MongoDB), an OpenWhisk-based central

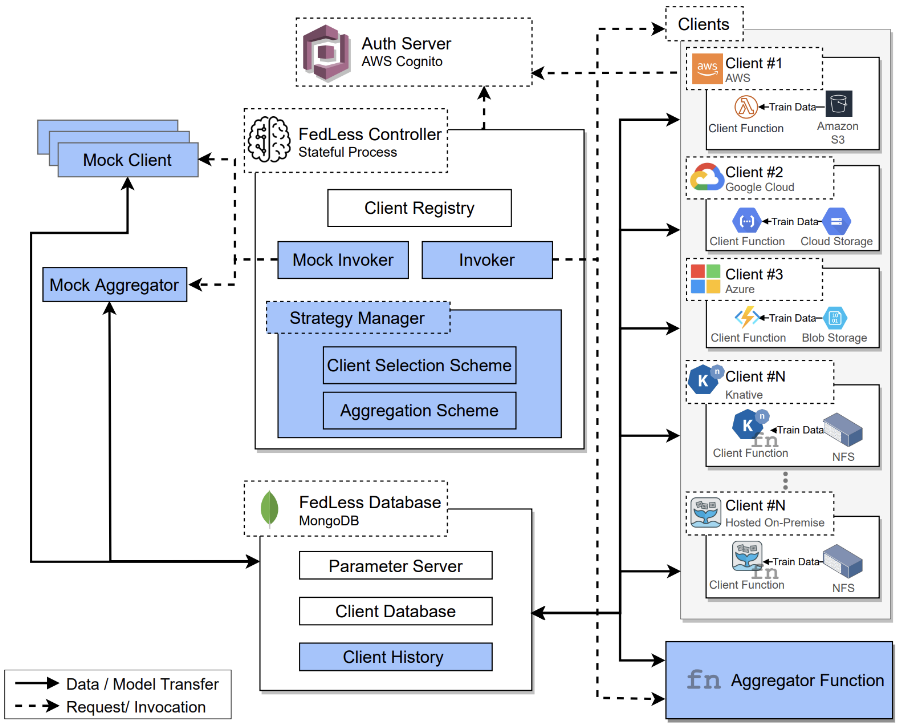


Figure 3.3: Modified architecture of FedLess[11] for FedLesScan[17]

aggregator function, and the FedLess Controller. Parameter Server is responsible for storing the global model weights, hyperparameters, and architecture whereas the Client Database stores client-specific information such as invoke URL, their hyperparameters, dataset locations, and any other information supplied by the administrator for the client to execute successfully. The stateful controller manages the complete training cycle and is responsible for client selection and invocation of the client as well as aggregator functions. The aggregator function is invoked by the controller once all participating clients have finished their local data training and it uses FedAvg[53] to aggregate the client model parameters and pushes it into the Parameter Server. It also AWS Cognito[5] for authentication, authorization, and integration with external identity providers based on JSON Web Tokens (JWTs)[31] attached to all HTTP requests between the server and clients.

Another work FedLesScan proposed by Elzohairy et al.[17] builds on top of FedLess in which the key contribution is to mitigate stragglers in the system along with some architecture enhancements. A common issue in large-scale FL systems are stragglers which are the slower clients that affect the overall training process negatively in terms of time, resources, and accuracy of the global model. The main causes of stragglers are differences in clients' computation and communication capabilities along with the presence of unbalanced non-i.i.d. data. They propose an adaptive clustering-based client selection algorithm that selects current round clients based on their behaviors in the previous round and a staleness-aware aggregation scheme to reduce slow model updates and avoid wasted client contribution. As shown in figure 3.3, it has the following architecture extensions to the existing FedLess system with new components highlighted in blue:

- Central stateful controller does not depend on complex Kubernetes deployments and can now be run on any system as a lightweight process
- To reduce development costs and better debugging on FedLess, they implemented mock functions to test the whole system on a single local machine without deployment
- Added Strategy Manager component to the controller responsible for controlling client selection as well as selecting the aggregation strategy

Apart from these changes, the overall training process flow remains similar to FedLess.

## 3.2 Knowledge Distillation in standard FL

Several works apply distillation-based approaches to federated learning[54] as the demand for more personalized models on resource-constrained devices has increased. FL algorithms mainly use KD for[47]: (1) Enabling client heterogeneity; (2) Mitigating the impact of data heterogeneity on global model accuracy. KD based FL architectures can mainly be divided into four categories[54]:

1. **Distillation of knowledge to each FL client to learn stronger personalized models:** In these architectures, the focus is to learn client-level personalized architectures. An interesting work in this category is FedMD proposed by Li and Wang[38]. It comprises a publicly labeled "transfer" dataset and client-specific private data sets. In every round, the public dataset is used for logit-based distillation between all the clients and then each client fine-tunes its model on their private datasets. Therefore, every client obtains a personalized model while also leveraging knowledge from other clients. We have used this architecture in our work and will dive deeper in the upcoming sections.

2. **Distillation of knowledge to the FL server to learn stronger server models:** In this type, the focus is on learning stronger student server models with the help of several candidate teacher models. We will be using FedDF architecture [42] in this work as it is designed for edge clients having different model architectures and computing capacities.

3. **Bidirectional distillation to both the FL clients and the FL server:** These are also known as online distillation techniques where both the FL server and clients distill knowledge to each other within the same training procedure. Interesting state-of-the-art architectures in this paradigm include FedET[13] and FedGKT[26] that use alternative minimization techniques to train the small client models and the large server model through bi-directional distillation.

4. **Distillation amongst clients:** KD in a federated learning setting can also be performed among neighboring clients in the network such as Distributed Distillation Algorithm proposed by Bistritz et al. [8] where each client uses an unlabeled public dataset and broadcasts it's soft decisions to its neighbors for distillation.

Since the main focus of this work is client model heterogeneity, we analyze several approaches by taking two important factors into account:

1. Does it require a public data set? If yes, then labeled or unlabelled.

2. Does it allow all possible modeling tasks and architecture types?

Table 3.1 shows heterogeneous client model KD algorithms with an analysis of the above-mentioned factors. FedMD belongs to the first category of KD algorithms in which knowledge distillation occurs to each FL client in order to get stronger personalized models, it requires a carefully chosen labeled public transfer dataset and can be used to perform any learning task ranging from image to text data applications. For the second KD category, we evaluated two algorithms i.e. MHAT and FedDF which involve knowledge distillation to the server for better central server models, and both of them require an unlabelled public dataset for the distillation process. Most algorithms belonging to category three above involve alternate model training between the server and clients for every round which results in high communication costs in terms of a serverless architecture since the aggregator and client functions need to exchange parameters frequently. Similarly, category four involves knowledge distillation among all the participating clients resulting in even higher communication costs because of all the clients distilling knowledge to their neighbors. Therefore, we do not consider any distillation algorithms from categories three and four for our serverless implementation.

| Method | Needs Public Data? | KD Category | Possible Tasks |
|---|---|---|---|
| FedMD [38] | Yes, Labeled | 1 | Any |
| MHAT [28] | Yes, Labeled/Unlabeled | 2 | Any |
| FedDF [42] | Yes, Unlabeled | 2 | Any |
| FedGKT [26] | N/A | 3 | Only Image |
| FedET [13] | Yes, Unlabeled | 3 | Any |
| DS-FL [29] | Yes, Unlabeled | 4 | Any |

Table 3.1: Evaluated model heterogeneous KD techniques

Taking all these factors into account along with the robustness towards data heterogeneity among the clients, we decide to extend the FedLesScan[17] system using the **FedMD**[38] and **FedDF**[42] algorithms for enabling client model heterogeneity. From category one, we choose FedMD since it satisfies all our requirements i.e. allows client model heterogeneity and generalization to any machine learning task. Additionally, it is relatively robust to a certain level of data heterogeneity compared to other algorithms in this category. Also importantly, considering there are several synchronous steps involved during each training round for FedMD, we believe that it can greatly benefit from a serverless implementation in terms of execution costs. In the second category, we choose FedDF because of its simple yet powerful server-side knowledge distillation scheme that does not require any modifications on the client side and is very robust to heterogenous data distribution among the participating clients. FedDF does not place any restrictions on the unlabelled public dataset which can even be generated from generative networks in case of unavailability. Also, the knowledge distillation method of FedDF can greatly benefit from being parallelized using serverless functions which will result in reduced costs and execution time. We go into more depth about these algorithms in section 3.2.1 since it would be crucial for the understanding of this work.

### 3.2.1 In-Depth Explanation - FedMD[38] and FedDF[42]

In this section, we introduce the two KD-based federated learning architectures FedMD and FedDF in depth since we have used them as base algorithms throughout this work to enable client model heterogeneity. Also, both these methods can be applied to various tasks such as Computer Vision (CV) and NLP.

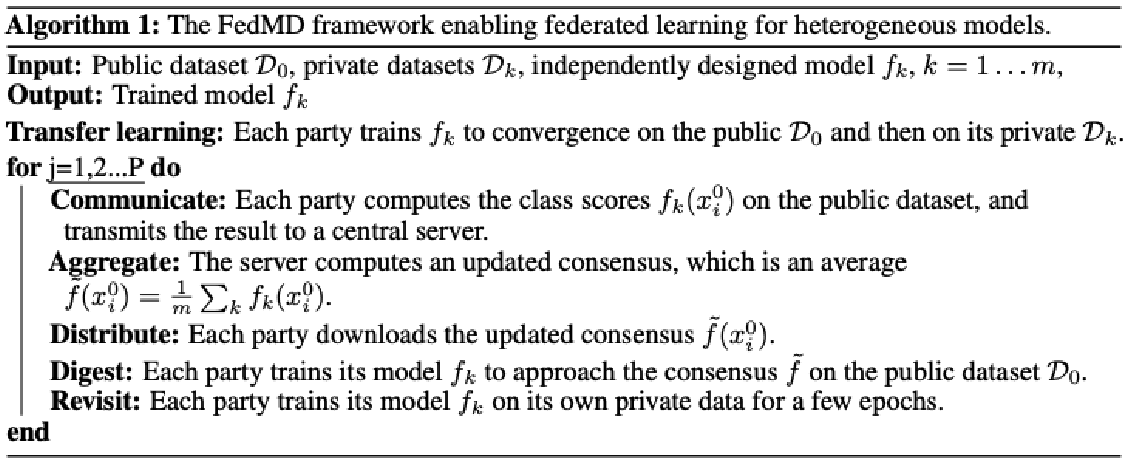
### FedMD

FedMD[38] involves the distillation of knowledge to each FL client to learn stronger personalized models. Also, each participant can design their unique model architecture and they can still benefit from collaborative distillation with the other clients. In this approach, the centralized server does not need to have any information regarding the client models and it can be considered a black box. In terms of data, there is a labeled public dataset accessible to all clients and individual private client datasets. It comprises several steps for completing each round of collaborative training as shown in figure 3.4:

1. **Initial Transfer Learning:** Each client trains on the labeled public dataset until convergence and then on its private dataset
2. **Communicate:** Each client performs a forward-pass inference on the public data samples and sends the resulting class logits to the central server
3. **Aggregate:** Server combines these class logits by averaging
4. **Distribute:** Each client downloads the averaged logits from the server
5. **Digest:** Each client trains its model using the averaged logits as soft targets on the public dataset
6. **Revisit:** Each client trains on its private dataset for a few epochs for personalization of the model

Step 1 mentioned above only takes place once initially and steps 2-6 occur repetitively for each communication round until the required client model accuracy is achieved. Some issues with this approach are the following:

- Requires an initial transfer learning step where all the clients need to be trained on the public dataset till convergence before starting the actual collaboration rounds and it is relatively time-consuming.
- Final output layer of the client model architecture needs to have a total number of output neurons equivalent to $(\#PrivateDataClasses + \#PublicDataClasses)$ since the same model needs to be trained on both the public as well as the private dataset during the complete training process and in most of the cases, both these datasets have a mutually exclusive set of classes. Therefore, the performance of the final model is affected in case there is a higher number of classes in the public dataset. Consequently, a good public dataset should have more data samples divided among less number of overall classes.



```
Algorithm 1: The FedMD framework enabling federated learning for heterogeneous models.
Input: Public dataset $\mathcal{D}_0$, private datasets $\mathcal{D}_k$, independently designed model $f_k$, $k = 1 \dots m$,
Output: Trained model $f_k$
Transfer learning: Each party trains $f_k$ to convergence on the public $\mathcal{D}_0$ and then on its private $\mathcal{D}_k$.
for j=1,2...P do
    Communicate: Each party computes the class scores $f_k(x_i^0)$ on the public dataset, and
      transmits the result to a central server.
    Aggregate: The server computes an updated consensus, which is an average
      $\tilde{f}(x_i^0) = \frac{1}{m} \sum_k f_k(x_i^0)$.
    Distribute: Each party downloads the updated consensus $\tilde{f}(x_i^0)$.
    Digest: Each party trains its model $f_k$ to approach the consensus $\tilde{f}$ on the public dataset $\mathcal{D}_0$.
    Revisit: Each party trains its model $f_k$ on its own private data for a few epochs.
end
```

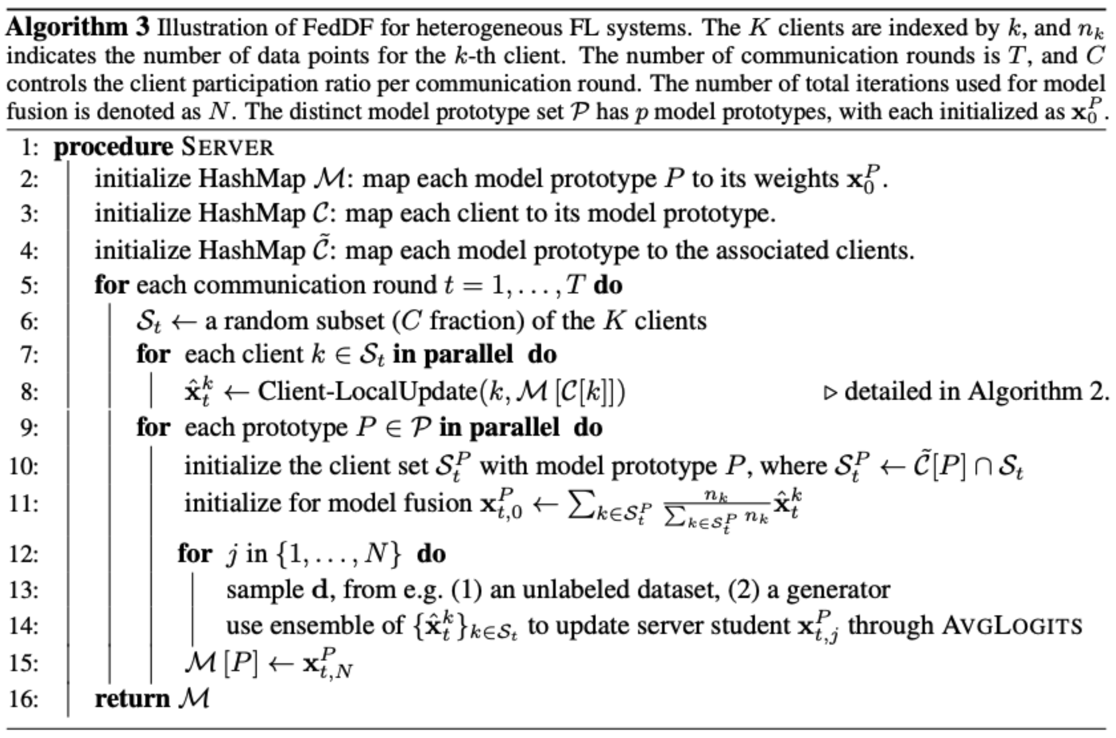
Figure 3.4: Original algorithm for FedMD[38]

**FedDF**

FedDF[42] involves learning stronger server models by distilling knowledge from several student models. It proposes an ensemble distillation approach for model fusion which supports heterogeneous client models and data. This approach solves both issues mentioned previously in section 3.2.1 as it works with unlabelled public data with can also be generated optionally by a Generative adversarial network (GAN) since it is quite robust to dataset selection.

In the case of homogeneous model architectures, an ensemble of several selected teacher models distills knowledge to a single server student model. In every round, each client performs its local model training and sends the updated weights to the central aggregation server. The aggregation server then initializes a student model using a federated averaging [53] of the client model parameters. Now, several iterations of ensemble distillation take place from the client models to the server model via the unlabeled dataset according to equation 3.1 which is one of the key contributions of this paper (Please refer [42] for details). Also, it is important to note that the complete distillation process takes place on the server side without any modifications to the regular local client training process. It comprises several steps for completion of each training round as shown in figure 3.5:

1. Three hashmaps are initialized in the beginning i.e. model-weights, client-model, and model-clients mappings which are required for the distillation process
2. A subset of random clients is selected for each communication round where they first perform their local private data training similar to FedLess.



**Algorithm 3** Illustration of FedDF for heterogeneous FL systems. The $K$ clients are indexed by $k$, and $n_k$ indicates the number of data points for the $k$-th client. The number of communication rounds is $T$, and $C$ controls the client participation ratio per communication round. The number of total iterations used for model fusion is denoted as $N$. The distinct model prototype set $\mathcal{P}$ has $p$ model prototypes, with each initialized as $\mathbf{x}_0^P$.

```
1:  procedure SERVER
2:      initialize HashMap $\mathcal{M}$: map each model prototype $P$ to its weights $\mathbf{x}_0^P$.
3:      initialize HashMap $\mathcal{C}$: map each client to its model prototype.
4:      initialize HashMap $\tilde{\mathcal{C}}$: map each model prototype to the associated clients.
5:      for each communication round $t = 1, \dots, T$ do
6:          $\mathcal{S}_t \leftarrow$ a random subset ($C$ fraction) of the $K$ clients
7:          for each client $k \in \mathcal{S}_t$ in parallel do
8:              $\hat{\mathbf{x}}_t^k \leftarrow$ Client-LocalUpdate($k, \mathcal{M}[\mathcal{C}[k]]$)        ▷ detailed in Algorithm 2.
9:          for each prototype $P \in \mathcal{P}$ in parallel do
10:             initialize the client set $\mathcal{S}_t^P$ with model prototype $P$, where $\mathcal{S}_t^P \leftarrow \tilde{\mathcal{C}}[P] \cap \mathcal{S}_t$
11:             initialize for model fusion $\mathbf{x}_{t,0}^P \leftarrow \sum_{k \in \mathcal{S}_t^P} \frac{n_k}{\sum_{k \in \mathcal{S}_t^P} n_k} \hat{\mathbf{x}}_t^k$
12:             for $j$ in $\{1, \dots, N\}$ do
13:                 sample $\mathbf{d}$, from e.g. (1) an unlabeled dataset, (2) a generator
14:                 use ensemble of $\{\hat{\mathbf{x}}_t^k\}_{k \in \mathcal{S}_t}$ to update server student $\mathbf{x}_{t,j}^P$ through AVGLOGITS
15:             $\mathcal{M}[P] \leftarrow \mathbf{x}_{t,N}^P$
16:     return $\mathcal{M}$
```

Figure 3.5: Original algorithm of FedDF[38] for heterogeneous client architectures

3. Now for each model prototype, the distillation process occurs in parallel for several iterations until the validation loss flattens out.

