Imperial College London

MENG INDIVIDUAL PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

FedGDKD: Federated GAN-Based Data-Free Knowledge Distillation for Heterogeneous Models

Author:
Arjun Banerjee

Supervisor:
Prof. Kin Leung

Second Marker:
Dr. Jonathan Passerat-Palmbach

June 20, 2022

Abstract

Federated Learning (FL) provides organisations with a tool to collaboratively train a single Deep Learning model while keeping their data private. Furthermore, allowing organisations to personalise and privatise their own model brings challenges to FL. Existing methods utilise Knowledge Distillation (KD) as a vehicle for extracting and transferring knowledge in heterogeneous model FL (HMFL) but this requires a shared *transfer* dataset. Data-based methods ask the partipants to assemble this *transfer* set through a proxy dataset from a similar domain or, if infeasible, require participants to share a portion of their private data. Data-free methods assemble the *transfer* set through synthetic data generation, however, this is a cumbersome process.

We introduce a novel algorithm *Federated GAN-based Data-Free Knowledge Distillation* (FedGDKD) that can collaboratively train heterogeneous Deep Learning models whilst maintaining privacy of both the data and model itself in a data-free manner. This method uses a conditional generative adversarial network (cGAN) to produce synthetic data that can be used, in place of sensitive data, to transfer knowledge between heterogeneous models. In developing this algorithm, we also introduce novel method of training a federated cGAN and formulate novel training objectives that further improves performance and efficiency over the state-of-the-art with up to an 82.0% reduction in FID Score. We empirically evaluate the performance through measuring the average top-1 accuracy across varying degrees of client data heterogeneity and find that our method improves the most upon its baseline when compared to state-of-the-art HMFL methods. Additionally, we evaluate communication and computational efficiency by approximating complexity using asymptotic notation, finding that our method improves upon the data-free state-of-the-art. Then, we evaluate privacy through formal discussion of leakage present and find that our method provides the best privacy guarantees in the scenario of this project. Finally, we show that overall our method provides the best balance of performance, efficiency and privacy compared to HMFL state-of-the-art methods.

Deep Learning has become a prevalent tool for solving problems in today's world. FedGDKD provides an iterative step forwards to applying collaborative Deep Learning in sensitive industries such as healthcare.

Acknowledgements

First, I would like to thank my supervisor, Prof. Kin Leung, who helped guide me to the completion of this project. Through regular meetings and discussions, they have helped me navigate the vast field that is Federated Learning. A special thanks also goes to my second marker, Dr. Jonathan Passerat-Palmbach, who toward the beginning of my project helped provide feedback to solidify my project's direction.

Furthermore, I would like to express how grateful I am to my family and friends for the support they have given me throughout my degree. In particular my parents: Dr. Asok K Banerjee and Dr. Soma D Banerjee, who have sacrificed so much for me. Without them, I would not be the person I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, Deep Learning (DL) has been a powerful tool in solving complex tasks that would be too difficult to solve analytically. Some prominent examples are: enabling autonomous vehicles, predicting how proteins fold (AlphaFold[6]) and smart virtual assistants. However, a limiting factor to the widespread application of Deep Learning is that it requires large amounts of data to train effectively.

In industries such as healthcare, insurance and finance, organisations may operate with highly sensitive data that must be kept private. Privacy-Preserving Machine Learning (PPML) [7] are methods of applying machine learning (and deep learning) while maintaining privacy guarantees. However, in the case where a single organisation does not have enough data, what can they do? Federated Learning (FL) [8] is a distributed PPML technique where a shared DL model is trained through the collaboration of various parties without sharing their data. This is done by training a copy of this DL model locally at each party, then aggregating its parameters in a centralised location to construct a global model. This global model will then have effectively trained on all of the parties' private data.

However, what if there are further constraints on the DL model architecture that differ between parties? This could be due to restrictions or preference that could also mean this DL model's details are sensitive information. This leads to the problem of performing FL over heterogeneous models and keeping them private. DL model parameter aggregation has become infeasible for this task. Indeed, how else can we aggregate what each local model has learnt?

This brings into light a fundamental question about DL models which are implemented via artificial neural networks. What do their parameters or layers mean? The lack of interpretability of DL models makes this a difficult problem as there are many factors that can constitute the "knowledge" stored in a model. Knowledge Distillation (KD) [9] is a paradigm that attempts to tackle this. KD is a model compression technique where a large deep learning model teaches what it knows to a smaller "student" model.

This project explores how to tackle the problem of Federated Learning in a heterogeneous model setting using Knowledge Distillation, whilst trying to improve privacy, performance and efficiency over existing methods.

1.1 Contributions

This project extends and explores the application of Federated Learning to the context of personalised DL models, more specifically heterogeneous neural network architectures. The contributions are as follows:

- We introduce a novel FL algorithm *Federated GAN-Based Data-Free Knowledge Distillation* (FedGDKD) in Chapter 4 that utilises a conditional generative adversarial network (cGAN) to perform data-free knowledge distillation to effectively train heterogeneous client models. We show that it performs well under various scenarios and is more efficient and private than some state-of-the-art methods.

- In the process of developing FedGDKD, we develop novel cGAN training objectives (Section 4.2.1), which does away with the need for a discriminator network and can operate with an unmodified classifier directly. This is the first method to our knowledge that implements this and further improves ease-of-use.

- Additionally, FedGDKD also proves to be a novel method of training cGANs in a FL setting (Section 5.2.5). Which, unlike existing state-of-the-art methods that share both generator and discriminator model parameters, only requires sharing the generator parameters. We show that this is more robust, private and communication efficient than the state-of-the-art.

1.2 Ethical Considerations

In this project, we explore Federated Learningc (FL), a privacy-preserving distributed machine learning (PPML) technique. In particular, we outline the use of FL with Deep Learning (DL) methods such as training Deep Neural Networks (DNN). Therefore, ethical, legal, and professional issues are those that are inherited from DL.

Firstly, DL was developed as a method to find automated feature representations from data without human intervention. This has led to better performing models that do not require human supervision. In doing so, a black-box solution is formed where the feature relations modelled by the DNN are not human-interpretable. This lack of transparency in the decision-making process means that, for highly critical applications, such as diagnosis in healthcare, not knowing how the decision was made is a major risk and can lead to catastrophic failures.

Furthermore, DL models are vulnerable to attacks[10] that can leak sensitive information or attacks that worsen the performance of the model. Regarding the latter point, *model poisoning* attacks are a key vulnerability in federated learning. This is where a malicious client can submit bad model updates or train on unrepresentative data that can induce bias towards certain groups of people and worsen the performance of a critical application. Bias can occur even if there are no malicious actors, and so proper collection and preparation of the training data need to be sourced to prevent this.

FL mitigates some privacy and legal issues, as clients do not need to share their data. This is important if the clients are globally distributed and have differing regulations such as GDPR compliance.

Chapter 2

Background

In this chapter, the stage is set for the rest of the project by providing introductions to relevant background theory and current work. To begin with, we provide an introduction to Deep Learning, discussing relevant concepts and privacy vulnerabilities. Then, we have a discussion on Federated Learning and the heterogeneous model setting. Moving on from this, preliminary information about Knowledge Distillation and its three main flavours is explored. Finally, a review on closely related works and the motivation behind this dissertation.

2.1 Deep Learning

Deep learning (DL) is a subclass of Machine Learning (ML) that mitigates the issue of extracting and learning features (representation) from raw data in conventional ML techniques. These features are learnt automatically rather than having to handcraft them. DL methods are representation-learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules (Artificial Neural Networks) that each transform the representation at one level (starting with the raw input) into a representation at a higher, more abstract level [11].

2.1.1 Artificial Neural Networks

The idea of replicating human learning has long been a goal in the field of Artificial Intelligence (AI). A key breakthrough in this area was to replicate the brain's biological neural networks, and so in the 1940s the concept of Artificial Neural Networks (ANNs) or Neural Networks (NNs) was created.

An ANN Figure 2.2 is generally composed of multiple interconnected elementary units called artificial neurons Figure 2.1. An artificial neuron represents a parameterised function that is a weighted sum of its input connections and a bias passed through an activation function (usually nonlinear) or more succinctly:

$$\hat{y} = a(\sum_{i=1..n} w_i x_i + b) \quad y, x_i, y_i, b \in \mathbb{R} \quad \forall i \quad a : \mathbb{R} \to \mathbb{R} \tag{2.1}$$

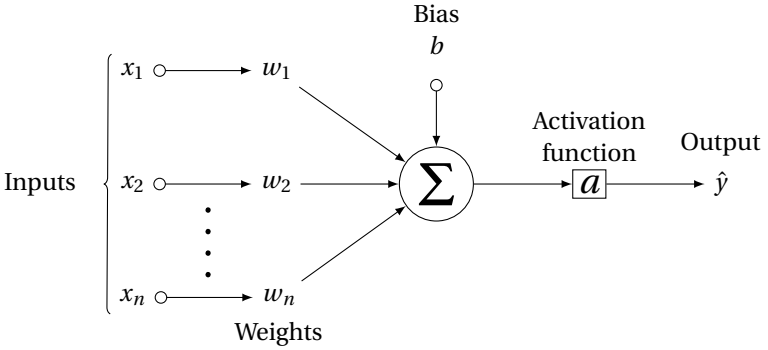


Figure 2.1: Artificial Neuron [2]

Conceptually, an ANN represents a parameterised function f_θ, where θ denotes the weights and biases of the network, that can approximate an arbitrary function $f : \mathbb{R}^M \to \mathbb{R}^N$. In the context of a problem such as image classification, the function f can be considered as the network's solution, taking an input image and outputting its guess as to what the image is of. Ideally, we would like this guess to be correct every time (100% accuracy) and so this means we need the network to approximate an optimal solution f_*.

ANNs are considered deep generally if they have multiple hidden layers - called Deep Neural Networks (DNNs). The width and depth of a network, when using nonlinear layers, can be correlated with the capacity of the model. We defined the capacity of the model as its computational complexity, the higher the capacity, the more complex the function it can represent. However, care needs to be taken when designing DNNs as they are prone to memorisation and overfitting of the input data. Overfitting is where there is high variance and low bias in the function approximated, reducing the generalisability of the network. This has the effect of high performance on training data but low on test or unseen data.

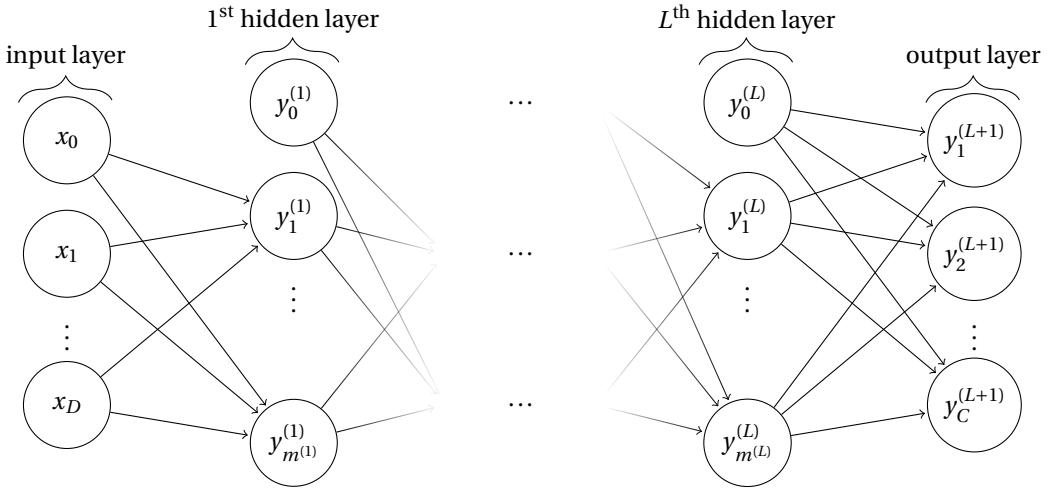


Figure 2.2: Network graph of a $(L+1)$-layer perceptron with D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units. [12]

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a branch of ANNs, inspired by the biological visual cortex, that are designed to process data in the form of multiple arrays [11]. Many data modalities can be expressed through multiple arrays such as: 1D for signals and sequences, including language; 2D for images or audio spectograms; and 3D for video or volumetric images. What makes CNNs good at processing signals are: local connections, shared weights, pooling, and the use of many layers.

Convolutional Layer

The convolutional layer is where filters/kernels (learned during training) are convolved[13] over the input to detect the presence of features across regions or receptive fields of the input. Local connectivity refers to the fact that the output array or feature map of the convolution is mapped to a receptive field rather than individual input values. Note that the weights of the filter remain fixed during the convolution, this is known as weight sharing. This means that the memory footprint of this layer is much lower than its fully connected counterpart.

Pooling Layer

Additionally, the pooling layer (downsampling) reduces the dimensions of the input. This is done in a similar manner to the convolutional layer through convolving an aggregation filter across the input. Examples of this are max-pooling and average-pooling, where the maximum/average value is filtered from the receptive field, respectively. The pooling layer, similarly to the convolutional layer, also exhibits local connectivity.