4. In this distillation process, we first initialize a student server model using FedAvg[53] of the participating clients' weights for the particular model architecture

5. Then all the client teacher models participating in the communication round distill their knowledge to the initialized student server model using the update step mentioned in equation 3.1

6. Finally, we obtain one trained model for each unique model architecture that contains collaborative knowledge from all the participating clients across all the model architectures

$$\mathbf{x}_{t,j} := \mathbf{x}_{t,j-1} - \eta \frac{\partial \, \mathrm{KL}\left(\sigma\left(\frac{1}{|\mathcal{S}_t|}\sum_{k \in \mathcal{S}_t} f\left(\hat{\mathbf{x}}_t^k, \mathbf{d}\right)\right), \sigma\left(f\left(\mathbf{x}_{t,j-1}, \mathbf{d}\right)\right)\right)}{\partial \mathbf{x}_{t,j-1}} \tag{3.1}$$

In the equation 3.1, KL stands for Kullback-Leibler divergence, $\sigma$ is the softmax function, and $\eta$ is the step size.

## 3.3 Model Architecture Agnostic Serverless Federated Learning

As we saw in section 3.2, there has been a lot of research with a focus on applying model-agnostic knowledge distillation to standard IaaS-based federated learning architectures. Moreover, experiments conducted for FedMD are simulated locally only for 10 participating clients without much emphasis on the infrastructure components and therefore do not provide a full picture of the algorithm performance in an actual distributed setting which involves many other factors such as system heterogeneity and stragglers. Similarly, for FedDF, there has not been any focus on the distributed infrastructure, its optimization, and execution time in such a setting and they leave this part as future work. To the best of our knowledge, we are the first ones to apply KD methods in the serverless federated learning paradigm. In this work, we present an extension of the FedLess[17] architecture that allows clients with heterogeneous model architectures to participate in the serverless federated learning process and is also robust towards data heterogeneity among the clients while yielding performance results comparable to its predecessors. We also perform rigorous execution time and cost analysis for the FedMD and FedDF algorithms in a distributed computing setting across 100 participating clients.

# 4 Enabling Model Heterogeneity Using Knowledge Distillation

As mentioned in the previous sections, we will be using existing knowledge distillation algorithms to achieve this and we decided to go ahead with **FedMD**[38] and **FedDF**[42] algorithms based on their performance and ease of integration into the existing system design. In this chapter, we will highlight the major changes and new components added to the existing base framework for achieving our final research goals.

## 4.1 Central Controller Enhancements

The central stateful controller is responsible for initializing clients and global models, executing training strategies, and monitoring the complete process. Earlier, creating the data configurations and the global model was a relatively straightforward process due to homogeneous client architectures. However, to enable heterogeneous model training, we added the capability to create and initialize heterogeneous model architectures for each client based on the input configuration. Currently, we can build and initialize the following models for each FaaS client depending on the task:

- **Convolutional Neural Networks** up to 3 layers with any dimensional specifications depending on the input size and a specific dropout rate that will be applied after each layer followed by a fully connected layer in the end for classification tasks.
- **LSTMs** up to 2 layers with configurable hyperparameters such as units, vocab size, sequence length, and embedding size followed by a fully connected layer.

In the original FedLess framework, we have a fixed set of network architectures depending on the task which is used for all the participating clients without any option of architecture-related input parameters or arguments. We have now added generic network creation functions for convolutional neural networks and LSTMs that can create arbitrary model architectures based on a specific set of variable inputs such as the number of filters in the case of convolutional networks and the number of units for LSTMs. We modified the existing YAML input configuration file structure such that these architecture-dependent variables can be specified for each participating client function.

All the network-related hyperparameters are currently being passed from the input configuration YAML file separately for each client participating in the collaborative training. These current model architectures are limited to the experimentation learning tasks in the scope of this thesis. However, we have created a modular code structure so that new architectures for other learning tasks can be added to the system with ease and without any difficulty.

Also, since distillation algorithms require a separate public "*transfer*" dataset to transfer knowledge between models, we have enhanced the client data configuration creation process to take care of all these algorithm-specific data requirements. In sections 4.4 and 4.6 we will go into more detail regarding these required datasets.

## 4.2 Tunable Data Heterogeneity

One of the major challenges in FL is non-independently and identically (non-iid) distributed data among the participating clients where each client's data reflects only a fraction of the data distributions of all the clients. This can lead to issues such as slow convergence and poor generalization performance of the FL model therefore it becomes an important factor to consider in FL benchmarks such as LEAF[9]. We enable controlling client data heterogeneity for classification tasks with the help of Dirichlet distribution as in [42]. This helps us in analyzing the behavior and robustness of training strategies towards different levels of data heterogeneity among the participating clients. We perform sampling using draws from a Dirichlet distribution controlled by the parameter $\alpha$ where a smaller $\alpha$ increases the probability of clients holding training samples from only one class and vice-versa.

## 4.3 Intelligent Client Selection for Heterogeneous Architectures

In the original FedDF[42] algorithm, they select a random subset of clients for participation in each round. In this work, we customize the existing FedLesScan[17] intelligent clustering-based client selection algorithm to integrate it with the FedDF algorithm for straggler mitigation. We modify FedLesScan[17] such that it considers heterogeneous client model architectures while selecting the participation clients for every round. Algorithm 1 represents the pseudo-code for this modified selection scheme and we recommend having a look at algorithm 2 in the paper FedLesScan[17] first since most of the algorithm components are similar to our implementation and we will not dive deep into the individual components in this section. In terms of terminology, we refer

**Algorithm 1:** Intelligent Client Selection for Heterogeneous Architectures

```
Function Select_Clients(clients, round, maxRounds, nClientsPerRound):
    init map Rookies: maps model prototypes to associated rookie clients
    init map Participants: maps model prototypes to associated participant clients
    init map Stragglers: maps model prototypes to associated straggler clients
    totalRookies = # clients not called before
    totalParticipants = # clients available for clustering
    totalStragglers = # clients where Rounds.last + cooldown > round
    startClusteringRound = -1
    if totalRookies $\geq$ nClientsPerRound then
        return RoundRobinSelection(Rookies, nClientsPerRound)
    end
    nClientsFromClustering = min(nClientsPerRound - totalRookies,
      totalParticipants)
    if nClientsFromClustering $\geq$ 0 and startClusteringRound == -1 then
        startClusteringRound = round
    end
    nStragglers = nClientsPerRound - nClientsFromClustering - totalRookies
    roundStragglers = RoundRobinSelection(Stragglers, nStragglers)
    for each model prototype p in P do
        clusteringData = []
        for each client in participants[p] do
            trainingEma = getEma(client.trainingTimes)
            missedRoundRatios = divide(client.missedRounds, round)
            missedRoundEma = getEma(missedRoundRatios)
            clusteringData.append((trainingEma, missedRoundEma))
        end
        labels = DBScanClustering(clusteringData)
        sortedClusters = sortClusters(Participants[p], labels, round)
        clusteringResults[p] = Sample(sortedClusters, round, maxRounds,
          startClusteringRound, len(participants[p]))
    end
    roundClustering = RoundRobinSelection(clusteringResults,
      nClientsFromClustering)
    roundRookies = RoundRobinSelection(Rookies, "All")
return [roundRookies + roundClustering + roundStragglers]
```

to each unique model architecture as a model prototype. Therefore the model architecture for each client belongs to a particular model prototype. First, we initialize three mappings that map respective rookies, clustering participants, and straggler clients to their associated model prototypes/architectures since we perform this client separation separately for each model prototype (lines 2-4). As per the original FedLesScan paper, this is how they are defined:

1. *Rookies (first-tier)*: Clients that have never been called before to participate in the training
2. *Participants (second-tier)*: Clients that can participate in the clustering for the current round
3. *Stragglers (third-tier)*: Clients that have missed one or more successive rounds $cooldown \geq 0$

If the current remaining rookie clients are more than or equal to the number of clients to be selected, we perform a standard Round Robin [21] based selection from the pool of rookie clients i.e. selecting one rookie client at a time from each model prototype group in a circular fashion until the required number of clients have been selected(lines 9-11). Then, the number of clients available for clustering is selected by subtracting the total number of rookies from the clients required to be selected for the current round (line 12). We also keep track of the first clustering round in order in order to decide which cluster to train (lines 13-15). If the clients from rookies and participants are not enough for the training round, we randomly sample the remaining clients from stragglers (lines 16-17). Now we perform clustering for participating clients associated with each model prototype separately. In the clustering process, first, we calculate two attributes *trainingEma* (line 19) and *missedRoundEma* (line 21) for each participating client which is defined as follows:

1. *trainingEma*: It is the exponential moving average of the training time for previous rounds.
2. *missedRoundEma*: It is a penalty factor based on the previous missed rounds. It is calculated by dividing the numbers in the missed rounds list by the current round number to obtain a list of ratios and then we get an exponential moving average of this list.

$$totalEma = trainingEma + missedRoundEma * maxTrainingTime \tag{4.1}$$

Once this data has been collected for all participating clients of the current model prototype group, we use the DBSCAN [18] algorithm for partitioning this data into separate clusters (line 24). After this, obtained clusters are increasingly sorted based on the average *totalEma* (equation 4.1) of the cluster member clients (line 25). Then, we start sampling clients from these sorted clusters by first choosing clients belonging to the faster clusters and slowly moving towards slower clusters until we obtain the required number of clients (line 26). After the complete clustering process for all model prototype groups, we obtain the *clusteringResults* mapping that maps the sampled participant clients to their model prototype groups. In order to get a similar representation for clients from all model prototypes, we then sample the final participating clients through Round Robin-based selection from *clusteringResults* (line 30) and perform a similar selection in order to get all the rookie clients(line 31). Finally, the algorithm returns a list of clients comprising a combination of selected rookies, participants, and stragglers.

To summarize, the original FedLesScan client selection algorithm performs the clustering process across the complete client pool without taking the model architectures of each client into account during this process. However, since we have enabled the clients to have heterogeneous model architectures now, it becomes important to perform this clustering process at a model architecture level because the clustering metrics such as training times vary across different architectures based on their complexity. Therefore, in this extended version of FedLesScan, we first perform clustering across clients belonging to the same model prototype groups and finally perform a round-robin selection of the sorted clients from each model prototype group until the required number of clients is selected.

## 4.4 FedMD Integration

FedMD algorithm[38] uses transfer learning and knowledge distillation to enable collaborative federated learning across clients with different model architectures and we recommend having a look at 3.2.1 to understand the algorithm. In this section, we dive into the optimized implementation-specific details of this algorithm for serverless federated learning systems.

### 4.4.1 Training Workflow

In figure 4.1, we demonstrate the complete training workflow for FedMD in the serverless paradigm. We have omitted some technical details like client invocation authorization and authentication along with other minor interactions with the database for simplicity. Firstly, the FL admin configures the client models, datasets (public and private), and the required client and FedMD-specific hyperparameters before starting

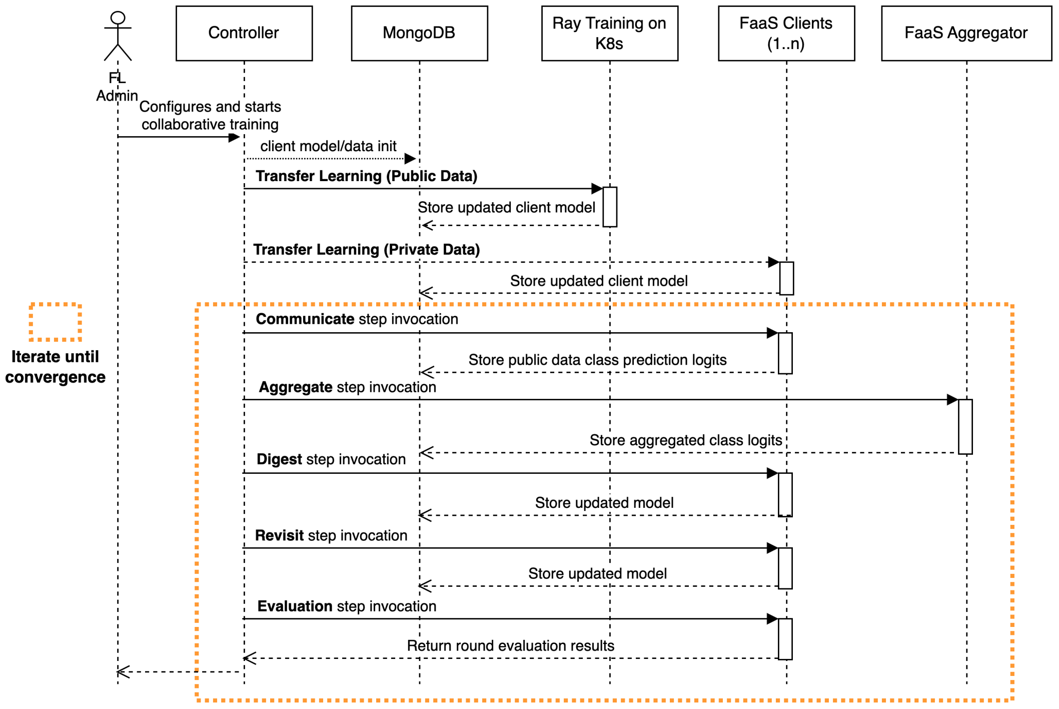

Controller
MongoDB
Ray Training on K8s
FaaS Clients (1..n)
FaaS Aggregator
FL Admin
Configures and starts collaborative training
client model/data init
Transfer Learning (Public Data)
Store updated client model
Transfer Learning (Private Data)
Store updated client model
Iterate until convergence
Communicate step invocation
Store public data class prediction logits
Aggregate step invocation
Store aggregated class logits
Digest step invocation
Store updated model
Revisit step invocation
Store updated model
Evaluation step invocation
Return round evaluation results

Figure 4.1: Serverless training flow for FedMD

the training process. After the training process starts, the controller first builds and initializes the heterogenous client models according to the provided configuration as explained in section 4.1 and also creates the required data loaders for each client. In FedMD, each client has access to various datasets as follows:

- **Private training dataset:** This is the learning task-specific dataset and each client may have a different number of samples as well as data heterogeneity with partial access which can be controlled by the $\alpha$ configuration parameter as mentioned in section 4.2.
- **Private testing dataset**: The whole testing dataset is available to each client since we would like to test every client on all test samples to see the effect of collaborative training.
- **Public training dataset:** This is a labeled public dataset fully accessible by every client and is used for knowledge distillation among the clients for collaborative learning. It should be chosen carefully depending on the learning task.

Now we perform a one-time initial transfer learning process for all the individual client models before the start of the collaboration phase where each client model is trained until convergence on the public training data and then on its private training data. To optimize this process, we divided this into two successive steps as follows:

- **Transfer Learning (Public Data):** Since the public dataset is available openly to all clients, we perform this training on an autoscaling KubeRay[49] operator deployed on a Kubernetes cluster as explained in section 4.6.1. We perform this initial public training in parallel depending on the cluster compute capacity for all the client models until convergence. To achieve this, the controller communicates with the deployed Ray cluster and schedules the training jobs accordingly.
- **Transfer Learning (Private Data):**: Now the controller performs client invocations where each client trains its model further on their private data before starting the collaboration rounds.

After the initial transfer learning phase, we now move on to the collaborative training rounds where the clients start distilling knowledge to each other to achieve higher overall test accuracies. Each collaborative training round comprises several steps which are separated into individual FaaS client invocations that occur until the required individual client model performances are reached:

- **Communicate:** All clients run a prediction on the public dataset and store the prediction logits (pre-softmax) in the database. In this phase, we use a random subset of the total public data samples for each round to speed up the process without performance loss.
- **Aggregate:** Controller invokes the aggregator function which computes the mean of these prediction logits from all the clients. We can also try more sophisticated aggregation methods such as weighted averages based on the client model architectures.
- **Digest:** In this step, each client loads the averaged logits and trains their model to approach the averaged prediction logits from the previous step. In other words, they try to align their logits with the globally averaged logits and therefore distill knowledge towards each other.
- **Revisit:** Finally, each client fine-tunes their model on private data for a few epochs to retain their model personalization.
- **Evaluation:** After the end of every collaborative training round, the controller performs an evaluation invocation to get the performance of each client model on the global test data samples.

Importantly, we observe that there are no model weight or parameter exchanges in the above learning process therefore it can be performed across heterogeneous model architectures. Another interesting aspect to note is that the complete training process is synchronous in nature since all the participating clients need to finish a particular step invocation before moving on to the next step. Therefore, it is hugely beneficial to execute this algorithm in a serverless environment in terms of cost and total execution times since clients can scale down to zero when they are idle and awaiting other clients to finish a particular invocation step. Once the learning process is complete, we observe significant gains in client test accuracy in contrast to their performance when trained only on their private data without collaboration.

## 4.5 FedDF Integration

In FedDF[42], we aim to train central classifiers through predictions of client models on unlabelled data. It is an on-server distillation method for model fusion on the aggregator side and does not require any changes in the client training process i.e. it uses unlabelled data to aggregate knowledge from all the client models to train student server models and this ensemble distillation process allows heterogeneous client models and data. Similarly in our serverless system, there are no significant changes and additional inference burden on the FaaS clients however there are major modifications in the aggregator function to enable this ensemble knowledge distillation. Unlike FedMD, this method relies on unlabeled data for knowledge transfer that can be from any other domain and even works well with synthetic data from pre-trained generators like GANs.

### 4.5.1 Ensemble Distillation for Model Fusion

The key feature of this knowledge distillation process lies in the ensemble distillation step. Originally, ensemble learning methods like bagging and boosting involved combining predictions from multiple weakly supervised models however that it is difficult in a federated learning setting since model parameters are distributed among the clients and it is not possible to keep such a large number of models on the server and combine predictions from them during inference. FedDF makes the ensemble learning process possible in a federated learning setting with the help of knowledge distillation and therefore the term **ensemble distillation** which helps in distilling knowledge from an ensemble of multiple client models with heterogeneous architectures towards a master server model.

Once all the clients have completed their local training for the current round, we start with the ensemble distillation process in our FaaS-based aggregator function. We

| Model ID | # Layer 1 Filters | # Layer 2 Filters |
|---|---|---|
| 1 | 256 | 512 |
| 2 | 256 | 512 |
| 3 | 512 | 512 |
| 4 | 512 | 512 |
| 5 | 512 | 1024 |

Table 4.1: Example of client model configuration for convolutional networks

already know that each participating client may have a different model architecture therefore each client can be part of a specific model prototype. For instance, please refer to table 4.1, we observe that there are 3 model prototypes comprising model ID - (1,2), (3,4), and (5). We execute this ensemble distillation process in parallel for each model prototype i.e. in our example, the aggregator function is invoked three times in parallel once for each model prototype.

The distillation process for each model prototype comprises the following steps:

1. We initialize a client set with all the clients belonging to the particular model prototype.
2. Then we create a student server model with the same architecture as the clients belonging to the current model prototype/architecture group and initialize it using FedAvg[53] of all the client model parameters selected in the previous step. The student server model is just a temporary model created and initialized in the aggregator memory every time it is invoked.
3. Finally, we start with the iterative ensemble distillation process where we sample mini-batches of unlabeled data and then use this data to perform training updates to the student server model as mentioned in equation 4.2 to distill knowledge from the ensemble of **all** teacher client models to the student server model until the server model validation performance plateaus.

$$\mathbf{x}_{t,j} := \mathbf{x}_{t,j-1} - \eta \frac{\partial \, \mathrm{KL}\left(\sigma\left(\frac{1}{|\mathcal{S}_t|}\sum_{k\in\mathcal{S}_t} f\left(\hat{\mathbf{x}}_t^k, \mathbf{d}\right)\right), \sigma\left(f\left(\mathbf{x}_{t,j-1}, \mathbf{d}\right)\right)\right)}{\partial \mathbf{x}_{t,j-1}} \tag{4.2}$$

where:

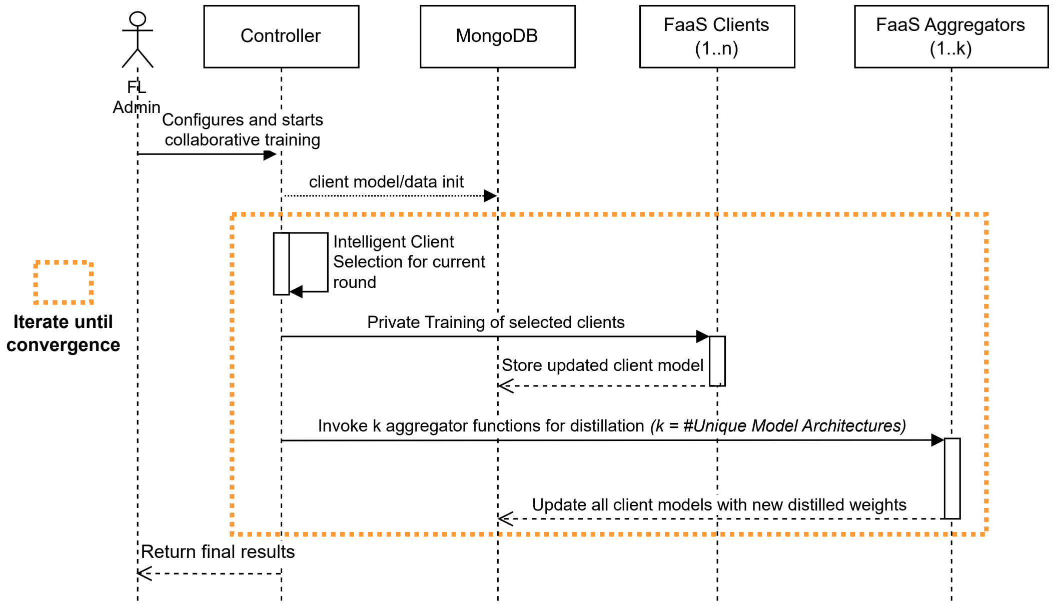

Controller
MongoDB
FaaS Clients (1..n)
FaaS Aggregators (1..k)
FL Admin
Configures and starts collaborative training
client model/data init
Intelligent Client Selection for current round
Iterate until convergence
Private Training of selected clients
Store updated client model
Invoke k aggregator functions for distillation (k = #Unique Model Architectures)
Update all client models with new distilled weights
Return final results

Figure 4.2: Serverless training flow for FedDF

| | |
|---|---|
| $\sigma$ | = Softmax function |
| $\mathcal{S}_t$ | = Subset of selected clients for the training round |
| $f\left(\hat{\mathbf{x}}_t^k, \mathbf{d}\right)$ | = Logit outputs of all clients on mini-batch $\mathbf{d}$ |

Also, KL stands for Kullback-Liebler divergence[32] in equation 4.2. After each round of the ensemble distillation process, we obtain distilled server models for every architecture/prototype group that contains knowledge from all the participating clients. These server models are then distributed back to the clients based on their respective model prototypes for the next training round.

### 4.5.2 Training Workflow

In figure 4.2, we demonstrate the serverless training workflow which is relatively simple given that FedDF is a server-side distillation algorithm. However, since our serverless implementation does not have a central aggregation server, we make use of FaaS aggregation functions for this server-side distillation process which is more efficient as explained in section 4.6.2. Similar to FedMD, first the FL admin configures the client models, public and private datasets, as well as the required client FedDF-specific hyperparameters mentioned in table 5.8 before starting the training process. Now we start iterating through the training rounds in which firstly, the controller selects a fraction of clients from the client pool using our clustering-based intelligent client selection algorithm explained in section 4.3. Afterward, these selected clients are then invoked for their local training process on their private datasets and then these clients store their updated models back into the MongoDB parameter server. Once all the invoked clients have completed their individual training process, the central controller invokes the aggregator functions for the ensemble distillation process explained in section 4.5.1. These aggregator functions are invoked in parallel for each unique model architecture and once finished, each aggregator function updates the weights of clients associated with their particular model architecture. This private client training and the ensemble distillation process iterate until we reach the desired accuracies for each model architecture. Similar to FedMD, all the invocation steps are synchronous and we require results from all the invoked functions until we can proceed to the next round which provides a solid reason to execute this algorithm in a serverless setting resulting in cost benefits.

## 4.6 Overall System Design Improvements

In the previous sections, initially, we discussed the implementation details of the framework enhancements such as the central controller changes, tunable heterogeneous data distribution among the clients, and an extended version of the intelligent client selection algorithm to incorporate heterogeneous client architectures. Then we focused on the implementation of FedMD and FedDF in our serverless federated learning framework. In this section, we highlight the system architecture-level enhancements and modifications that were implemented on top of the existing FedLesScan[17] architecture in order to execute these algorithms in an efficient and optimized manner in a serverless environment. Figure 4.3 provides an overview of the updated system architecture and the existing core components where the modified components are highlighted in green. Overall, there are two major changes that we discuss in the upcoming sections 4.6.1 and 4.6.2. Also, we have added a *Client Parameter Server* MongoDB database that stores the model weights at a client level because we cannot have a single global parameter server due to different client model architectures.

### 4.6.1 Optimized FedMD Pre-Training using Ray

The first major enhancement is the addition of an auto-scaling serverless Ray[49] cluster which runs on top of a Kubernetes deployment. As we explained in section 4.4, the FedMD algorithm requires an initial pre-training of all the client models on the public dataset until convergence before we start with the communication rounds. However, this training cannot take place inside the FaaS clients since the execution time and

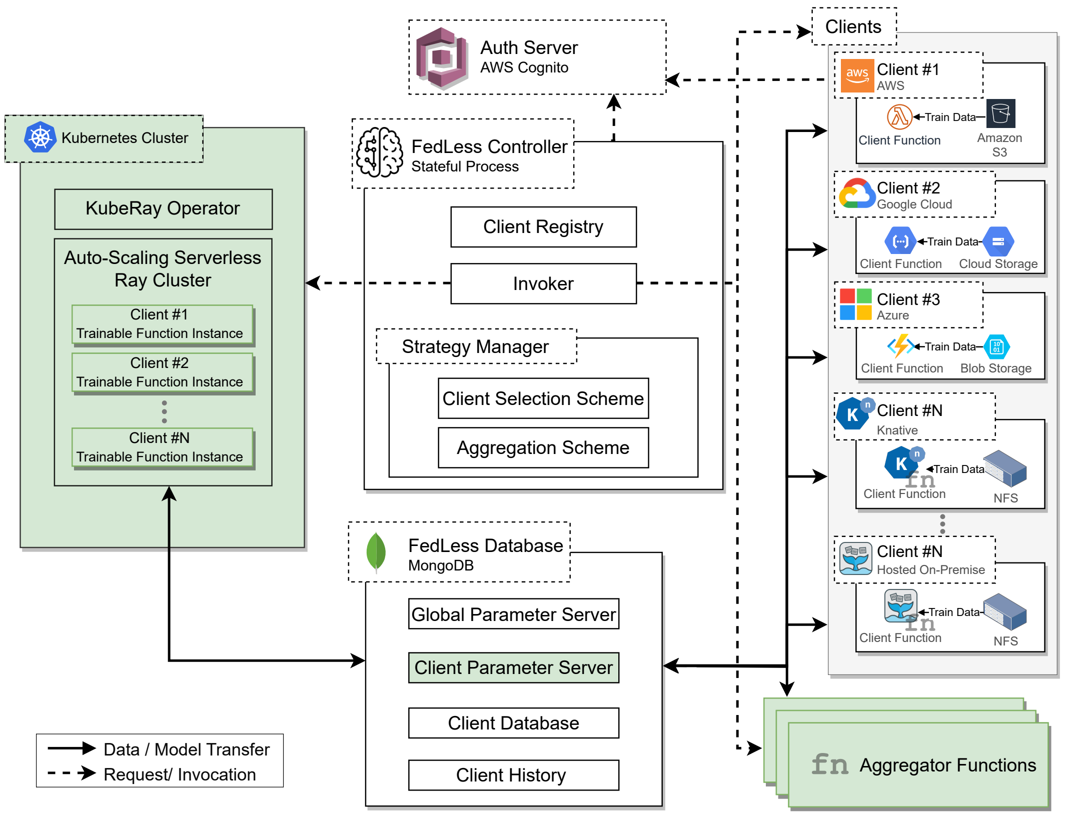


Figure 4.3: Modified architecture of FedLesScan [17]. Enhancements and modifications are highlighted in green.

computation required for this pre-training are relatively higher than the FaaS function limits. The central controller can directly invoke client function instances for model training in an autoscaling Ray cluster. We use the Ray Tune[41] feature for this which creates a *Ray Actor* for each client trainable function instance which are independent Python processes that execute in a distributed parallel manner across the cluster. The deployed Ray cluster automatically creates replica worker pods for function execution based on the number of submitted jobs and then downscales the resources as the jobs finish without any execution time limitations. This helps us to achieve significant speed up for the pre-training process on the public data in FedMD since we can train all the client models in parallel as well as optimize resource usage due to the serverless nature of Ray on Kubernetes. For the FedMD MNIST and CIFAR image classification experiments, we use the *EMNIST Letters* and *CIFAR-10* public datasets as explained in section 5.1.2. We allocate 1vCPU and about 2GB RAM to each parallel training Ray function instance in the cluster and achieve a speed-up of **3.2x** and **3.7x** for MNIST and CIFAR respectively when compared to the total pre-training time for each client model in a sequential fashion.

### 4.6.2 FedDF Parallel Aggregation using FaaS

The framework has now been enhanced to use multiple aggregator functions instead of only one earlier. This was done specifically in order to optimize the knowledge distillation process for the FedDF serverless implementation. The ensemble distillation process explained in section 4.5.1 can take place in parallel for each model prototype-/architecture group and therefore in order to benefit from this property of the algorithm in the serverless setting, we trigger individual aggregator functions for each model prototype in parallel as soon as the knowledge distillation step starts. In the original FedDF implementation, this aggregation process takes place sequentially for each model prototype. However, due to the parallelism in our serverless FaaS aggregators, we achieve speed-ups equivalent to the number of unique model prototypes since that decides the degree of parallelism. For our experiments, we set up 6, 5, and 3 OpenFaaS aggregator functions for MNIST, CIFAR, and Shakespeare respectively based on our choice of unique model architectures as shown in tables 5.3, 5.4 and 5.5 with each aggregator function having limits of 6vCPUs and 16GB RAM. We achieved a decrease in the overall execution time of about 83.33% for MNIST, 80% for CIFAR, and 66.67% for Shakespeare compared to sequential execution i.e. when we have a single aggregator function performing knowledge distillation for each unique model architecture sequentially.

# 5 Experiments and Evaluation

In this section, we first provide an insight into our experiment setup that includes evaluation procedure and metrics, various datasets used along with their distribution among clients, and the heterogenous client model architectures used depending on the tasks. Then we demonstrate the system's performance and accuracy on various CV and NLP tasks. Finally, we conduct the time and FaaS-based cost analysis of our proposed system. .

## 5.1 Experiment Setup

### 5.1.1 Overall Evaluation Procedure

In this section, we give an overview of the evaluation methodologies and metrics that we have used in order to evaluate our system. For the evaluation, we focus on the following three major aspects for our serverless implementation of FedMD[38] and FedDF[42] algorithms:

- Performance, test accuracy, and convergence rates of heterogeneous client models for different CV and NLP learning tasks at various levels of data heterogeneity
- Fine-grained time and cost analysis of different system components for specific learning tasks and datasets

For evaluating model performance, we use the conventional top-1 accuracy metric which means that the model output with the highest probability should be the expected answer. We analyze the top-1 test accuracy as well as the model loss for each client model prototype with respect to the increasing number of communication rounds until convergence. We perform a test performance evaluation for every client after each training round. In the case of FedMD, since the algorithm focuses on training a personalized model for each client, we average the accuracy from clients belonging to the same model prototype group. We perform this complete analysis for different levels of data heterogeneity among the clients as mentioned in section 4.2 since this is a common situation in the FL setting.

We perform a fine-grained time analysis for various knowledge distillation steps that take place in both algorithms in order to complete a collaborative training round.

Our client and aggregator functions are deployed and managed in-house using the OpenFaaS serverless framework on a self-managed Kubernetes cluster. For cost-related analysis, we use Google's Cloud Function pricing[15] in order to estimate the costs for each client function invocation and aggregate them for the complete serverless process execution cost.

### 5.1.2 Datasets and Heterogeneous Client Data Distribution

In our experiments, we evaluate the framework on both CV and NLP tasks and each algorithm requires different types of data loaders including private datasets as well as labeled/un-labeled public datasets. We run experiments on image classification and text character prediction benchmarks as demonstrated in table 5.1 and table 5.2.

MNIST dataset[37] is a classic handwritten digit classification dataset comprising 60,000 training images and 10,000 testing images. EMNIST Letters[16] is a dataset comprising images of handwritten letters from the English language with 145,600 characters distributed across 26 balanced classes and this has been used as a public dataset in both our implementations.

For evaluating more complex image tasks, we use CIFAR-10/100[35, 36] image classification datasets that comprise 60000 color images divided uniformly into 10 and 100 mutually exclusive classes for CIFAR-10 and CIFAR100 respectively.

For the language modeling domain, we use the Shakespeare dataset from LEAF [10] which is a federated learning benchmark as the main private training dataset in both our implementations and a self-prepared character prediction dataset from the openly available Nietzsche text corpus.

As mentioned in section 4.2, the value of Dirichlet distribution $\alpha$ controls the data heterogeneity i.e. the degree of non-i.i.d. private data distribution among the participating clients for classification tasks. As the value of $\alpha$ goes towards zero, the data heterogeneity among the clients increases.

#### FedMD

In the FedMD algorithm, we require a private dataset that represents the main federated learning task and a carefully chosen public dataset to which all the clients have complete access since it is used to distill the learned knowledge among the various heterogeneous participating client models. Each client is initially trained on this public dataset until convergence, and then 5000 randomly sampled records are used for knowledge distillation in each collaboration round for computational efficiency.

| Task Domain | Task Type | FedMD | |
|---|---|---|---|
| | | Private | Public |
| CV | Classification | MNIST | EMNIST Letters |
| CV | Classification | CIFAR100 (Subset) | CIFAR10 |
| NLP | Character Prediction | Shakespeare | Nietzsche |

Table 5.1: Summary of evaluation tasks and datasets for serverless FedMD

Figure 5.1 visualizes the client private data distribution of MNIST among 100 clients for different values of $\alpha$ where the size of each data point represents the number of samples for a particular class. A small $\alpha$ value increases the likelihood of a client containing data for only a subset of unique classes whereas a high value ensures uniform distribution of classes across all the clients. Furthermore, we use the EMNIST Letters as the public distillation dataset for the MNIST classification task. In terms of the testing, each client model gets evaluated on the complete test dataset comprising 10000 testing images.

For the CIFAR benchmark, we use a subset of classes from CIFAR-100 {0, 2, 20, 63, 71, 82} for the main learning task, and the client private data distribution is visualized in figure 5.2 at different levels of data heterogeneity. We use the complete CIFAR-10 as the public distillation dataset for this task. As explained earlier in section 3.2.1, the final output layer of the client model architecture in FedMD has a total number of output neurons equal to $(\#PrivateDataClasses + \#PublicDataClasses)$ since the same model needs to be trained on both the public as well as the private dataset for the training process and both these datasets have a mutually exclusive set of classes in most cases. Therefore, we choose the public dataset in such a way that we can minimize the total number of output classes and maximize the amount of available public data for better performance of the algorithm since a greater number of output neurons negatively affects the final accuracy levels. For testing, each client is evaluated on the complete global CIFAR-100 test dataset subsetted for these six classes.