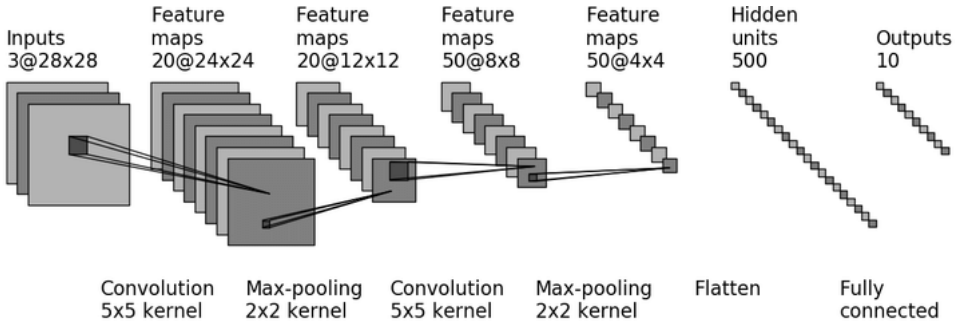


Figure 2.3: Structure of LeNet-5 CNN [3]

The LeNet-5 fig. 2.3 is a CNN designed to recognise simple digit images. We can decompose the architecture into two sections: the feature extractor (up to but not including the flatten layer) and the task specific network. As described before the idea of representation learning persists in CNNs too. The deeper the layers are in the feature extractor, the larger the receptive field becomes in the input thus the more abstract the features represented becomes. CNNs therefore benefit from depth and, due to their smaller memory footprint, scale better than other flavours of ANN.

2.1.3 Generative Adversarial Networks

In this thesis, generating synthetic data for training is an idea in solving the problem of data heterogeneity (Section 2.2.3). A key player in doing so is to utilise generative networks that can produce samples from a learned input distribution. A way to train these types of networks is to do so adversarially, where a network (generator) in charge of generating synthetic or fake samples is trying to fool a network (discriminator), designed to discriminate between real and fake samples, into thinking they are real. This is the premise of a generative adversarial network (GAN) [14]. If labels or input-output pairs are provided we call this a conditional GAN.

More formally, given a set of data instances X and a set of labels Y:

- The generator will capture the joint probability $p(X,Y)$, or just $p(X)$ if there are no labels.
- The discriminator will capture the conditional probability $p(Y|X)$.

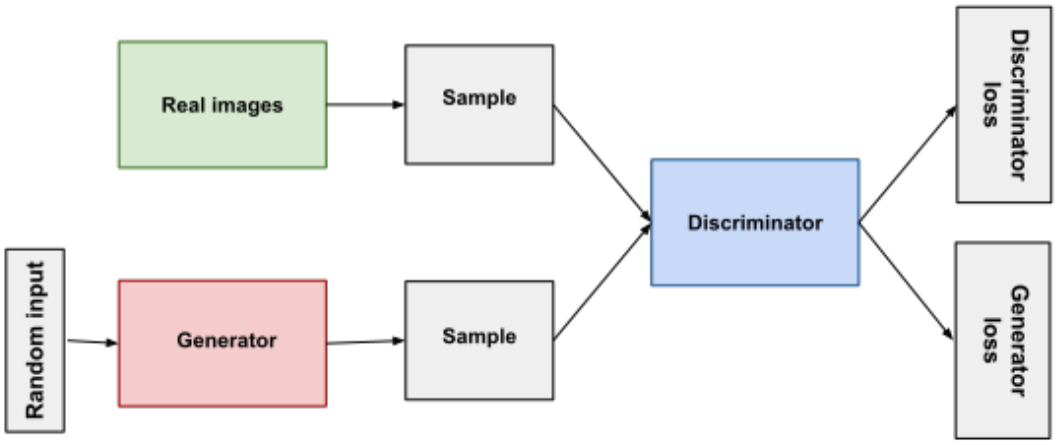


Figure 2.4: GAN structure [4]

As can be seen in Figure 2.4, we can view the set up for training a GAN. Both the discriminator and generator networks are trained at the same time, competing with each other. They are both trained in turn using the other network to inform parameter updates.

2.1.4 Supervised Learning

In this project, the supervised learning problem setting is explored. In this setting, at training time we have both the input x and the expected output y for all training samples. We then want to approximate the mapping $f(x) = y$. Supervised learning can further be broken down into classification and regression problems. Classification problems involve categorising the input data into one or more of the predetermined classes, e.g. identifying hand-written digits in images. Regression, on the other hand, is used to understand the relationships between dependent and independent variables, mainly focusing on tasks that involve predictions, e.g. sales forecasting.

2.1.5 Optimisation

As mentioned in Section 2.1.1, one would like ANNs to learn a mapping f_θ between input and output. f_θ is parameterised by the network weights and biases, so optimisation of the parameters θ needs to be performed to minimise the error between the f_θ and an optimal target mapping $f*$. This can be summarised by

$$\min_\theta J(\theta) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} L(f_\theta(x_i), y_i) = \mathbb{E}_{x,y} L(f_\theta(x), y) \tag{2.2}$$

where $|\mathcal{D}|$ is the size of the training set, $L(.)$ is the sample-wise loss/cost function that measures the error between the predicted value $\hat{y} = f_\theta(x)$ and the true value y. The aim is to minimise the expected loss for all pairs of x and y. Note that this is the describing the supervised setting Section 2.1.4.

Gradient Descent

One of the most popular optimisation techniques used to train ANNs is gradient descent. This method iteratively updates the parameters θ in the direction of steepest descent with respect to the gradient of the expected loss function $J(\theta)$ eq. (2.2):

$$\nabla_\theta J(\theta) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \nabla_\theta L(f_\theta(x_i), y_i) = \mathbb{E}_{x,y} \nabla_\theta L(f_\theta(x), y) \tag{2.3}$$

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} J(\theta_t) \tag{2.4}$$

where eq. (2.3) refers to the gradient of the expected loss function, eq. (2.4) refers to the update where α is the step size or learning rate and $\mathcal{D}$ is the entire training set. Usually, backpropagation [15] is used to calculate the gradient by application of the chain rule.

If the objective function eq. (2.2) is non-convex then the optimisation process will be sensitive to initialisation. Given a well-chosen step size, the process may converge to local minima. Another drawback of gradient descent is that it requires the whole training set to approximate the gradient. Therefore, for problems with extremely large (theoretically can be infinitely sized) training sets, this approach is computationally costly.