For the language modeling benchmark, we use the Shakespeare dataset from the LEAF federated benchmarking framework for the next character prediction given the previous 80 characters of a sentence. It contains texts from 1129 different users and in the non-i.i.d. sampling scenario, we simply distribute data according to the raw users since data distributions naturally vary across users. For the i.i.d. sampling scenario, every text data point is equally likely to be sampled, and therefore all clients have similar distributions. Individual client-specific test sets are also created during data pre-processing using similar sampling strategies for performing a distributed evaluation of the clients. In terms of the public knowledge transfer dataset, we create

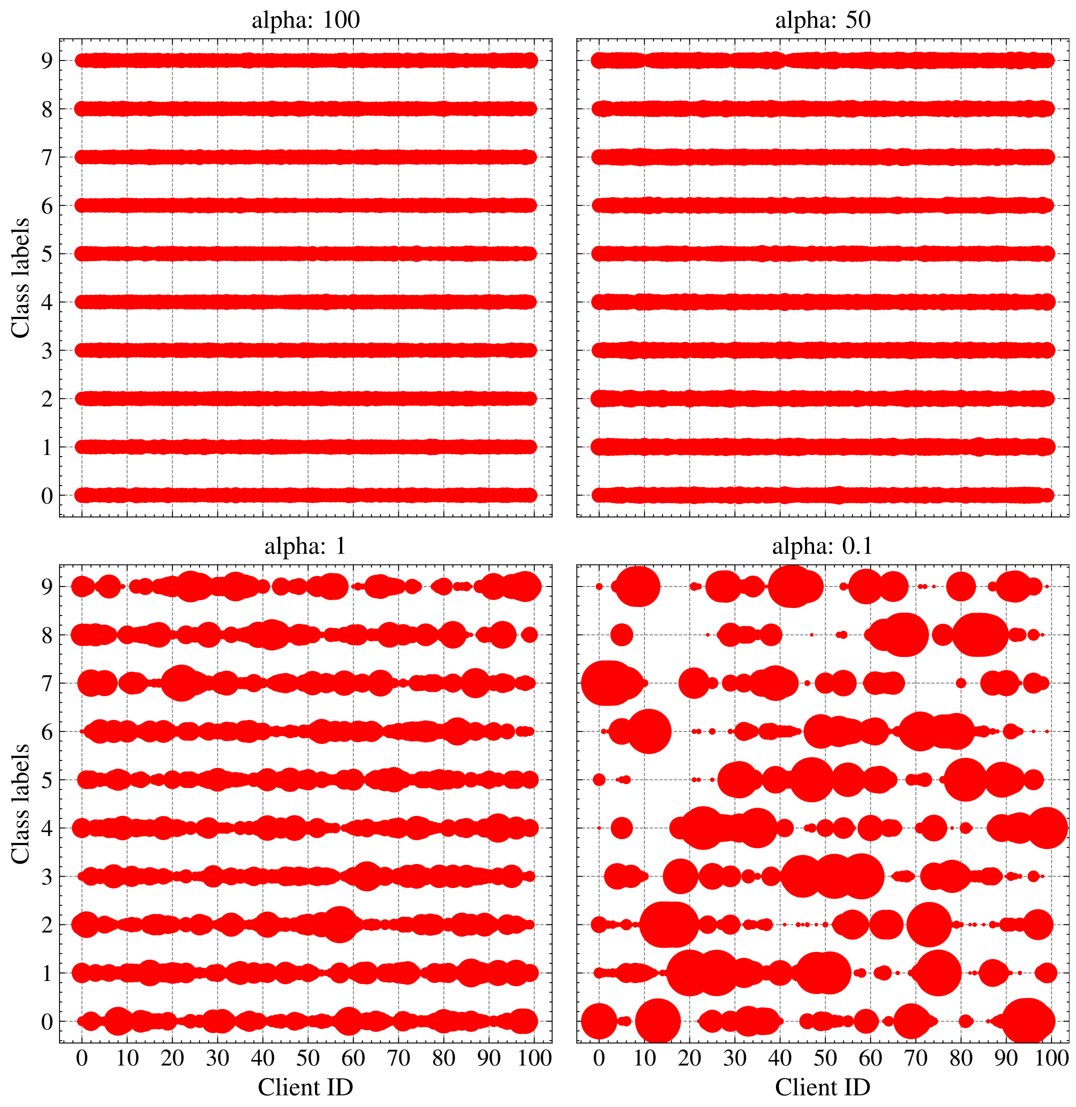


Figure 5.1: FedMD/FedDF Private Client Data Distribution for MNIST Dataset

synthetic character prediction data samples from the openly available Nietzsche text corpus with a sequence length of 80 as the feature and a single character as the label similar to the structure of Shakespeare and a major chunk of code for this processing was borrowed from [55].

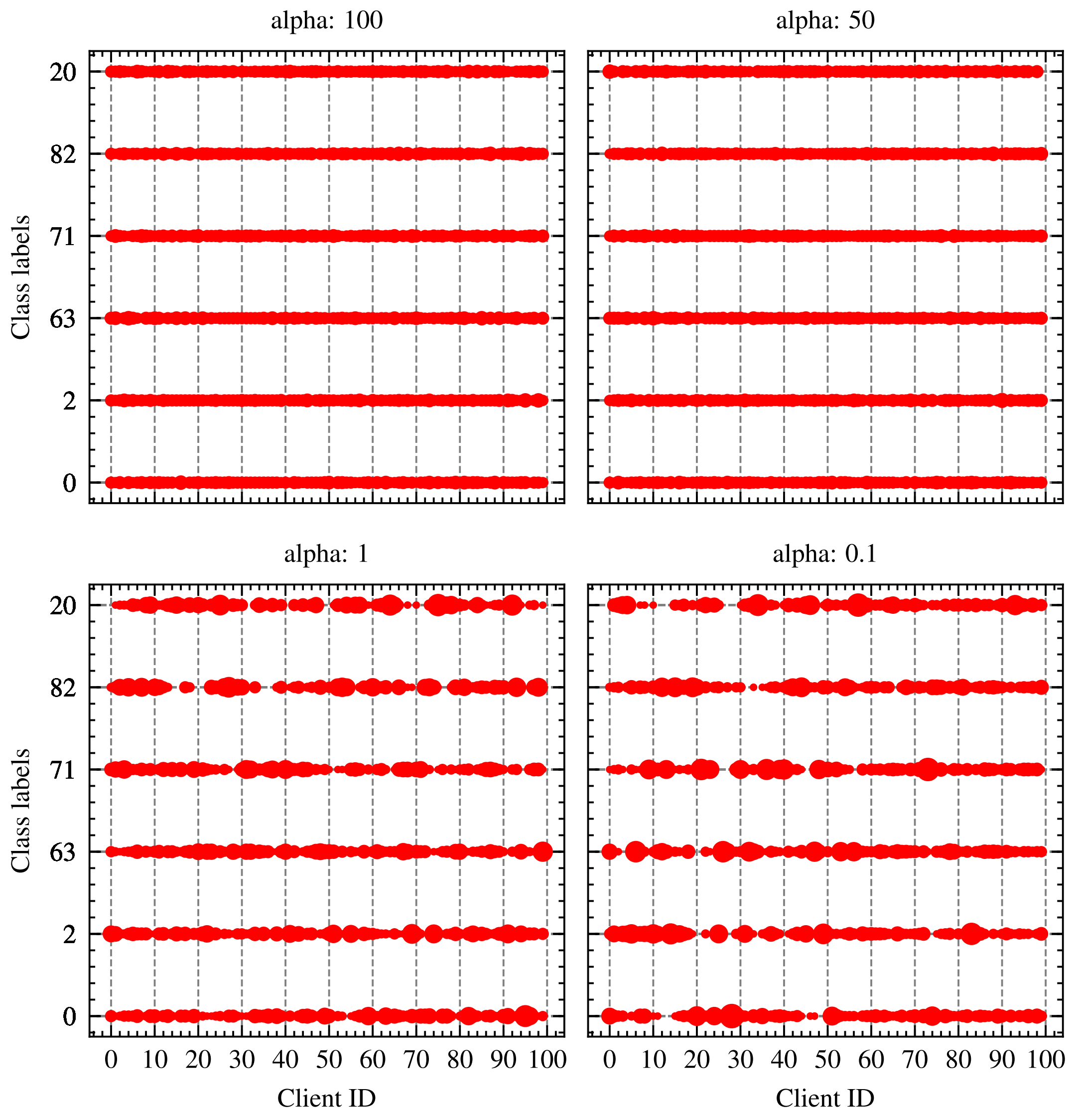


Figure 5.2: FedMD Private Client Data Distribution for CIFAR-100 (Subset) Dataset

### FedDF

Similar to other algorithms, FedDF requires a private client dataset and an unlabelled public dataset which can be chosen with fewer restrictions compared to the FedMD algorithm in terms of both sizes as well as quality.

In the case of MNIST, we use the private and public data distributions similar to the FedMD algorithm in terms of both training and testing where the private client

| Task Domain | Task Type | FedDF | |
|---|---|---|---|
| | | Private | Public |
| CV | Classification | MNIST | EMNIST Letters(Unlabeled) |
| CV | Classification | CIFAR10 | CIFAR100 (Unlabeled) |
| NLP | Next Character Prediction | Shakespeare | Nietzsche |

Table 5.2: Summary of evaluation tasks and datasets for serverless FedDF

distribution is shown in figure 5.1 however, we do not use any labels from the public dataset in this case and just require the image features for the distillation process.

For the CIFAR classification task, we use CIFAR-10 as the private training dataset and the client distribution has been visualized in figure 5.3 for various degrees of data heterogeneity. For the public dataset, we use 20000 randomly sampled unlabelled data points from the CIFAR-100 dataset which are available to all the participating clients for knowledge distillation. Also, we use the complete CIFAR-10 test set comprising 10000 images for evaluating individual client models.

For the language modeling benchmark, we use exactly the same datasets and distribution as FedMD without any labels for the public Nietzsche dataset since we only require an unlabelled public dataset of the same modality for FedDF. Please refer previous FedMD data section for more details on the preparation of this public dataset.

### 5.1.3 Client Model Architectures

Since the main goal of this project is to enable clients with heterogenous model architectures to collaborate and learn in a federated setting, we use several different architectures depending on the task which are divided among 100 participating clients.

For both MNIST as well as CIFAR, we use 2-layer and 3-layer convolutional neural networks which are divided unevenly among the clients in order to simulate real-world scenarios as shown in tables 5.3 and 5.4. We use the same client model distributions for both FedMD and FedDF in order to ensure a fair comparison between the performance of both algorithms in the serverless paradigm. Each convolution filter layer is followed by batch normalization, ReLU activation, and a dropout of 0.2 for regularization and to prevent the client models to overfit on small amounts of private local data. Finally, we have a fully connected layer with output neurons equal to $(\#PrivateDataClasses + \#PublicDataClasses)$ in the case of FedMD and $(\#PrivateDataClasses)$ in case of FedDF followed by softmax activation and a sparse categorical cross-entropy loss.

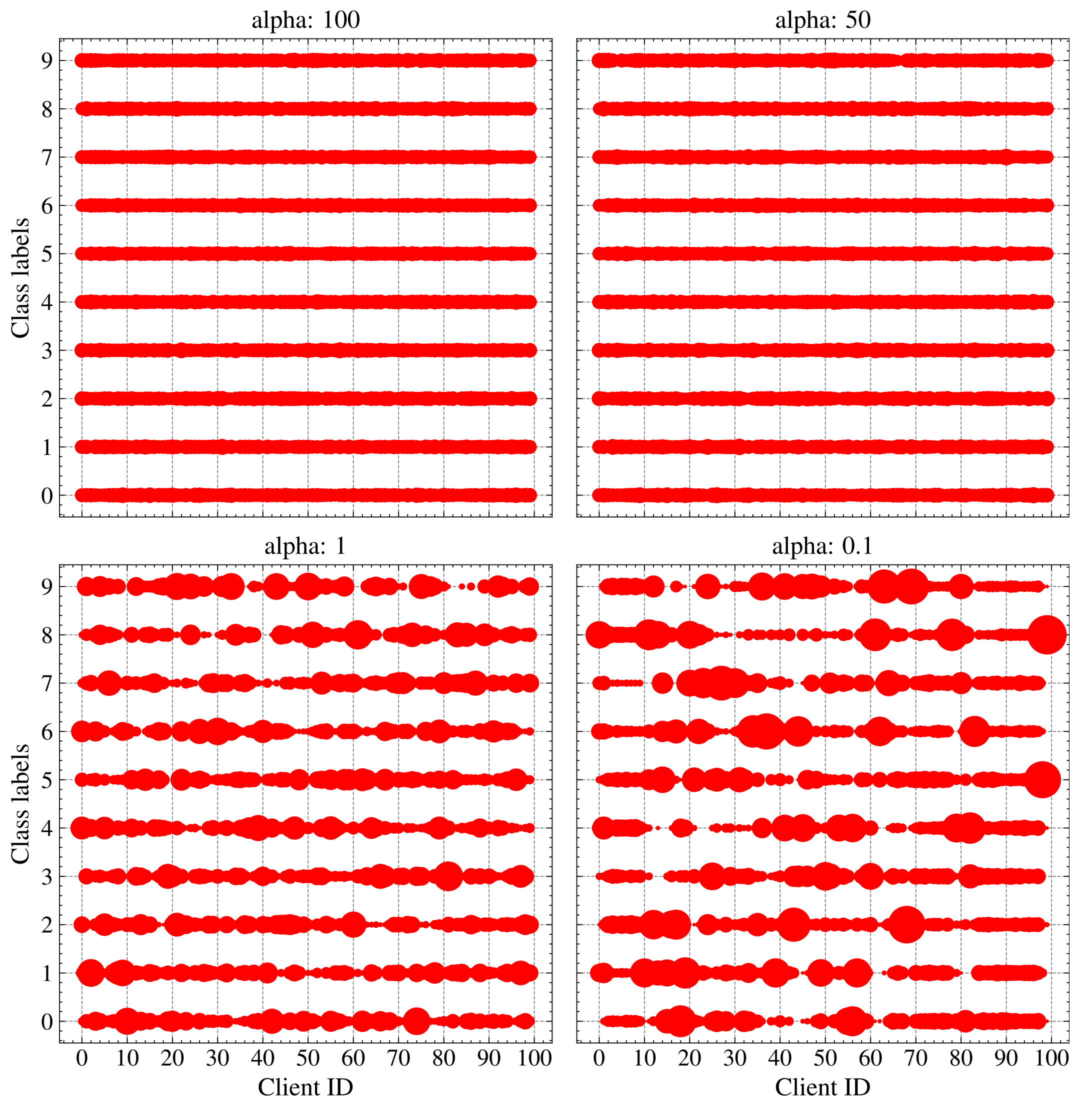


Figure 5.3: FedDF Private Client Data Distribution for CIFAR-10 Dataset

For Shakespeare, we make use of Long Short-Term Memory (LSTM) Recurrent Neural Networks with a single layer and a varying number of units as shown in table 5.5. Every network takes an input sequence length of 80 units followed by an initial embedding size of 8 and then a single LSTM layer. Finally, we have a softmax output

| Model ID | #Clients | #Layer 1 Conv Filters | #Layer 2 Conv Filters | #Layer 3 Conv Filters | #Trainable Parameters |
|---|---|---|---|---|---|
| 0 | 10 | 128 | 256 | - | 729,856 |
| 1 | 30 | 128 | 512 | - | 1,458,176 |
| 2 | 20 | 64 | 128 | 128 | 193,280 |
| 3 | 20 | 64 | 128 | 256 | 352,640 |
| 4 | 10 | 128 | 128 | 128 | 226,816 |
| 5 | 10 | 128 | 128 | 256 | 386,176 |

Table 5.3: Client Convolutional Neural Network Models for MNIST

| Model ID | #Clients | #Layer 1 Conv Filters | #Layer 2 Conv Filters | #Layer 3 Conv Filters | #Trainable Parameters |
|---|---|---|---|---|---|
| 0 | 10 | 128 | 256 | - | 729,856 |
| 1 | 10 | 64 | 128 | 192 | 274,112 |
| 2 | 30 | 64 | 64 | 128 | 104,128 |
| 3 | 30 | 64 | 128 | 256 | 352,640 |
| 4 | 20 | 128 | 128 | 128 | 226,816 |

Table 5.4: Client Convolutional Neural Network Models for CIFAR

layer comprising a vocabulary size of 82 units with a sparse categorical cross-entropy loss. We did not use deeper LSTM networks since these networks are resource hungry and FedMD requires every client to participate in each round.

### 5.1.4 Hyperparameters

Both algorithms require several important hyperparameters for various steps involved in each collaboration round in order to achieve the required performance.

| Model ID | #Clients | #Units | Embedding Dim | #Trainable Parameters |
|---|---|---|---|---|
| 0 | 60 | 128 | 8 | 81,378 |
| 1 | 10 | 64 | 8 | 24,674 |
| 2 | 30 | 256 | 8 | 293,090 |

Table 5.5: 1-Layer LSTM Network Models for Shakespeare

| Dataset | Step | Epochs | Batch Size |
|---|---|---|---|
| MNIST | Transfer Learning (Public) | 20 | 256 |
| | Transfer Learning (Private) | 5 | 128 |
| | Digest | 1 | 128 |
| | Revisit | 3 | 5 |
| CIFAR | Transfer Learning (Public) | 20 | 256 |
| | Transfer Learning (Private) | 6 | 64 |
| | Digest | 1 | 64 |
| | Revisit | 2 | 5 |
| Shakespeare | Transfer Learning (Public) | 40 | 256 |
| | Transfer Learning (Private) | 10 | 256 |
| | Digest | 1 | 128 |
| | Revisit | 2 | 5 |

Table 5.6: Hyper Parameters for FedMD

**FedMD**

Table 5.6 shows the various hyperparameters that were used depending on the task for FedMD. We use the Adam optimizer in all the steps with a fixed learning rate of 0.001. For the digest step, we randomly sample 5000 data points from the public dataset for each round in order to accommodate the computational resources. We observe a higher number of epochs for the Transfer Learning (Public) step since it takes place on the server side in a Ray training cluster and involves training on the complete public dataset until convergence. We also apply validation accuracy-based early stopping in the public transfer learning step.

**FedDF**

Table 5.7 shows the local client training hyperparameters for the FedDF learning tasks. Since this is a server-side aggregation algorithm, the client training parameters are relatively simple without involving any separate steps. Similar to the previous section, all clients use the Adam optimizer with a fixed learning rate of 0.001 for private training.

The noisy knowledge distillation process for FedDF takes place on the server side which involves transferring knowledge from all the individual client-teacher models

| Dataset | Epochs | Batch Size |
|---|---|---|
| MNIST | 5 | 64 |
| CIFAR | 8 | 64 |
| Shakespeare | 10 | 128 |

Table 5.7: Client Private Training Hyper Parameters for FedDF

| Hyperparameter | Value | Description |
|---|---|---|
| Pseudo Batches | 200 | Total number of public data batches used for noisy knowledge distillation |
| Pseudo Batch Size | 64 | Size of each pseudo batch |
| Eval Frequency | 10 | Frequency for evaluation on validation dataset to perform early stopping |
| Patience | 6 | Number of evaluations with no improvement after which distillation will be stopped |

Table 5.8: FedDF Aggregator Hyperparameters

to the server student models as means of aggregation. This server-side knowledge distillation process takes place in parallel for each unique model architecture by invoking multiple aggregator functions having the same distillation hyperparameters as mentioned in table 5.8. As explained in section 4.5.1, the distillation process involves the transfer of learning knowledge from a set of clients towards a central server model with the help of a public dataset because of different architectures. For each invocation of this aggregator function to perform the distillation process, a set of aggregator-specific parameters for the execution of this process are required since it requires several iterations of small noisy knowledge transfer steps for each invocation. These knowledge transfer steps take place using batches of the public data set until the validation accuracy does not increase for *Patience* number of evaluations. We perform an evaluation using the validation data after every *Eval Frequency* number of batches and table 5.8 shows the values of these hyperparameters that we have used in all our FedDF experiments along with their descriptions. They need to be selected very carefully especially in a serverless environment since these parameters determine the total execution time and computation capacity required by the aggregator function for each invocation. Therefore, we limit the *Pseudo Batches* parameter to 200 since that is the maximum amount of public data batches that can be used for knowledge distillation in a single invocation and it fits easily into the FaaS execution time limit of 20 minutes without any loss in performance.

### 5.1.5 Infrastructure and Experiment Setup

The MongoDB parameter server and the Nginx file server hosting all the required datasets were deployed on individual virtual machines with 10 vCPUs and 45GB RAM. Furthermore, we executed the central controller script on a separate machine with 10 vCPUs and 45GB RAM in order to avoid any resource interference with other components.

We deployed 100 OpenFaaS client functions for all the experiments. Each client had a limit of 2vCPUs and 4GB RAM with a maximum execution timeout of 20 minutes. In terms of aggregation functions, we deployed a single OpenFaaS-based aggregation function with limits of 4vCPUs and 8GB RAM for FedMD since the algorithm does not require many resources on the aggregator side. However, in the case of FedDF, we deployed 6 aggregator functions (one for each unique model architecture) with each function having limits of 6vCPUs and 16GB RAM because most of the processing takes place on the aggregator side. For the Ray training cluster, we set up a Ray head pod with limits 4vCPUs and 20GB RAM along with auto-scaling pods with up to 8 replicas where each replica has limits of 4vCPUs and 20GB RAM.

For image classification datasets, we perform experiments at different levels of data heterogeneity i.e. {100, 50, 1, 0.1} as mentioned in section 5.1.2 in both FedMD as well as FedDF in order to evaluate the robustness of both algorithms. For Shakespeare text character prediction task, we perform two experiments i.e. i.i.d. and non-i.i.d. data distribution for both algorithms. Overall, we perform around 20 training runs overall with each run lasting 20 communication rounds. For time and cost-related analysis, we combine the data generated from all 4 levels of data heterogeneity in order to get better variance estimates since execution times are independent of the data heterogeneity levels among the clients.

## 5.2 Learning Task Performance and Accuracy

In this section, we perform an in-depth analysis of the test performance and convergence for both algorithms on the different machine learning tasks we described in the previous section. For classification tasks, we perform this analysis at different levels of data heterogeneity among the clients which are controlled by the $\alpha$ parameter as explained earlier in section 5.1.2, and each client is evaluated on a global test dataset after the end of each collaboration round in order to understand the performance improvement through client collaboration and knowledge distillation. We use relatively simple model architectures for all the learning tasks because our system is based on FaaS training clients, therefore our goal in this section will be to demonstrate convergence and increased accuracy through knowledge distillation among heterogeneous model clients in contrast to achieving state-of-the-art accuracies on these tasks.

### 5.2.1 FedMD

On the MNIST image classification task, we performed four experiments with varying levels of data heterogeneity for 20 communication rounds. Each line in the plots belonging to figure 5.4 represents the average test accuracy across the clients belonging to a particular model architecture group since we distribute 6 unique model architectures across the clients as shown in table 5.3. We observe that the majority of the knowledge distillation takes place during the first few epochs and then the accuracy curve plateaus without much change in the performance levels. Interestingly, we observe that as we go towards the maximum level of client data heterogeneity, there is a significant negative impact on the top test accuracies that can be achieved which means that this algorithm is robust to data heterogeneity but only upto a certain extent and cannot handle extreme heterogeneity levels. In the standard i.i.d. scenario, we were able to achieve maximum accuracy of 96% for *model 4*. The average accuracy across all models drops by around 46.5% at alpha equal to 0.1 compared to the i.i.d. scenario with alpha equal to 100. One of the major reasons for this decrease in accuracy with increased data heterogeneity is the divergence of the global model gradients due to high variance in the private data distribution among the clients especially for personalized federated learning algorithms like FedMD.

Furthermore, we show the test loss curves for these experiments in Figure 5.5 and observe a clear correlation with the increasing accuracy levels. As expected, we see a high loss during the initial rounds, especially for smaller convolutional network models with fewer learning capacities which stabilizes and plateaus later on after a few distillation rounds.

Similarly, for the next image classification task CIFAR, we distribute 5 unique convolutional model architectures among the participating clients as shown in table 5.4. Figure 5.6 depicts the test accuracy curves over increasing communication rounds for different data heterogeneity levels. We observe a relatively smooth accuracy increment with collaboration rounds in contrast to the MNIST dataset. Here as well, we were able to achieve similar accuracy levels compared to the original IaaS-based FedMD[38] implementation in the serverless paradigm with a small difference of 4-5% due to varying model architectures. We got a maximum test accuracy of 66.4% for *model 2*. However, we observe a drop in test accuracy with increasing data heterogeneity equivalent to a 13% average drop across all model architectures for the case where alpha is equal to 0.1 compared to the standard i.i.d. scenario. Taking a look at the loss

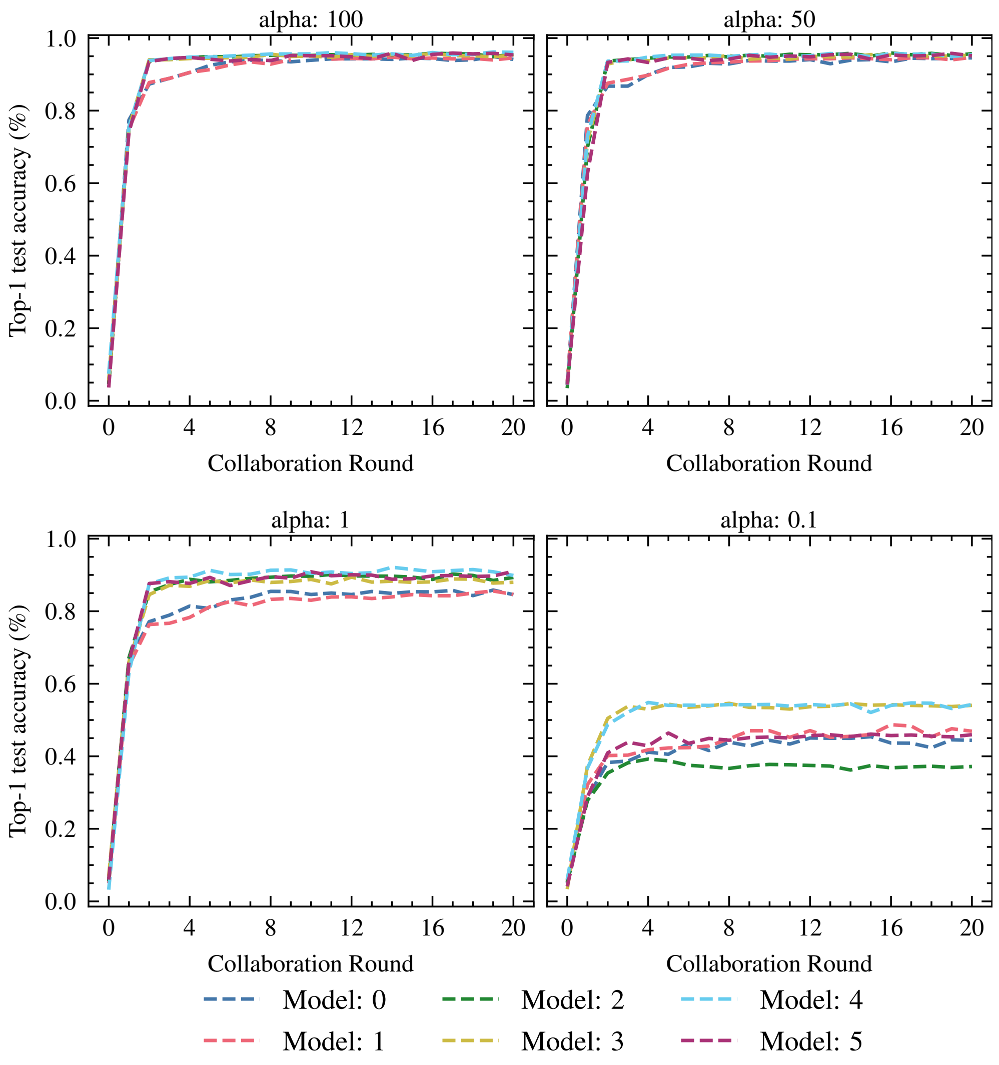


Figure 5.4: Test accuracies for FedMD on MNIST Dataset with different data heterogeneity levels. Refer to table 5.3 for model specifications.

curves in Figure 5.7, it varies with the accuracy as expected along with jitters for the smallest low-capacity model: 0 which eventually reduces as the knowledge distillation process moves further in time.

Finally, figure 5.8 shows the client test performance on the Shakespeare dataset for the next character prediction task averaged across three unique model architectures. In this case, we run experiments for IID and Non-IID scenarios since creating synthetic heterogeneity using Dirichlet distribution is only suitable for classification tasks. The

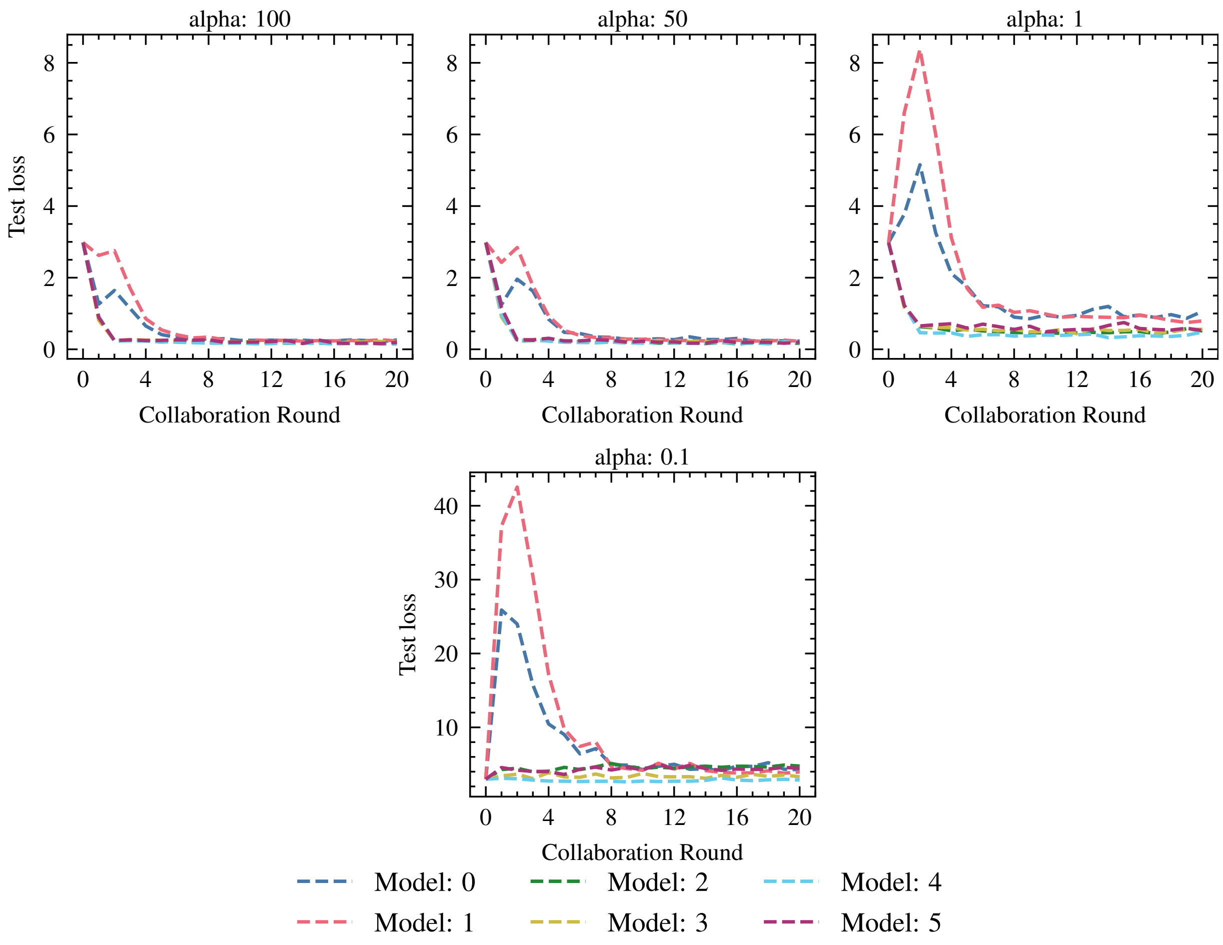


Figure 5.5: Test loss for FedMD on MNIST Dataset with different data heterogeneity levels. Refer to table 5.3 for model specifications.

LEAF[9] FL benchmark already provides preprocessing scripts for distributing data across clients in both IID and non-IID scenarios. We were able to achieve similar performance compared to our predecessor FedLesScan with only 3% lower accuracy even with single-layer heterogeneous LSTM models whereas FedLesScan uses a two-layer stacked LSTM recurrent neural network. *Model 1* achieved the highest test accuracy in both i.i.d. as well as non-i.i.d. cases equivalent to 36% and 37.8% respectively. In terms of data heterogeneity, we were able to converge much earlier in the IID case however, we do not see much difference in the highest achieved test accuracy performance between IID and non-IID scenarios. We believe that this might be due to the nature of the task at hand since we perform the FedMD transfer learning step on a

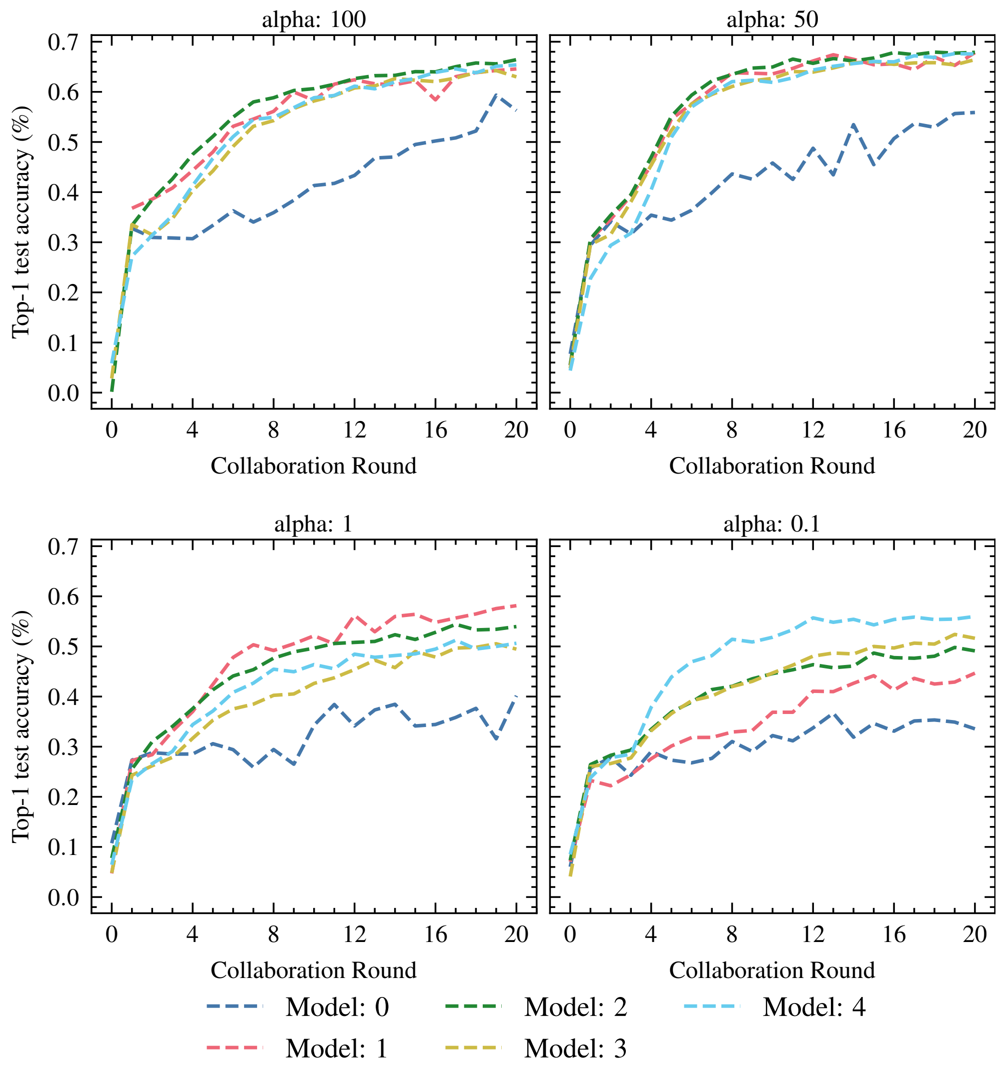


Figure 5.6: Test accuracy for FedMD on CIFAR Dataset with different data heterogeneity levels. Refer to table 5.4 for model specifications.

public dataset(Nietszche) predicting the same set of classes as the private Shakespeare dataset with just a difference in the text corpus between both of them. As a result, we already have strong character prediction models at the start of the collaborations rounds and then it is just a matter of performing collaborative fine-tuning among the clients specific to the Shakespeare dataset. Also interestingly, in terms of loss curves as shown in figure 5.9, we observe that we were able to achieve the minimum loss much