Stochastic Gradient Descent

Stochastic gradient descent was developed to remedy the cost of gradient descent. Rather than calculating the gradient using the full training set, a random subset $\mathcal{S}_t$ is chosen at each iteration/epoch to approximate the gradient instead. We can then alter the gradient calculation in eq. (2.3) to

$$\nabla_\theta J(\theta) = \frac{1}{|\mathcal{S}_t|} \sum_{i \in \mathcal{S}_t} \nabla_\theta L(f_\theta(x_i), y_i) = \mathbb{E}_{x,y} \nabla_\theta L(f_\theta(x), y) \quad \mathcal{S}_t \subseteq \mathcal{D} \tag{2.5}$$

We can define the size of $\mathcal{S}_t$ as the batch size, where this is set as a hyper parameter. Note that eq. (2.5) is an unbiased estimate of the gradient which is important for convergence properties. This is because $\mathcal{S}_t$ is uniformly sampled from the full training dataset $\mathcal{D}$ and so it will be identically and independently distributed (IID) with respect to $\mathcal{D}$: $\mathcal{S}_t \overset{i.i.d}{\sim} D$. This method of optimisation and extended methods are used in Federated Learning, however non-IID (Section 2.2.3) is common so this needs to be considered for convergence in the federated setting.

2.2 Federated Learning

Federated learning (FL)[8] was developed in 2016 to address the privacy concerns of big data and the exploitation of personal data. The goal of FL is to train a global model without the need of sharing data between entities. More formally, FL is a machine learning environment where multiple entities (clients) collaborate to solve a machine learning problem under the coordination of a central server. The data of each client is stored locally and not exchanged or transferred (decentralised data); instead, focused updates intended for immediate aggregation are used to achieve the learning objective [1]. See details in the federated learning problem setting in table A.1.

2.2.1 A Few Definitions

Client An individual or organisation that participates in FL.

Device A device refers to the physical hardware of a client.

Central server A server used to aggregate local models into a global model. Synonymous with: *central aggregation server*, *global server*.

2.2.2 A Federated Training Process

We can broadly generalise the steps of FL via the following steps:

1. **Initialisation:** The central server initialises a global model randomly or by pre-training with public data. The choice of architecture and initialisation is a non-trivial problem which is not relevant for this thesis.

2. **Client selection:** The central server samples from a set of clients that meet the eligibility requirements. For example, mobile phones that are connected, in a wifi connection and idle, to avoid impacting the user of the device[1]. FedCS [16] is an example of such a protocol.

3. **Broadcast:** The selected clients download the current weights and training program from the central server.

4. **Local/Client update:** Each selected client locally computes an update to the model by executing the training program, for example, running SGD on the local data (as in Federated Averaging [8]).

5. **Global update:** The selected clients send their model updates θ_i to the central server that aggregates them to construct an improved global model θ. For example, Federated Averaging [8] computes the average by weighting in proportion to each client's dataset size $|\mathcal{D}_i|$ against the total dataset size $|\mathcal{D}|$:

$$\theta = \sum_{i=1}^{N} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \theta_i \tag{2.6}$$

6. **Iteration:** Steps **2-5** are repeated until a terminating condition is met, i.e. convergence or the predefined maximum iterations have been met.

2.2.3 Limitations

Federated learning is indeed the future of privacy-preserving machine learning on the edge. However, there are many roadblocks in practical application. In the context of this thesis, the following are considered:

Data Heterogeneity: Non-IID Data

In the federated setting, one cannot assume that the data will be independently and identically distributed (IID) among clients. This is due to skews in data distribution such as in the case of two coffee shops tracking sales where one is located in London and another in Beijing. One could argue that the demographics and taste differences will affect the sales of each type of coffee bought. The IID sampling of the training data is important to ensure that the stochastic gradient is an unbiased estimate

of the full gradient. Thus, this is problematic as it can reduce performance of the global model due to the divergence of the local models resulting from Non-IID client data. That is, local models having the same initial parameters will converge to different local optima because of the heterogeneity in local data distribution. [17]

Various methods have been proposed to mitigate this issue, such as synthetically generating data using generative networks, using a public dataset for parameter updates, and weighting client updates, e.g. Federated Averaging (FedAvg) [8].

Heterogeneous System

It is highly possible that the clients participating in the algorithm will have different hardware and network constraints. Those with lower computational capacity will have slower local updates and may not even be able to participate if the model architecture is too large. These clients are likely to be ineligible to participate in learning, which coupled with the assumption of non-IID data (Section 2.2.3) can lead to bias in the data used to train the global model.

Personalisation

In standard FL, the goal is to train a global model that performs well on most FL clients. Compared to local training, the globally shared model trained through FL generalises well to unseen data as it is trained on large amounts of data. However, these models are designed to fit the "average client". They might therefore not perform well in the presence of statistical data heterogeneity (i.e. if the local data distribution of a client deviates significantly from the global data distribution). Enabling FL to deal with heterogeneous data is important given the non-IID (Section 2.2.3) nature of data that originate from clients in the real world. Besides data heterogeneity, FL also needs to deal with heterogeneity in device capabilities in edge computing applications. [18]

Additionally, clients may have different requirements or standards they may need to meet and so having flexibility in designing their local model is essential. This can arise in areas such as healthcare, finance, and supply chain management. An example of this is when several medical institutions would like to collaborate without sharing private patient data but also need to meet distinct specifications. This introduces the problem at the crux of this thesis: *Heterogeneous Model Federated Learning*.

2.3 Knowledge Distillation

Many recent advances in Deep Learning Section 2.1 has led to breakthroughs in applications of natural language processing, reinforcement learning, and computer vision. Some of these Deep Neural Networks (DNNs) that have been developed can have thousands of layers and can be cumbersome or difficult to deploy for real-time applications, especially on devices with limited resources (edge devices), such as video surveillance or autonomous driving cars [19]. One such avenue for building more efficient DNNs is Knowledge Distillation (KD)[9], a model compression technique that transfers "knowledge" from a large, cumbersome *teacher* model (or ensemble of teachers) to a smaller, more computationally efficient *student* model.

2.3.1 Approach

In the original paper by Hinton et al.[9], a simple approach was proposed for KD. The student minimises a weighted combination of two objectives

$$L_s := \alpha L_{Task} + (1-\alpha) L_{KD}, \quad \alpha \in [0,1) \tag{2.7}$$

where L_{Task} is the loss associated with the original task objective, which in the paper is the cross-entropy loss for image classification, and an added distillation loss term L_{KD} that encourages the student to match the teacher. In the original paper, the distillation loss is the cross-entropy between the student and teacher predictive distributions (discussed in Section 2.3.2). We can also see the distillation loss acting like a regularisation term or label smoothing which should further mitigate overfitting and improve generalisation.

The optimisation of the student model for this new objective is carried out using gradient-based optimisation techniques for neural networks. A set of held-out data samples, the *transfer set*, is shared between the teacher and student models to perform KD.

2.3.2 What Is The Knowledge In A DNN?

As discussed in Section 2.1, DNNs learn abstract representations of the input data to ultimately produce an output. This approach can be described as black-box due to their multilayer nonlinear structure. Therefore, how exactly do they store knowledge of the input data? There have been three main approaches to answering this question:

Response-Based Knowledge

This refers to the response of the last output layer of the teacher model. The main idea is to directly mimic the final prediction of the teacher model. This approach is used in the classic method of KD[9], which uses the logits (un-normalised classification prediction) vector z of the larger model as the *teacher knowledge*.

The distillation loss associated with this approach can be formulated as

$$L_{ResD}(z_t, z_s) = \mathcal{L}_R(z_t, z_s) \tag{2.8}$$

where $\mathcal{L}_R(.)$ indicates the divergence loss of logits, and z_t and z_s are the logits of the teacher and student models, respectively. [19]

In the original paper proposed by Hinton et al., [9] details a popular response-based knowledge implementation for image classification called soft targets. These soft targets are the probabilities that the input belongs to each class and is estimated by the a softmax function:

$$ST(z_i, \tau) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)} \tag{2.9}$$

where z_i is the logit for the i-th class, and a temparature factor τ is introduced to control the importance of each soft target. A higher temperature results in softer targets. When soft targets have high entropy, they provide much more information per training case than hard targets and much less variance in the gradient between training cases, so the small model can often be trained on much less data than the original cumbersome model and using a much higher learning rate [9]. The distillation loss using soft targets can then be rewritten as

$$L_{ResD}(ST(z_t, \tau), ST(z_s, \tau)) = \mathcal{L}_{KL}(ST(z_t, \tau), ST(z_s, \tau)) \tag{2.10}$$

where $\mathcal{L}_{KL}(*)$ is the Kullback-Leibler divergence loss.

We can describe response-based knowledge as capturing the “dark knowledge" of the teacher model as the student model is trained to imitate it. However, since only the output of the last layer is used, supervision of intermediate layers is not possible, which can be detrimental to transferring the representations learnt by the teacher. Additionally, response-based knowledge distillation is limited to the supervised setting.

Feature-Based Knowledge

As discussed in Section 2.1, DNNs are great at learning representations of the feature space at increasingly abstract levels. Using this intuition, we can utilise the outputs of both the hidden layers i.e. the feature maps and the output layer as the knowledge to distil into the student model. The goal is to have the same feature activations between the student and teacher models. This is an extension of the aforementioned response-based knowledge that is specifically good for training thinner and deeper networks.

Generally, we formulate the distillation loss for feature-based knowledge transfer as

$$L_{FeaD}(f_t(x), f_s(x)) = \mathcal{L}_F(\Phi_t(f_t(x)), \Phi_s(f_s(x))) \tag{2.11}$$

where $f_t(x)$ and $f_s(x)$ are the feature maps of the hidden layers of teacher and student models, respectively. Additionally, we have denoted the feature map transforms of the teacher and student as Φ_t and Φ_s, respectively. These transforms are to make the feature maps consistent in shape. Finally,$\mathcal{L}_F(.)$ denotes a similarity function used to match the feature maps of the teacher and student models.

However, although feature-based knowledge distillation will provide useful information to the student model, effectively choosing how to choose layers (feature maps) between the teacher (hint layers) and student (guide layers) is a tough problem that requires further investigation. Additionally, due to their potential size differences, effectively matching feature representations of the teacher and student also needs to be explored.

Relation-Based Knowledge

Notice that both response-based and feature-based knowledge use the outputs of specific layers from the teacher model. What relation-based knowledge explores further is the relationship between different layers or data samples.

We can explore the relationship between layers by calculating the correlation between selected feature maps. An example of this would be to calculate the Gram matrix between two layers. It is calculated by using the inner products between features from two layers. Then, using the correlations between feature maps as the distilled knowledge, knowledge distillation via singular value decomposition can be performed to extract key information in the feature maps [20].

More generally, we can formulate the distillation loss of relation-based knowledge as

$$L_{RelD}(f_t, f_s) = \mathcal{L}_{Re}(\psi_t(f_t^{(1)}, f_t^{(2)}), \psi_s(f_s^{(1)}, f_s^{(2)})) \tag{2.12}$$

where f_t and f_s are the teacher and student feature maps, respectively. Pairs of feature maps are chosen from both the teacher model, $f_t^{(1)}$ and $f_t^{(2)}$ and from the student model, $f_s^{(1)}$ and $f_s^{(2)}$. $\psi_t(.)$ and $\psi_s(.)$ are similarity functions for a pair of feature maps for the teacher and student models, respectively. Finally, $\mathcal{L}_{Re}$ denotes a correlation function between the teacher and student feature maps.

2.3.3 Distillation Schemes

The distillation schemes (training schemes) we can use to optimise the objective discussed in Section 2.3.1 can vary according to whether or not we want to simultaneously update the teacher model with the student model or not. It can broadly be categorised into three categories:

Offline Distillation

Offline distillation is a scheme where we have a pre-trained teacher model that does not need to be updated with the student model. This is the simplest scheme to implement and employs a one-way knowledge transfer with two-phase training (teacher training and student training). However, the complex high-capacity teacher model with a long training time cannot be avoided, while the training of the student model under the supervision of the teacher is usually efficient[19].

Online Distillation

In online distillation, both the teacher model and the student model are updated simultaneously, and the whole knowledge distillation framework is end-to-end trainable (one-phase) [19]. An extension of this is co-distillation [21], where the mean model output of all other students is the teacher's output. Many of the relevant works on knowledge distillation in a federated setting (Chapter 3) use a variant of this approach.

Self-distillation

Self-distillation is where the same networks are used for both the teacher model and the student model. One case where this is used is to transfer knowledge from deeper sections of the network to shallower ones [22].

2.3.4 Challenges

Task Limitations

The research surrounding knowledge distillation has been largely focused on classification tasks. There is literature to explore research directions in regression tasks [19].

Model Capacity Gap

The model capacity gap between the large DNN teacher model and small student DNN can degrade knowledge transfer. We can define the model capacity as the complexity of a DNN, the higher the capacity the more complex functions the network can approximate. In the context of knowledge distillation, if the student's capacity is far lower than the teacher then the feature representations learnt by the teacher may not be possible for the student to learn.

How Well Does The Transfer Work?