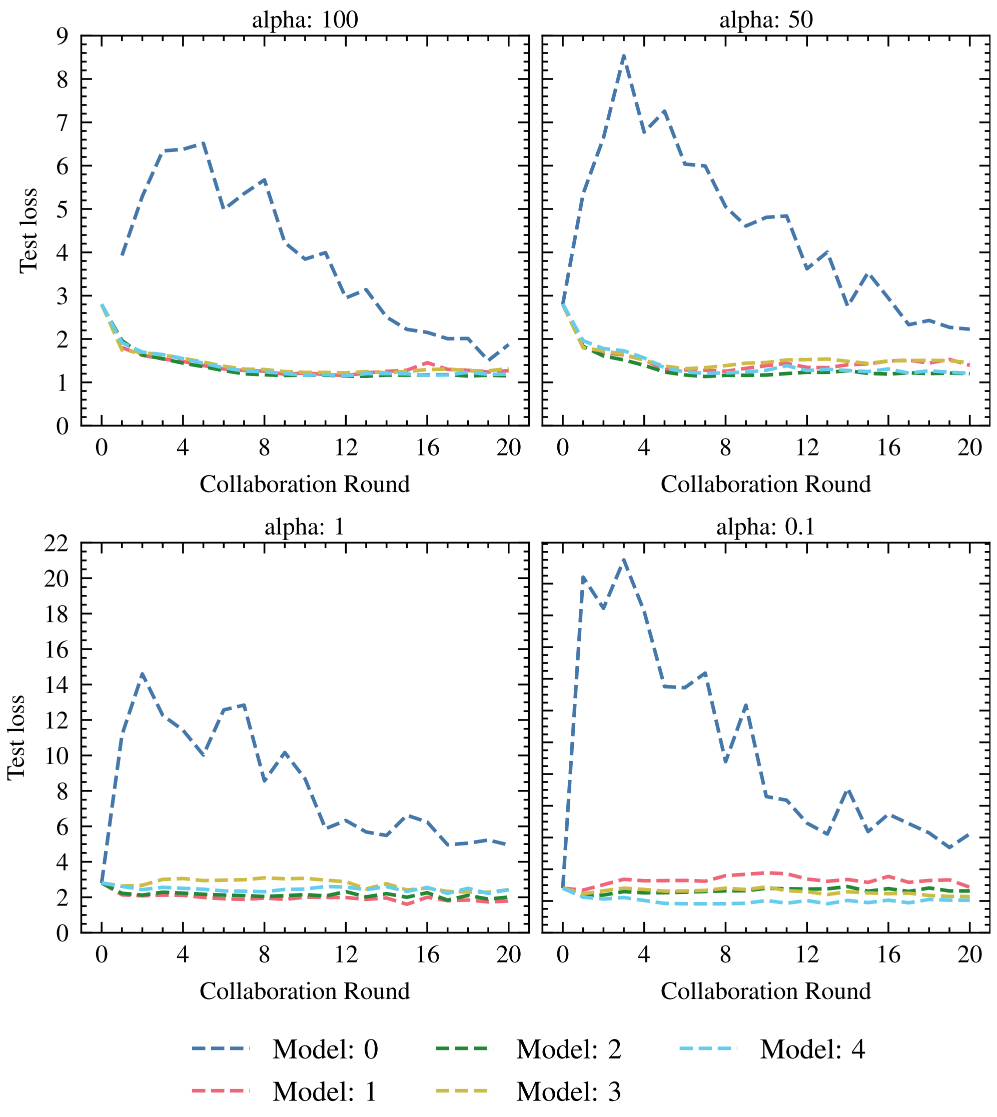


Figure 5.7: Test loss for FedMD on CIFAR Dataset with different data heterogeneity levels. Refer to table 5.4 for model specifications.

before completing all the training rounds as it is also evident from the accuracy curves and the loss starts diverging when we train further leading to minor drops in the test accuracy.

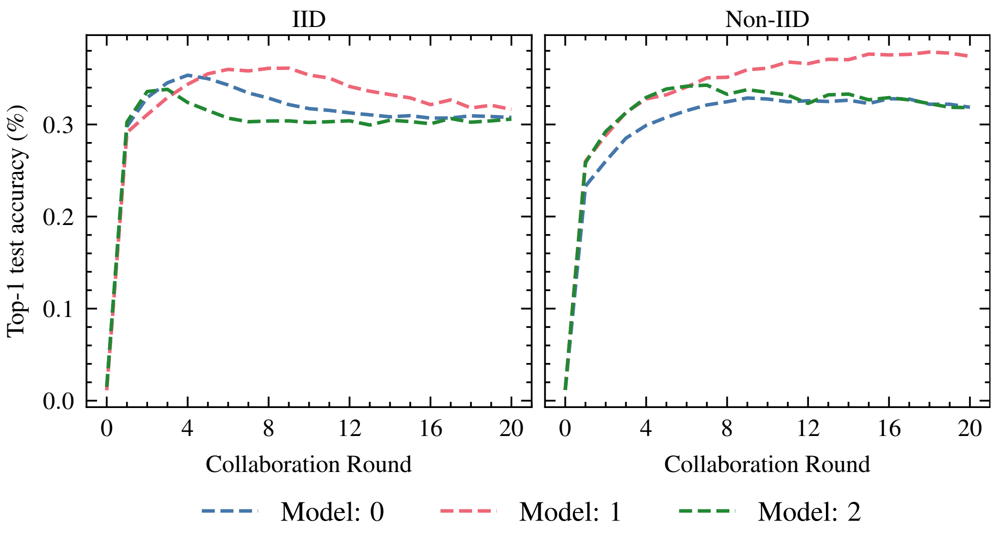


Figure 5.8: Test accuracy for FedMD on Shakespeare Dataset for IID and Non-IID scenarios. Refer to table 5.5 for model specifications.

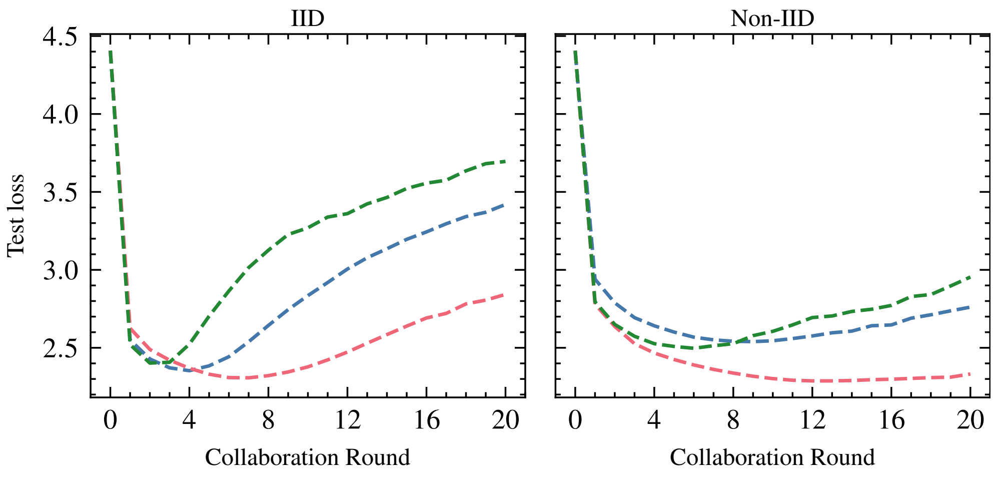


Figure 5.9: Test loss for FedMD on Shakespeare Dataset for IID and Non-IID scenarios. Refer to table 5.5 for model specifications.

### 5.2.2 FedDF

For our serverless FedDF implementation, we perform 4 experiments at different levels of data heterogeneity similar to the FedMD experiments. In the case of classification tasks MNIST and CIFAR, each accuracy line represents the top-1 test accuracy of the global server model for that particular model architecture group. As a further clarification, in the case of FedDF, we have one aggregator function per model architecture that distills knowledge from all the participating clients to the corresponding server model for that model architecture and then reports test accuracy for this model on a global test dataset before distributing the updated model back to the clients. However, in the case of Shakespeare, we do not have access to a central global test dataset but rather individual client-level test datasets therefore we follow a client-level test accuracy averaging approach similar to FedMD after each communication round. Also, since not all aggregators for each model architecture type are invoked during the initial rounds based on our intelligent client selection scheme, we interpolate the test accuracy data for these rounds. We do not show any loss curves for FedDF since it is a noisy knowledge distillation process and loss curves do not provide us with much useful performance-related information.

Investigating the performance on the MNIST task as shown in figure 5.10 different data heterogeneity levels, we observe a maximum accuracy of 97% in standard i.i.d. scenario with alpha 100 and 94.8% in extreme non-i.i.d. scenario with alpha 0.1. We were able to achieve similar test accuracy compared to our predecessor FedLesScan[17](98.5% in 60 rounds) for heterogeneous client architectures in relatively lower communication rounds. We observe that FedDF is much more robust to extreme non-IID data distribution since it retains the test accuracy on all heterogeneity levels in contrast to the FedMD algorithm having 44% less test accuracy averaged across all the architectures for 0.1 alpha level.

For the next image classification task based on the CIFAR dataset, figure 5.11 demonstrates the learning task convergence over collaboration rounds. Given the relatively simple convolutional networks that we use for this particular task, we were able to achieve a maximum test accuracy levels of 57.5% in the ideal i.i.d. scenario and 55% in the extreme non-i.i.d. case. Similar to MNIST, FedDF shows a similar robustness to increasing data heterogeneity on the CIFAR dataset.

Finally, for the Shakespeare dataset, figure 5.12 shows the test performance where we distribute 3 single-layer recurrent neural networks with different numbers of LSTM units among the participating clients. We achieved a maximum test accuracy of 40% for the IID case and 43% for the non-IID dataset which again demonstrates the robustness of FedDF to non-IID data distribution across the clients for the text data modality. Therefore, for this dataset, we were able to achieve 4.2% higher accuracy for the non-i.i.d. case compared to standard FedLesScan[17] results even with smaller single-layer heterogeneous LSTM neural networks and a lesser number of rounds.

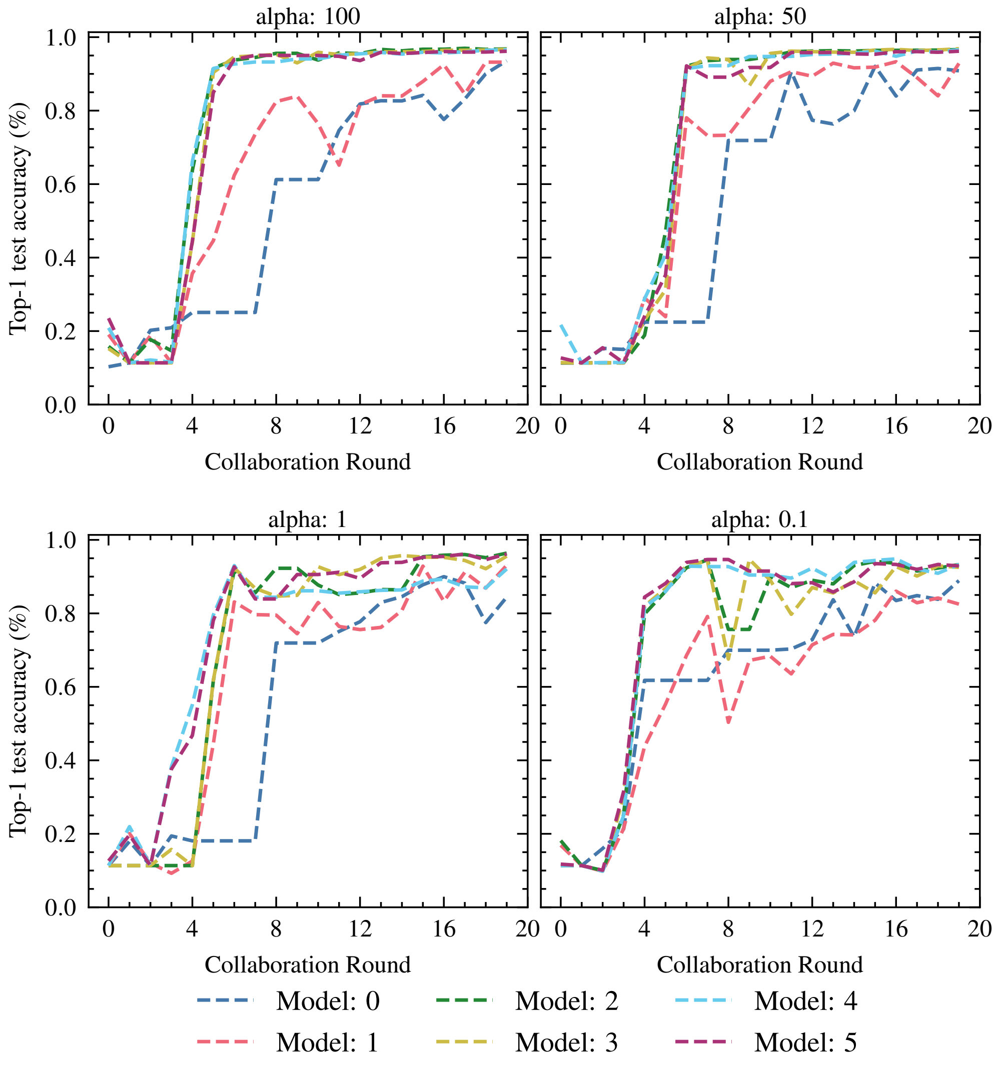


Figure 5.10: Test accuracy for FedDF on MNIST Dataset with different data heterogeneity levels. Refer to table 5.3 for model specifications.

## 5.3 Time and Cost Analysis

We talked about the accuracy and performance of our implementation in the previous section, however, an important factor in analyzing federated learning systems is the time and cost required to accomplish these training tasks, especially in our case since we use serverless FaaS which involves invocation time-based billing for computations. In this section, we perform an in-depth time and cost analysis of all the performed

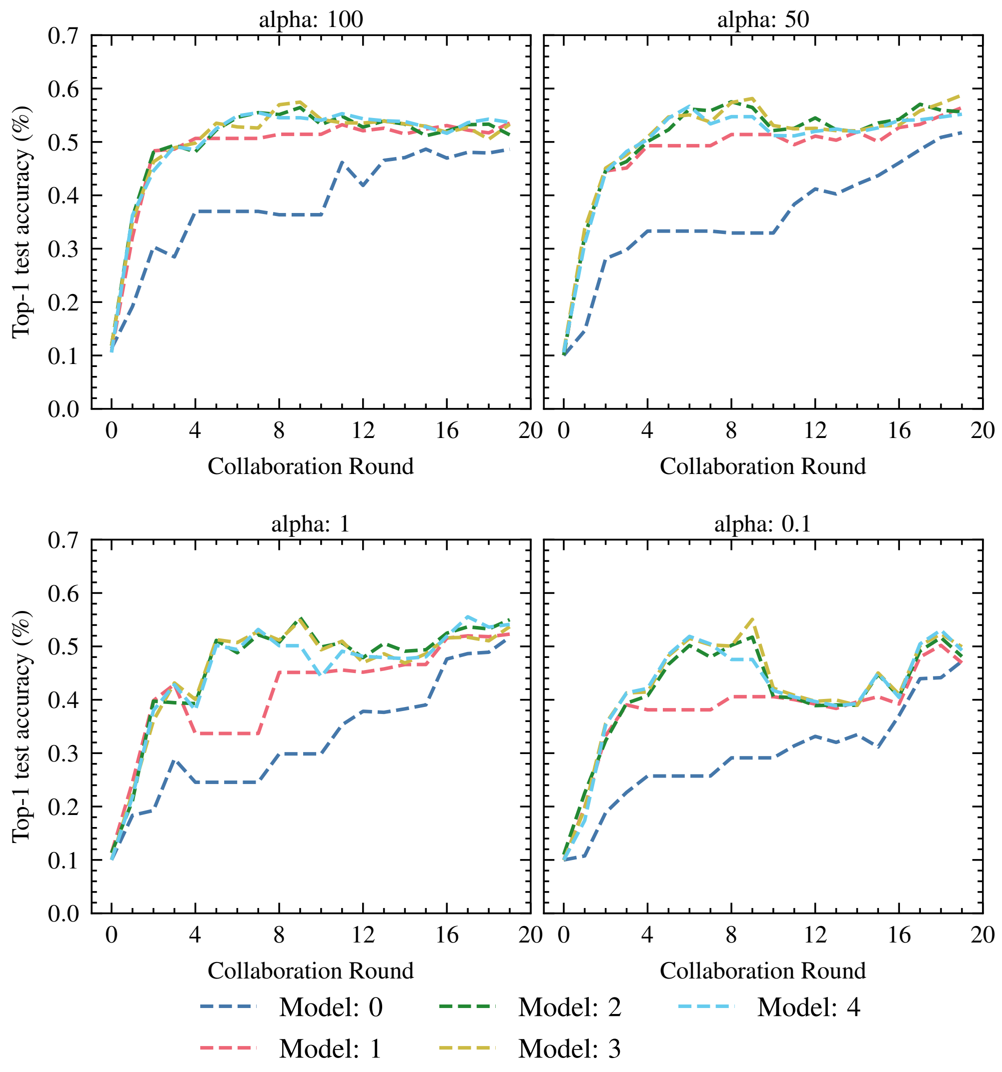


Figure 5.11: Test accuracy for FedDF on CIFAR Dataset with different data heterogeneity levels. Refer to table 5.4 for model specifications.

experiments for both FedMD and FedDF algorithms. We aggregate the timings and costs for the different levels of data heterogeneity since data heterogeneity does not have an impact on the execution times.

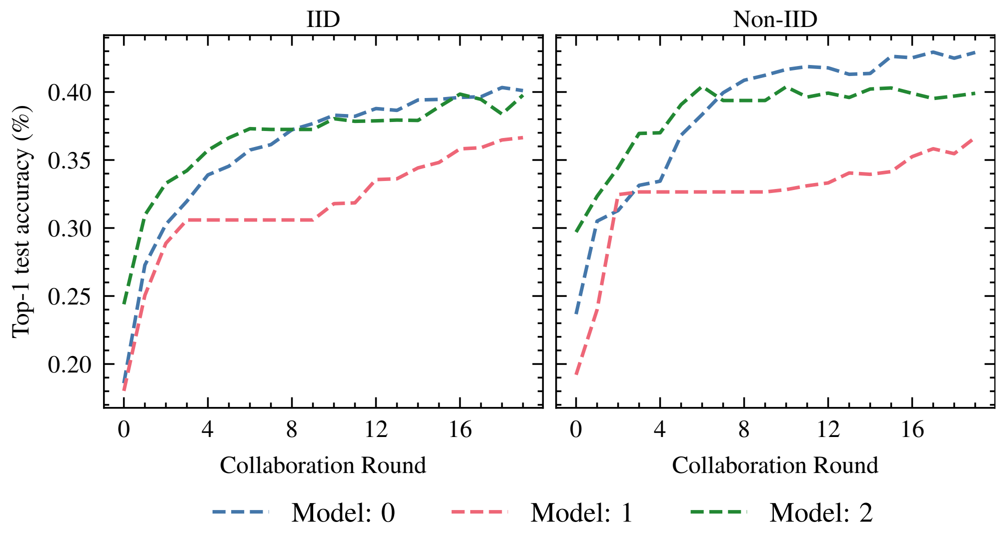


Figure 5.12: Test accuracy for FedDF on Shakespeare Dataset for IID and Non-IID scenarios. Refer to table 5.5 for model specifications.

### 5.3.1 FedMD

#### Time

For FedMD, we analyze the time taken by the participating clients in each step of the algorithm as well as the total time taken for completing each collaboration round comprising all the steps for all three datasets. In these graphs, we only analyze the time taken for steps involved in every communication round and not the initial one-time pre-training process.

On the MNIST dataset, we observe in figure 5.13 that each complete round takes approximately 360 to 390 seconds with more than 40% of the time being taken by the single aggregator function since we aggregate public dataset prediction logits from all the 100 participating clients in each round. Other than that, the revisit and digest steps take a similar amount of time and the communicate step being the fastest since it only involves a forward pass inference on the public dataset by all the clients. On the CIFAR dataset, as demonstrated in figure 5.14, the overall round time is relatively less compared to MNIST mainly because of the faster revisit step because of fewer local revisit epochs and a relatively smaller private dataset. Other than that, all the steps look similar in proportion with the aggregate step taking the largest amount of time. Finally, looking at figure 5.15 for the Shakespeare dataset, we observe a significant skew of timings with the majority of the chunk i.e. more than 90% being taken by the revisit step since the LSTM text models require a much greater number of epochs for this local training step in contrast to the convolutional neural networks for images.

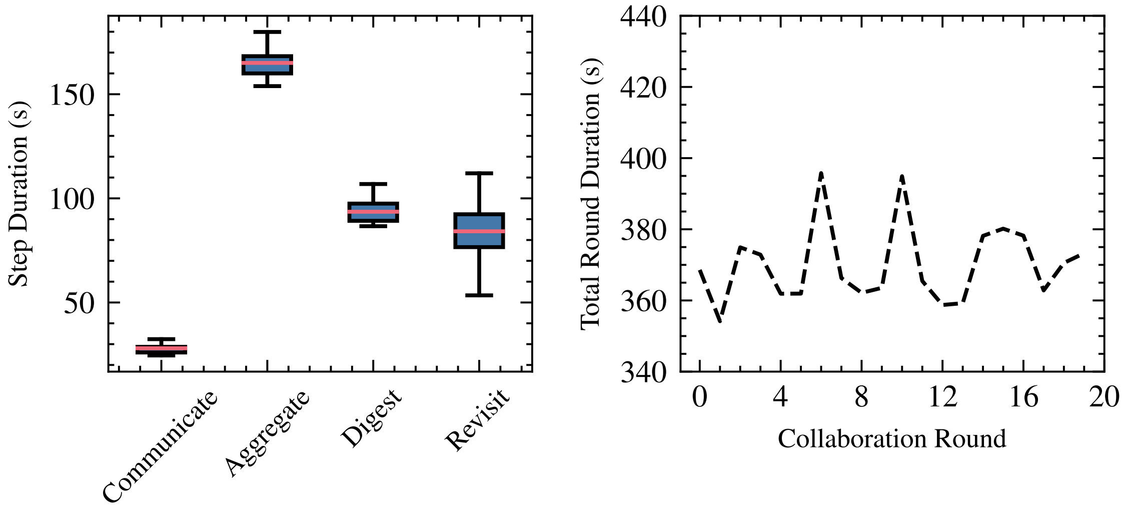


Figure 5.13: Individual step duration and overall round duration for FedMD on the MNIST dataset

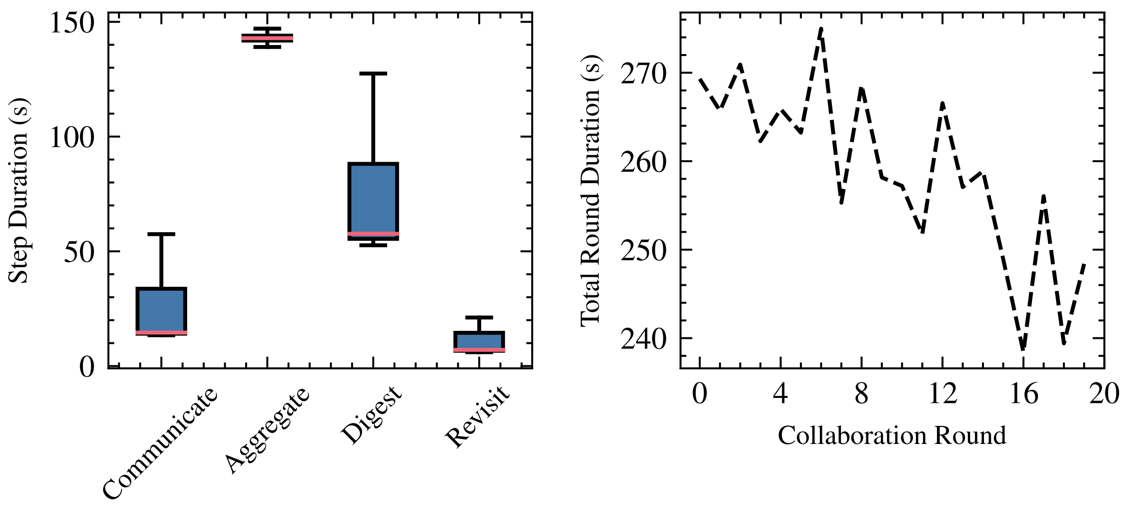


Figure 5.14: Individual step duration and overall round duration for FedMD on the CIFAR dataset

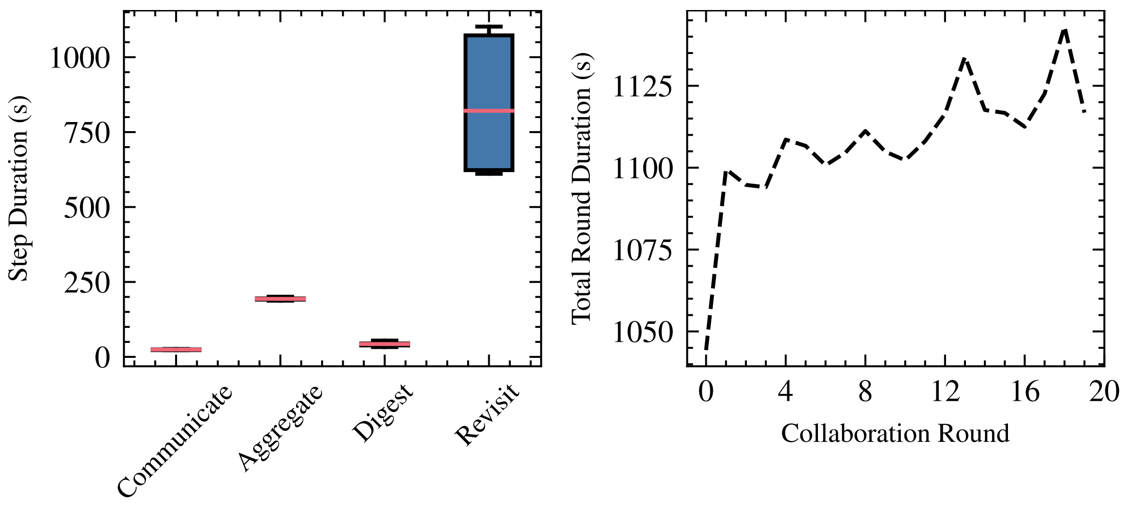


Figure 5.15: Individual step duration and overall round duration for FedMD on the Shakespeare dataset

#### Cost

In this section, we perform a fine-grained analysis of the FaaS costs for all the FedMD experiments. We factor in the total client as well as the aggregator function invocation times for the various steps in order to calculate these costs. We use the Google Cloud function cost computation model [15] to convert these execution times into expected costs.

Table 5.9 shows the total function execution durations as well as the cost for each execution incorporating the hardware capabilities of those functions for the three different datasets. For MNIST and CIFAR datasets, we observe that a major part of the total cost is due to the Digest step since it is executed on each client for every round. On the Shakespeare dataset, we observe a high increase in the costs which can be attributed to the revisit step as also evident from the timing graphs in the previous section.

### 5.3.2 FedDF

FedDF is an algorithm with each round majorly involving two steps that are the client side private training and the knowledge distillation process that takes place in the aggregator function invocations. In contrast to FedMD, we only use a small subset of

| Dataset | Metric | Aggregate | Communicate | Revisit | Transfer Learning (Private) | Digest | Overall |
|---|---|---|---|---|---|---|---|
| MNIST | Duration (min) | 55.2 | 378.4 | 867 | 42.5 | 1329.8 | **2672.9** |
| | Cost ($) | 0.37 | 1.32 | 3.02 | 0.15 | 4.63 | **9.49** |
| CIFAR | Duration (min) | 47.7 | 579.4 | 111.6 | 6.5 | 746.4 | **1491.6** |
| | Cost ($) | 0.32 | 2.02 | 0.39 | 0.02 | 2.6 | **5.35** |
| Shakespeare | Duration (min) | 64.5 | 300.8 | 5945.2 | 122.7 | 656.3 | **7089.5** |
| | Cost ($) | 0.43 | 1.05 | 20.69 | 0.43 | 2.28 | **24.88** |

Table 5.9: Overall client execution time and cost analysis for FedMD on all datasets.

clients for training in each round which leads to significantly lower overall execution time and cost.

### Time

For the MNIST and CIFAR dataset, figure 5.16a and 5.16b shows the total duration for each round along with the individual client and knowledge distillation components. Also, the highlighted area around these lines shows the variation for several runs across different levels of data heterogeneity. We see that a major part i.e. more than 90% of the total round duration is taken by the knowledge distillation process that takes place in the aggregator functions for each unique model architecture. Overall, each round takes about 400-600 seconds on average to complete for both datasets. For Shakespeare, figure 5.17 shows that the knowledge distillation and the client training have similar duration because of the high number of local epochs required in the client private training for LSTM networks. Consequently, this leads to higher overall round durations with each round taking around 500-1100 seconds. Also, it is worth noting that the knowledge distillation duration is in a constant range for all three datasets and only the private client training duration increases the overall round time beyond this duration.

### Cost

In terms of cost, table 5.10 shows the total aggregator and client training durations for FedDF across all three tasks. In contrast to FedMD, the total costs for FedDF are relatively lower and close for all experiments and this is mainly because only a certain fraction of clients take part in each communication round which is selected by our intelligent selection algorithm explained in section 4.3. We perform a deeper

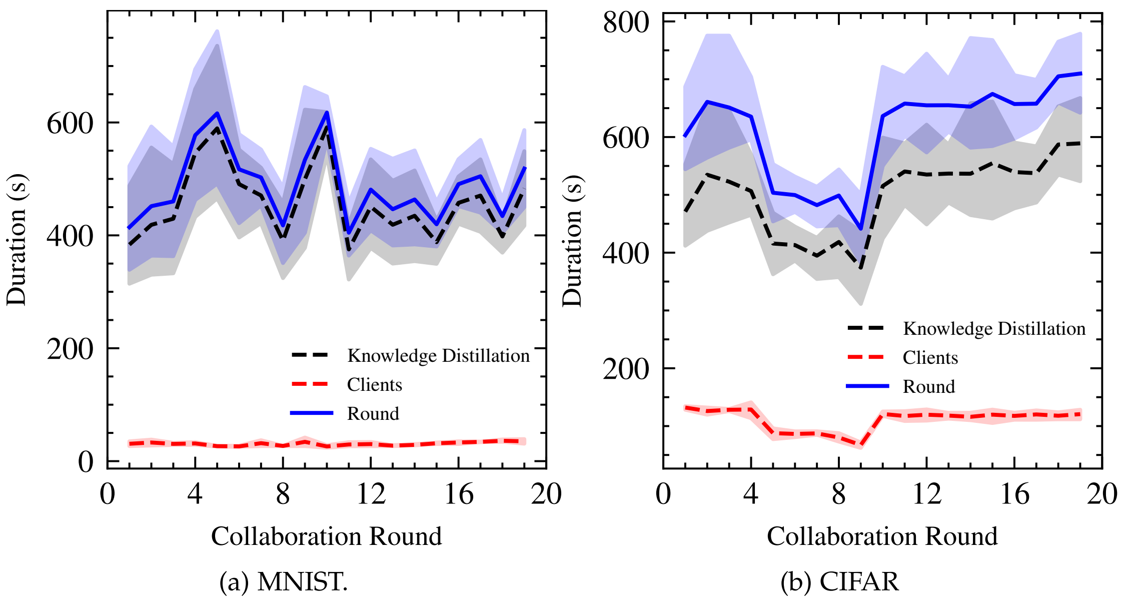


Figure 5.16: Collaboration round duration for FedDF

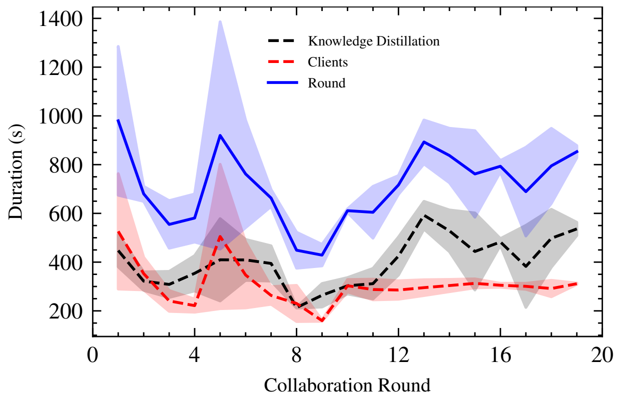


Figure 5.17: Collaboration round duration for FedDF on Shakespeare dataset.

quantitative analysis between FedDF and FedMD in terms of costs in the upcoming section 5.4.

| Dataset | Metric | Aggregators | Clients | Overall |
|---|---|---|---|---|
| MNIST | Duration (min) | 152 | 56.4 | **208.4** |
| | Cost ($) | 6.07 | 0.2 | **6.27** |
| CIFAR | Duration (min) | 163.87 | 213.55 | **377.42** |
| | Cost ($) | 5.46 | 0.74 | **6.2** |
| Shakespeare | Duration (min) | 134.4 | 466.9 | **601.3** |
| | Cost ($) | 2.68 | 1.62 | **4.3** |

Table 5.10: Overall client execution time and cost analysis for FedDF on all datasets

## 5.4 Quantitative comparison - FedMD and FedDF

In this section, we compare the FedMD and FedDF algorithms quantitatively in terms of performance, experiment execution duration, and FaaS execution costs across all three datasets i.e. MNIST, CIFAR, and Shakespeare. Table 5.11 shows the maximum Top-1 test accuracy that was achieved during the 20 rounds of collaborative training by each unique model architecture for both FedMD and FedDF on three datasets. For simplicity, we keep the data heterogeneity alpha level equal to 1 for all the comparisons since that simulates a standard non-i.i.d. data distribution scenario and gives us a good picture of the algorithm's robustness towards data heterogeneity. The model architecture level comparison is fair since we use the same architectures and client distribution for both algorithms. In the case of MNIST, FedDF has better accuracy levels for all model types than FedMD with an average performance improvement of 5% across the 6 unique model architectures. For CIFAR, we see a similar trend except for model 1 where FedMD outperforms FedDF, however, we see a positive difference of 3.2% on average across all 5 unique model architectures for FedDF when compared with FedMD in this case as well. Finally, for the Shakespeare language modeling task, FedDF again outperforms FedMD by an average of 5% across the three unique model architectures.

Table 5.12 compares both the algorithms regarding the complete experiment execution duration and the computation costs incurred by FaaS invocations during the

| | Maximum Top-1 test accuracy (%) | | | | | |
|---|---|---|---|---|---|---|
| | MNIST | | CIFAR | | Shakespeare | |
| **Model ID** | FedMD | FedDF | FedMD | FedDF | FedMD | FedDF |
| 0 | 0.86 | **0.90** | 0.40 | **0.52** | 0.33 | **0.43** |
| 1 | 0.86 | **0.93** | **0.58** | 0.52 | **0.38** | 0.37 |
| 2 | 0.90 | **0.96** | 0.54 | **0.55** | 0.34 | **0.40** |
| 3 | 0.89 | **0.96** | 0.51 | **0.55** | - | - |
| 4 | 0.92 | **0.93** | 0.51 | **0.56** | - | - |
| 5 | 0.91 | **0.96** | - | - | - | - |

Table 5.11: Comparison of maximum Top-1 test accuracy achieved across each dataset and client model architecture between FedMD and FedDF with data heterogeneity $\alpha = 1$. For model architecture details, please refer table 5.3 for MNIST, table 5.4 for CIFAR and table 5.5 for Shakespeare.

| | Duration (min) | | Cost ($) | |
|---|---|---|---|---|
| **Dataset** | FedMD | FedDF | FedMD | FedDF |
| MNIST | **123.4** | 154.5 | 9.49 | **6.27** |
| CIFAR | **86.3** | 194 | **5.35** | 6.2 |
| Shakespeare | 369.3 | **226.2** | 24.88 | **4.3** |

Table 5.12: Comparison of overall experiment duration and FaaS invocation cost between FedMD and FedDF

complete training process comprising 20 collaboration rounds. To make the comparison fair, we do not include the time and costs of Ray-based pre-training for the FedMD algorithm but it should be kept in mind when considering the complete execution costs. Also, we average the durations and costs across experiments with different data heterogeneity levels to get more confident estimations. In terms of execution duration, FedMD performs better for both the MNIST as well as CIFAR datasets since the FedDF knowledge distillation step takes up a major chunk of execution time i.e. around 77% of the total execution time. For Shakespeare, FedMD takes relatively more execution time mainly due to its *revisit* step which requires a higher number of local epochs for text data modality compared to image classification tasks.