The ideal goal of knowledge distillation is for the student to match the teacher in terms of predictive distribution. However, doing this in practice surprisingly leads to better generalisation of the student model but a large discrepancy between predictive distributions. In a study on "Does knowledge distillation really work?"[23], it was found that ultimately yes it usually improves the accuracy of the student model. However, matching the teachers output (fidelity) is rarely achieved and far from perfect. The conclusions of the paper were:

- *Good student accuracy does not imply good distillation fidelity:* even outside of self-distillation, the models with the best generalisation do not always achieve the best fidelity.
- *Student fidelity is correlated with calibration when distilling ensembles:* although the highest-fidelity student is not always the most accurate, it is always the best calibrated.
- *Optimisation is challenging in knowledge distillation:* even in cases when the student has sufficient capacity to match the teacher on the transfer set, it is unable to do so.
- *There is a trade-off between optimisation complexity and transfer set quality:* Enlarging the transfer set beyond the teacher training data makes it easier for the student to identify the correct solution, but also makes an already difficult optimisation problem harder.

2.4 Summary

To conclude, the preliminary information needed for the rest of the thesis has been discussed. In particular, an introduction into Deep Learning techniques: ANNs (and by extension DNNs), CNNs, GANs and their optimisation for the supervised learning setting. The optimisation technique discussed was (stochastic) gradient descent and how having an unbiased estimate for the gradient is essential for convergence. After this, the Federated Learning framework for training a model with decentralised data without sharing data between parties. Also, limitations of Federated Learning such as data heterogeneity (non-IID data among clients), a heterogeneous system and personalisation. Then, an introduction of the means of transferring knowledge between DNNs of differing architecture through Knowledge Distillation. It is concluded that there are many ways to interpret knowledge in a DNN and schemes of transferring this knowledge. However, successfully applying Knowledge Distillation has many challenges such as: how to extract the knowledge and whether the transfer was even successful or not.

Chapter 3

Related Work

In this chapter, we review current state-of-the-art methods that tackle the problem of heterogeneous client models in Federated Learning. These methods can be split into two groups: data-based, where clients will either share some private data or need to find a proxy dataset; data-free, where clients have no need to do the former. Also, note that since heterogeneous models are trained, clients will need to maintain state between communication rounds making the mentioned algorithms more suitable for the cross-silo federated setting (Table A.1).

3.1 Data-Based

3.1.1 Heterogeneous Federated Learning via Model Distillation (FedMD)

FedMD [24] was proposed to address the problem of clients independently designing their own model in the federated setting. The authors propose to answer the question: how can one perform federated learning when each participant has a different model that is a black box to others? We can see the algorithm below:

Algorithm 1: FedMD [24]

Input: Public dataset $\mathcal{D}_{public}$, private client datasets $\mathcal{D}_k$, independently designed models C_k, $k = 1..K$,
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 (logits) $l^k = C_k(X^{public})$ on the public dataset, and transmits the results to a central server.
  Aggregate: The server computes an updated consensus, which is an average

$$l = \frac{1}{K}\sum_k l^k$$

  Distribute: Each party downloads the updated consensus l.
  Digest: Each party trains its model C_k to approach the consensus l on the public dataset $\mathcal{D}_{public}$ through knowledge distillation.
  Revisit: Each party trains its model C_k on its own private data for a few epochs.
end

Note that a public dataset $\mathcal{D}_{public}$, is used as the transfer set for knowledge distillation (Section 2.3). They use response-based knowledge (Section 2.3.2) of the local models C_k on $\mathcal{D}_{public}$ and treat the teacher model as an ensemble of the local/student models when performing the distillation. A key limitation in this approach is the requirement for the public dataset $\mathcal{D}_{public}$ as this may be infeasible to construct in practical applications. $\mathcal{D}_{public}$ is constrained to samples from a similar domain with similar features, otherwise transfer learning will be ineffective. In the case where this is not possible, clients will need to share data to construct $\mathcal{D}_{public}$.

3.1.2 Federated Distillation and Federated Augmentation (FD + FAug)

Federated distillation (FD) and Federated Augmentation (FAug) [25] are two methods proposed to solve two problems.

First, in vanilla FL, each client transmits their local model's parameter updates to be aggregated at the central server. If the chosen model is large, there can be millions or even billions of floating point parameters being sent over the network, incurring a large communication cost. FD used online knowledge distillation, in particular co-distillation (Section 2.3.3) to reduce transmitted data to the output of the local model, achieving 26x less communication overhead [25]. Using FD also allows for heterogeneous client models.

Algorithm 2: Federated Distillation(FD)

Input: Client models C_k, Private datasets $\mathcal{D}_k$, Loss function (cross-entropy): $\phi(F; label)$, Ground-truth label: y_{input}

while *not converged* **do**

  procedure *LOCAL TRAINING PHASE (at each client)*

    for *n steps* **do**

      $B, y \leftarrow B, \quad B \subset \mathcal{D}_k$ **for** *sample* $b \in B$ **do**

        `// γ constant learning rate, α distillation weight parameter`

        $\theta_{C_k} \leftarrow \theta_{C_k} - \gamma \nabla [(1-\alpha)\phi(C_k(b|\theta_{C_k}), y_b) + \alpha \cdot \phi(C_k(b|\theta_{C_k}), \hat{l}^{(y_b)})]$

        $l_k^{(y_b)} \leftarrow l_k^{(y_b)} + C_k(b|\theta_{C_k})$

        $cnt_k^{(y_b)} \leftarrow cnt_k^{(y_b)} + 1$

      end

    end

    for *label* $y = 1, 2, ..., c$ **do**

      `// k-th client local average logit vector for y-th ground truth label`

      $\bar{l}_k^{(y)} \leftarrow l_k^{(y)} / cnt_k^{(y)}$

      return $\bar{l}_k^{(y)}$ to server

    end

  procedure *GLOBAL ENSEMBLING PHASE (at central server)*

    for *each client k = 1,2,...,K* **do**

      for *label y = 1,2,...,c* **do**

        `// k-th client global average logit vector for y-th ground truth label`

        $\hat{l}^{(y)} \leftarrow \frac{1}{|K-1|} \sum_{i=1...K}^{i \neq k} \bar{l}_k^{(y)}$

        return $\hat{l}^{(y)}$ to client k

      end

    end

end

Secondly, the authors wanted to approach the problem of non-IID data among clients (Section 2.2.3) by synthetically generating IID data at each client using a GAN (Section 2.1.3). The GAN is trained at a separate, centralised high computing power server with fast internet connection. Each client will then identify the underrepresented samples (target labels) in its local dataset and upload a few *seed* samples to the server to train a conditional GAN. The generator is then downloaded to each client to perform oversampling and generate an IID dataset for training. To additionally keep the target labels private, each client will also upload redundant samples to hide them in plain sight. This client-server privacy leakage has been defined in the paper as:

$$PL_{client} = \frac{|\mathbf{L}_t^{(k)}|}{|\mathbf{L}_t^{(k)}| + |\mathbf{L}_r^{(k)}|} \tag{3.1}$$

$$PL_{inter\_client} = \frac{|\mathbf{L}_t^{(k)}|}{|\bigcup_{j=1}^{M} (\mathbf{L}_t^{(j)} \cup \mathbf{L}_r^{(j)})|} \tag{3.2}$$

where $|\mathbf{L}_t^{(k)}|$ and $|\mathbf{L}_r^{(k)}|$ denotes the number of target and redundant labels, respectively, at the k-th client.

A key drawback to this method is the need for a high computing powered server to train the GAN and the privacy leakage from the GAN itself. For real-world applications, this algorithm is unsuitable as in the case where a GAN needs to be trained, sending the data to a third party may not be an option at all.

3.2 Data-Free

3.2.1 Federated Data-Free Knowledge Distillation via Three-Player Generative Adversarial Networks (FedDTG)

Federated Data-Free Knowledge Distillation via Three-Player Generative Adversarial Networks (FedDTG) [26] (Algorithm 3), is a recent method proposed to solve the issue of sourcing a shared proxy data set (such as that in FedMD (Section 3.1.1) to perform knowledge distillation over.

Algorithm 3: FedDTG

Input: Private client datasets, $\mathcal{D}_k$, independently designed classifiers C_k, global generator G, local generators G_k $k = 1..K$

Output: trained global generator G, trained K local C_k

Initialise model parameters $\theta_G, \theta_D \quad \theta_{C_k}$ for $k = 1..K$

for *each communication round* $t = 1, ..., T$ **do**

$S_t \leftarrow$ random subset of K clients.

for *each client* $k \in S_t$ ***in parallel*** **do**

$\theta_{G_k} \leftarrow \theta_{G_k}, \theta_{D_k} \leftarrow \theta_D$

`// Local Adversarial Training`

for *n local epochs* **do**

calculate GAN losses $\mathcal{L}_{G_k}, \mathcal{L}_{C_k}$ over local data $\mathcal{D}_k \subset \mathcal{D}$

$\theta_{G_k} \leftarrow \theta_{G_k} - \nabla \mathcal{L}_{G_k}$

$\theta_{D_k} \leftarrow \theta_{D_k} - \nabla \mathcal{L}_{D_k}$

$\theta_{C_k} \leftarrow \theta_{C_k} - \nabla \mathcal{L}_{C_k}$

end

return $\theta_{G_k}, \theta_{D_k}$ to Aggregation Server

end

`// Server Aggregation`

$\theta_G = \frac{1}{|S_t|} \sum_{k \in S_t} \theta_{G_k}$

$\theta_D = \frac{1}{|S_t|} \sum_{k \in S_t} \theta_{D_k}$

`// Federated Distillation`

Server generates noise matrix Z^t

for *each client* $k \in S_t$ ***in parallel*** **do**

$\theta_{G_k} \leftarrow \theta_{G_k}, \theta_{D_k} \leftarrow \theta_D$

`// Client executes`

generate label vector $\hat{y} \backsim \mathcal{U}(\infty, \text{num}_\text{c}\text{lasses})$ $X_{KD} \leftarrow G(Z^t, y_{KD}^t | \theta_{G_k})$ generate synthetic inputs.

$l^k \leftarrow C_k(X_{KD} | \theta_{C_k})$ calculate logits

return l^k logits to Aggregation Server

end

$l_{teacher}^k = \frac{1}{|S_t - 1|} \sum_{i \in S_t}^{i \neq k} l^i$ server calculates ensemble teacher logits per client. **for** *each client* $k \in S_t$ ***in parallel*** **do**

calculate losses $\mathcal{L}_{Student}^k$ over $\mathcal{D}_{KD}$, $l_{teacher}^k$

$\theta_{C_k} \leftarrow \theta_{C_k} - \nabla \mathcal{L}_{Student}^k$

end

end

Assuming k clients, in addition to heterogeneous client models (classifiers) C_k, the authors propose training a shared conditional generative network G to model the joint distribution of labelled data between clients, that is, $p_G(X,y) \backsim p_{data}(X,y)$. G can then be used to generate a synthetic proxy dataset for co-distillation (Section 2.3.3) to transfer global knowledge to C_k. G's parameters are shared between k clients that train a local copy G_k on their private data and aggregated after training in a way similar to Federated Averaging [8]. Similarly, there is also a shared discriminator D whose equivalent copy in client k is denoted D_k.

However, unlike regular GANs (Section 2.1.3) that are trained with two networks: the generator and the discriminator, FedDTG includes C_k in the mix. C_k is used to verify the output of the conditional generator so that it produces samples of the class on which the output was conditioned. This is similar to how the popular Auxiliary Classifier GAN [5] trains a conditional generator, although explicitly separates the roles of discriminator and classifier into two separate networks.

Unfortunately, with the benefit of increased privacy comes the cost of additional overhead, both in computation and in communication. The client now has to store and train three models as opposed to one compared to FedMD (Section 3.1.1) and FD (Section 3.1.2) and must communicate two sets of model parameters as well as logits with the server.

3.3 Summary

In summary, we find that response-based knowledge distillation Section 2.3.2) is the technique of choice when trying to transfer knowledge between heterogeneous client models. FedMD requires a proxy dataset; however, there are difficulties in sourcing such a dataset. FD used class-wise logit (soft label) averaging to avoid the need for a common dataset for knowledge distillation but succumbs to the challenge of data heterogeneity section 2.2.3, employing FAug as a way to tackle this, but sacrificing privacy in the process. Finally, FedDTG tackles training heterogeneous client models by training a generative network to synthesise a proxy dataset - the downside to this approach is high overhead. This thesis explores the solution of this problem in a more private and efficient manner than the previously stated methods.

Chapter 4

Federated GAN-Based Data-Free Knowledge Distillation (FedGDKD)

In the previous chapters, we have explored the fundamentals of Federated Learning as well as the motivation for its use. Furthermore, state-of-the-art methods and their limitations to this paper's problem focus have been explored. In this chapter, a novel method for Federated Learning with heterogeneous client models is presented.

4.1 Problem Statement

The problem we are trying to solve addresses the case where clients have limitations or preferences on the architecture of the deep learning model they are training and would also like to keep the details of this model private in addition to their data. These clients could be hospitals with strict regulation on the type of model they can use. This implies that the algorithm must be stateful as each client is keeping track of the latest iterations of their own model. In addition, the heterogeneous client models are learning the same global objective so should have similar final results that outperform training solely on the data they govern.

Therefore, we can further break down the challenges that the proposed method must tackle to the following:

Heterogeneous Models The algorithm should allow each client to select their own personalised model architecture.

Private Each client must not share their private data or details of their model.

Good Performance The performance of each heterogeneous model should perform well on the global objective.