In terms of FaaS costs, we see relatively stable costs for FedDF across all three tasks in contrast to FedMD which incurs significantly high costs for the Shakespeare task that is mainly due to the expensive **revisit** step because it involves training for a high number of local epochs by each participating client. However, in the case of FedDF, we only select a subset of clients for each training round and the aggregation process takes place in parallel for these selected clients leading to less costs. To summarize, in the case of MNIST and Shakespeare, FedDF costs 34% and 82.7% less compared to FedMD, and for CIFAR, FedDF costs around 16% more compared to FedMD.

# 6 Conclusion and Future Work

In this work, our main focus was to extend the existing serverless federated learning system FedLess[23] to allow the participating clients to have heterogenous model architectures and still collaborate with each other to achieve good performance. In order to achieve this, we explored various IaaS-based knowledge distillation techniques and finally implemented two of these techniques i.e. FedMD[38] and FedDF[42] based on their performance and the ease of integration into our current serverless federated learning platform. We faced some limitations with FedMD in the serverless paradigm since it involves multiple client invocations and most of the knowledge distillation process takes place on the client side which is not ideal due to limited processing and function execution times. Also, it involves calling all or at least a major chunk of the participating clients for each round which is not scalable when we are dealing with thousands of client functions. FedDF on the other side, being a completely server-side knowledge distillation algorithm and with only a fraction of clients participating in each round was more natural and simpler to integrate into our existing system and also yielded a better overall performance in most of our experiments. Overall, we were able to demonstrate that similar performance levels can be achieved without much cost and overall execution time differences in serverless federated learning systems with heterogeneous client model architectures when compared to our predecessors FedLesScan[17] and FedLess[23] that only allow homogeneous architectures.

Secondly, we performed a thorough analysis of the impact and robustness of different data heterogeneity levels among the participating clients for both algorithms using a Dirichlet distribution data sampling approach. Also, we enhanced the existing FedLesScan[17], an adaptive clustering-based intelligent client selection algorithm to perform the client selection by also taking their model heterogeneity into account and performing clustering on a client-model architecture level rather than just client level.

Finally, we performed a detailed analysis of both algorithms on different learning tasks and different levels of data heterogeneity in order to understand their performance as well as the durations and associated FaaS costs for each of them. We were able to achieve good results in terms of both performance as well as execution costs, however, since these algorithms were developed mainly for IaaS-based federated learning systems, we believe that better results can be achieved by spending more effort on tuning the many complex parameters of these algorithms to work better in the time and computation constrained serverless environments.

It is worth exploring some more advanced data-free knowledge distillation algorithms like FedGen[60] in a serverless setting since they do not require separate public transfer datasets which can result in a much more robust self-contained collaborative training process. In the current system, both our knowledge distillation implementations are synchronous and require central aggregation functions for the distillation process which is a bottleneck. More sophisticated decentralized knowledge distillation algorithms like [8] where client functions can invoke and distill knowledge to their neighboring clients can help eliminate this bottleneck and lead to better results in terms of efficiency.

To conclude, considering the increasing usage of edge devices of varying computational capacities across the world, enabling client model heterogeneity in serverless federated learning settings is something that needs to be tackled further and we believe that there is significant research potential in developing federated knowledge distillation algorithms tailored to serverless environments.

# Abbreviations

| | |
|---|---|
| **ML** | Machine Learning |
| **FL** | Federated Learning |
| **FaaS** | Function-as-a-Service |
| **IaaS** | Infrastructure-as-a-Service |
| **i.i.d.** | independent and identically distributed |
| **KD** | Knowledge Distillation |
| **NLP** | Natural Language Processing |
| **CV** | Computer Vision |
| **GAN** | Generative adversarial network |

# List of Figures

1.1 Impact of increasing data heterogeneity on the test accuracy of the FedMD distributed learning algorithm for MNIST and CIFAR image classification tasks . . . 3

2.1 Standard FL setup . . . 9

3.1 Original architecture of FedKeeper [11] . . . 15
3.2 Original architecture of FedLess [23] . . . 16
3.3 Modified architecture of FedLess[11] for FedLesScan[17] . . . 17
3.4 Original algorithm for FedMD[38] . . . 22
3.5 Original algorithm of FedDF[38] for heterogeneous client architectures . 23

4.1 Serverless training flow for FedMD . . . 30
4.2 Serverless training flow for FedDF . . . 34
4.3 Modified architecture of FedLesScan [17]. Enhancements and modifications are highlighted in green. . . . 36

5.1 FedMD/FedDF Private Client Data Distribution for MNIST Dataset . . 41
5.2 FedMD Private Client Data Distribution for CIFAR-100 (Subset) Dataset 42
5.3 FedDF Private Client Data Distribution for CIFAR-10 Dataset . . . 44
5.4 Test accuracies for FedMD on MNIST Dataset with different data heterogeneity levels. Refer to table 5.3 for model specifications. . . . 50
5.5 Test loss for FedMD on MNIST Dataset with different data heterogeneity levels. Refer to table 5.3 for model specifications. . . . 51
5.6 Test accuracy for FedMD on CIFAR Dataset with different data heterogeneity levels. Refer to table 5.4 for model specifications. . . . 52
5.7 Test loss for FedMD on CIFAR Dataset with different data heterogeneity levels. Refer to table 5.4 for model specifications. . . . 53
5.8 Test accuracy for FedMD on Shakespeare Dataset for IID and Non-IID scenarios. Refer to table 5.5 for model specifications. . . . 54
5.9 Test loss for FedMD on Shakespeare Dataset for IID and Non-IID scenarios. Refer to table 5.5 for model specifications. . . . 54

5.10 Test accuracy for FedDF on MNIST Dataset with different data heterogeneity levels. Refer to table 5.3 for model specifications. . . . . . . . . . 56
5.11 Test accuracy for FedDF on CIFAR Dataset with different data heterogeneity levels. Refer to table 5.4 for model specifications. . . . . . . . . . 57
5.12 Test accuracy for FedDF on Shakespeare Dataset for IID and Non-IID scenarios. Refer to table 5.5 for model specifications. . . . . . . . . . . . 58
5.13 Individual step duration and overall round duration for FedMD on the MNIST dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.14 Individual step duration and overall round duration for FedMD on the CIFAR dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.15 Individual step duration and overall round duration for FedMD on the Shakespeare dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
5.16 Collaboration round duration for FedDF . . . . . . . . . . . . . . . . . . . 62
5.17 Collaboration round duration for FedDF on Shakespeare dataset. . . . . 62

# List of Tables

3.1 Evaluated model heterogeneous KD techniques . . . . . . . . . . . . . . . 20

4.1 Example of client model configuration for convolutional networks . . . 33

5.1 Summary of evaluation tasks and datasets for serverless FedMD . . . . 40
5.2 Summary of evaluation tasks and datasets for serverless FedDF . . . . . 43
5.3 Client Convolutional Neural Network Models for MNIST . . . . . . . . 45
5.4 Client Convolutional Neural Network Models for CIFAR . . . . . . . . . 45
5.5 1-Layer LSTM Network Models for Shakespeare . . . . . . . . . . . . . . 45
5.6 Hyper Parameters for FedMD . . . . . . . . . . . . . . . . . . . . . . . . . 46
5.7 Client Private Training Hyper Parameters for FedDF . . . . . . . . . . . 47
5.8 FedDF Aggregator Hyperparameters . . . . . . . . . . . . . . . . . . . . 47
5.9 Overall client execution time and cost analysis for FedMD on all datasets. 61
5.10 Overall client execution time and cost analysis for FedDF on all datasets 63
5.11 Comparison of maximum Top-1 test accuracy achieved across each dataset and client model architecture between FedMD and FedDF with data heterogeneity $\alpha = 1$. For model architecture details, please refer table 5.3 for MNIST, table 5.4 for CIFAR and table 5.5 for Shakespeare. . 64
5.12 Comparison of overall experiment duration and FaaS invocation cost between FedMD and FedDF . . . . . . . . . . . . . . . . . . . . . . . . . 64

# Bibliography

[1] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton. *Large scale distributed neural network training through online distillation*. 2020. arXiv: `1804.03235 [cs.LG]`.

[2] E. Arani, F. Sarfraz, and B. Zonooz. *Noise as a Resource for Learning in Knowledge Distillation*. 2020. arXiv: `1910.05057 [cs.LG]`.

[3] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary. "Federated Learning with Personalization Layers." In: *CoRR* abs/1912.00818 (2019). arXiv: `1912.00818`.

[4] M. Asad, A. Moustafa, and T. Ito. *Federated Learning Versus Classical Machine Learning: A Convergence Comparison*. 2021. arXiv: `2107.10976 [cs.LG]`.

[5] *AWS Cognito*. `https://aws.amazon.com/cognito/`. Accessed: 2023-03-27.

[6] *AWS Lambda*. `https://aws.amazon.com/lambda/`. Accessed: 2023-03-24.

[7] *Azure functions – Serverless functions in computing: Microsoft Azure*. `https://azure.microsoft.com/en-us/products/functions/`. Accessed: 2023-03-24.

[8] I. Bistritz, A. J. Mann, and N. Bambos. "Distributed Distillation for On-Device Learning." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[9] S. Caldas, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. "LEAF: A Benchmark for Federated Settings." In: *arXiv preprint arXiv:1812.01097* (2018).

[10] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar. "LEAF: A Benchmark for Federated Settings." In: *CoRR* abs/1812.01097 (2018). arXiv: `1812.01097`.

[11] M. Chadha, A. Jindal, and M. Gerndt. "Towards Federated Learning Using FaaS Fabric." In: *Proceedings of the 2020 Sixth International Workshop on Serverless Computing*. WoSC'20. Delft, Netherlands: Association for Computing Machinery, 2021, pp. 49–54. ISBN: 9781450382045. DOI: `10.1145/3429880.3430100`.

[12] Y. Chang, S. Laridi, Z. Ren, G. Palmer, B. W. Schuller, and M. Fisichella. *Robust Federated Learning Against Adversarial Attacks for Speech Emotion Recognition*. 2022. arXiv: `2203.04696 [cs.SD]`.

[13] Y. J. Cho, A. Manoel, G. Joshi, R. Sim, and D. Dimitriadis. "Heterogeneous Ensemble Knowledge Transfer for Training Large Models in Federated Learning." In: (2022). DOI: `10.48550/ARXIV.2204.12703`.

[14] *Cloud functions - Google Cloud*. `https://cloud.google.com/functions`. Accessed: 2023-03-24.

[15] *Cloud Functions pricing*. `https://cloud.google.com/functions/pricing`. Accessed: 2023-04-02.

[16] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. "EMNIST: an extension of MNIST to handwritten letters." In: *CoRR* abs/1702.05373 (2017). arXiv: `1702.05373`.

[17] M. Elzohairy, M. Chadha, A. Jindal, A. Grafberger, J. Gu, M. Gerndt, and O. Abboud. "FedLesScan: Mitigating Stragglers in Serverless Federated Learning." In: *arXiv preprint arXiv:2211.05739* (2022).

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[19] *funcX*. `https://funcx.org/`. Accessed: 2023-03-26.

[20] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. *Born Again Neural Networks*. 2018. arXiv: `1805.04770 [stat.ML]`.

[21] H. Gibet Tani and C. El Amrani. "Cloud Computing CPU Allocation and Scheduling Algorithms using CloudSim Simulator." In: *International Journal of Electrical and Computer Engineering (IJECE)* 6 (Aug. 2016), p. 1866. DOI: `10.11591/ijece.v6i4.pp1866-1879`.

[22] J. Gou, B. Yu, S. J. Maybank, and D. Tao. "Knowledge Distillation: A Survey." In: *CoRR* abs/2006.05525 (2020). arXiv: `2006.05525`.

[23] A. Grafberger, M. Chadha, A. Jindal, J. Gu, and M. Gerndt. "FedLess: Secure and Scalable Federated Learning Using Serverless Computing." In: *2021 IEEE International Conference on Big Data (Big Data)*. 2021, pp. 164–173. DOI: `10.1109/BigData52589.2021.9672067`.

[24] Q. Guo, X. Wang, Y. Wu, Z. Yu, D. Liang, X. Hu, and P. Luo. "Online Knowledge Distillation via Collaborative Learning." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[25] N. Haim, G. Vardi, G. Yehudai, O. Shamir, and M. Irani. *Reconstructing Training Data from Trained Neural Networks*. 2022. arXiv: `2206.07758 [cs.LG]`.

[26] C. He, M. Annavaram, and S. Avestimehr. "Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge." In: (2020). DOI: `10.48550/ARXIV.2007.14513`.

[27] G. Hinton, O. Vinyals, and J. Dean. "Distilling the Knowledge in a Neural Network." In: (2015). DOI: `10.48550/ARXIV.1503.02531`.

[28] L. Hu, H. Yan, L. Li, Z. Pan, X. Liu, and Z. Zhang. "MHAT: An efficient model-heterogenous aggregation training scheme for federated learning." In: *Information Sciences* 560 (2021), pp. 493–503. ISSN: 0020-0255. DOI: `https://doi.org/10.1016/j.ins.2021.01.046`.

[29] S. Itahara, T. Nishio, Y. Koda, M. Morikura, and K. Yamamoto. "Distillation-Based Semi-Supervised Federated Learning for Communication-Efficient Collaborative Training with Non-IID Private Data." In: *IEEE Transactions on Mobile Computing* (2021), pp. 1–1. DOI: `10.1109/tmc.2021.3070013`.

[30] K. Jayaram, V. Muthusamy, G. Thomas, A. Verma, and M. Purcell. *$\lambda$-FL: Serverless Aggregation For Federated Learning*. 2022.

[31] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. May 2015. DOI: `10.17487/RFC7519`.

[32] J. M. Joyce. "Kullback-Leibler Divergence." In: *International Encyclopedia of Statistical Science*. Ed. by M. Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722. ISBN: 978-3-642-04898-2. DOI: `10.1007/978-3-642-04898-2_327`.

[33] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. *Federated Learning: Strategies for Improving Communication Efficiency*. 2017. arXiv: `1610.05492 [cs.LG]`.

[34] N. Kotsehub, M. Baughman, R. Chard, N. Hudson, P. Patros, O. Rana, I. Foster, and K. Chard. "FLoX: Federated Learning with FaaS at the Edge." In: *2022 IEEE 18th International Conference on e-Science (e-Science)*. 2022, pp. 11–20. DOI: `10.1109/eScience55777.2022.00016`.

[35] A. Krizhevsky, V. Nair, and G. Hinton. *CIFAR-10 (Canadian Institute for Advanced Research)*.

[36] A. Krizhevsky, V. Nair, and G. Hinton. *CIFAR-100 (Canadian Institute for Advanced Research)*.

[37] Y. LeCun and C. Cortes. "MNIST handwritten digit database." In: (2010).

[38] D. Li and J. Wang. "FedMD: Heterogenous Federated Learning via Model Distillation." In: (2019). DOI: `10.48550/ARXIV.1910.03581`.

[39] T. Li. *Federated Learning: Challenges, Methods, and Future Directions*. `https://blog.ml.cmu.edu/2019/11/12/federated-learning-challenges-methods-and-future-directions/`. Accessed: 2023-03-24.

[40] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. "Federated Learning: Challenges, Methods, and Future Directions." In: *CoRR* abs/1908.07873 (2019). arXiv: `1908.07873`.

[41] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. "Tune: A Research Platform for Distributed Model Selection and Training." In: *arXiv preprint arXiv:1807.05118* (2018).

[42] T. Lin, L. Kong, S. U. Stich, and M. Jaggi. "Ensemble Distillation for Robust Model Fusion in Federated Learning." In: 33 (2020). Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, pp. 2351–2363.

[43] O. Ltd. *Home*. `https://www.openfaas.com/`. Accessed: 2023-03-24.

[44] A. B. de Luca, G. Zhang, X. Chen, and Y. Yu. *Mitigating Data Heterogeneity in Federated Learning with Data Augmentation*. 2022. arXiv: `2206.09979 [cs.LG]`.

[45] B. McMahan and D. Ramage. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. Accessed: 2023-03-24.

[46] H. Mobahi, M. Farajtabar, and P. L. Bartlett. *Self-Distillation Amplifies Regularization in Hilbert Space*. 2020. arXiv: `2002.05715 [cs.LG]`.

[47] A. Mora, I. Tenison, P. Bellavista, and I. Rish. *Knowledge Distillation for Federated Learning: a Practical Guide*. 2022. arXiv: `2211.04742 [cs.LG]`.

[48] *Open source serverless cloud platform*. `https://openwhisk.apache.org/`. Accessed: 2023-03-24.

[49] *Ray Homepage*. `https://www.ray.io/`. Accessed: 2022-10-30.

[50] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. *FitNets: Hints for Thin Deep Nets*. 2015. arXiv: `1412.6550 [cs.LG]`.

[51] I. H. Sarker. "Machine Learning: Algorithms, Real-World Applications and Research Directions." In: *SN Computer Science* 2.3 (2021), p. 160. DOI: `10.1007/s42979-021-00592-x`.

[52] H. Shafiei and A. Khonsari. "Serverless Computing: Opportunities and Challenges." In: *CoRR* abs/1911.01296 (2019). arXiv: `1911.01296`.

[53] T. Sun, D. Li, and B. Wang. "Decentralized Federated Averaging." In: (2021). DOI: 10.48550/ARXIV.2104.11375.

[54] A. Z. Tan, H. Yu, L. Cui, and Q. Yang. "Towards Personalized Federated Learning." In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–17. DOI: 10.1109/TNNLS.2022.3160699.

[55] K. Team. *Keras documentation: Character-level text generation with LSTM*.

[56] S. Thapa. *On effects of Knowledge Distillation on Transfer Learning*. 2022. arXiv: 2210.09668 [cs.LG].

[57] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao. "A survey on federated learning." In: *Knowledge-Based Systems* 216 (2021), p. 106775. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.106775.

[58] L. Zhang, C. Bao, and K. Ma. "Self-Distillation: Towards Efficient and Compact Neural Networks." In: *IEEE Transactions on Pattern Analysis amp; Machine Intelligence* 44.08 (Aug. 2022), pp. 4388–4403. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3067100.

[59] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu. *Deep Mutual Learning*. 2017. arXiv: 1706.00384 [cs.CV].

[60] Z. Zhu, J. Hong, and J. Zhou. "Data-Free Knowledge Distillation for Heterogeneous Federated Learning." In: (2021). DOI: 10.48550/ARXIV.2105.10056.