Efficient The algorithm should not incur much additional overhead in terms of communication or computational cost when compared to other state-of-the-art methods.

Note that since heterogeneous models are being trained, each client must keep track of the parameters of their own model throughout training. This means that the algorithm will be stateful which is more suited for the cross-silo federated setting (Table A.1).

4.2 Proposed Method

In this paper, we consider the *supervised* Federated Learning setting: classification. Therefore, the aim is to train a personalised classifier C_k for each client k (K clients) who has a private dataset $\mathcal{D}_k \backsim p_{local_k}$ that will maximise the objective on the global distribution p_{global}:

$$\mathcal{L}_{global} = \mathbb{E}_{X,y \backsim p_{global}}[\log p_{C_k}(y|X)] \quad \forall k = 1,..,K \tag{4.1}$$

Furthermore, we can derive a global dataset $\mathcal{D} = \bigcup_{k=1...K} \mathcal{D}_k$ where $\mathcal{D} \backsim p_{global}$.

To allow heterogeneous models, an alternative approach to parameter aggregation (as used in FedAvg [8]) must be carried out, as this is infeasible in the current setting. Therefore, an alternative means for extracting, aggregating, and transferring global knowledge from client models is required.

We propose *Federated GAN-based Data-Free Knowledge Distillation* (FedGDKD), which uses a conditional GAN to facilitate data-free knowledge distillation that extracts and distributes global knowledge (Algorithm 4). Although similar to FedDTG (Algorithm 3), FedGDKD was developed without prior knowledge of it. The method is divided into three stages: local GAN training, generator aggregation and data-free knowledge distillation.

4.2.1 Local GAN Training

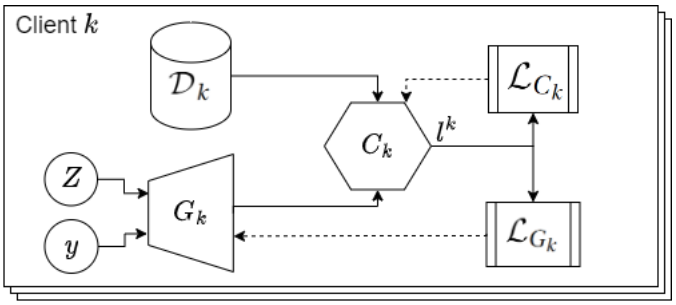


Figure 4.1: FedGDKD: Local GAN Training Stage (dashed line refers to backpropagation)

In this first stage, we implement a local knowledge extraction by taking a novel approach to training a conditional generator G_k for each client k. This conditional generator will learn to approximate the joint distribution of client data $p_{local_k}(X, y)$, where X are inputs and y are classification labels (c classes). We train the generators G_k adversarially directly with the unmodified client classifier C_k.

This is achieved by implementing a modified ACGAN [27]. A normal conditional discriminator D [14] is one that would predict whether the input X, given the label y, is real or fake: $D(X, y) \backsim p_{real}(X|Y)$. The modification made in ACGAN implements the discriminator as a network with two task heads, one discriminator task head D_k, and the other is an auxiliary classifier C_k that shares the same feature extractor as can be seen in Figure 4.2 (**Note** do not confuse D_k with D the figure).

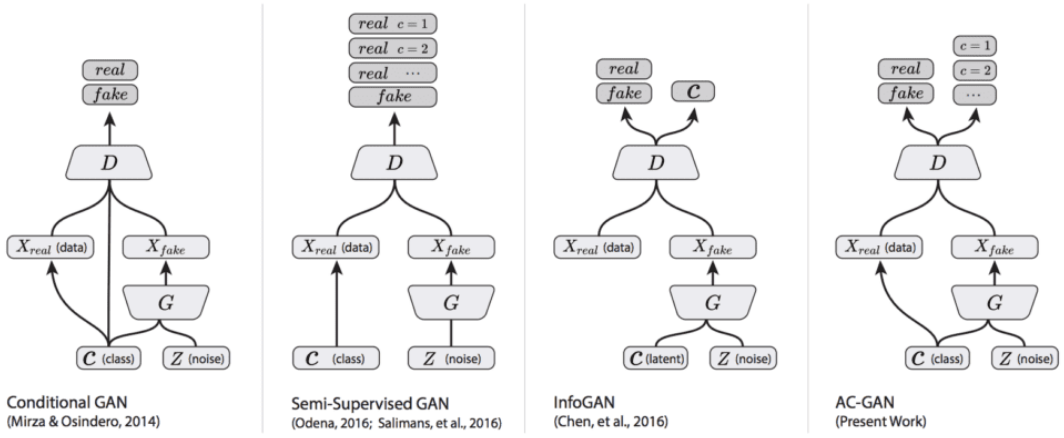


Figure 4.2: Differences between GAN architectures [5]

A modification proposed in [28], originally designed for semi-supervised GANs [29], eliminates the need for a discriminator head and allows just an unmodified classifier C_k. The discriminator output $D_k(X)$ can be reformulated from the as

$$
\begin{aligned}
D_k(X) &= \frac{Z_k(X)}{Z_k(X)+1}, \quad \text{where} \\
Z_k(X) &= \sum_{i=1}^{i=c} \exp[l_i^k(X)]
\end{aligned}
\tag{4.2}
$$

where $C_k(X) = \{l_i^k\}_{i=1..c}$ are the logits outputted by C_k for each class i. The summation of exponents can prove to be numerically unstable so instead we can use the logarithm of this.

$$
\begin{aligned}
logD_k(X) &= logZ_k(X) - log(Z_k(x)+1) \\
log(1-D_k(x)) &= -log(Z_k(x)+1)
\end{aligned}
\tag{4.3}
$$

Note that, $logZ_k(X)$ can be considered a smooth maximum function, referred to as LogSumExp [30] or RealSoftMax which is convex and its gradient is equivalent to SoftMax. Additionally, the second term $log(1-D_k(x))$ is equivalent to a negated Softplus activation function [31] on $logZ_k(X)$ with $\beta = 1$.

Using this alternative formulation for $D_k(X)$ will allow the client **not** to have to modify their personalised classifier to be able to participate in the algorithm, improving ease-of-use. We can then formulate the standard ACGAN objective functions where C_k needs to maximise:

$$
\begin{aligned}
\mathcal{L}_{C_k} &= \mathcal{L}_{GAN}^k + \mathcal{L}_{CLS}^k, \quad \text{where} \\
\mathcal{L}_{CLS}^k &= \mathbb{E}_{X,y\backsim p_{local_k}}[\log p_{C_k}(y|X)] + \mathbb{E}_{z\backsim\mathcal{N}(0,1),y_{gen}\backsim\mathcal{U}(0,c)}[\log p_{C_k}(y_{gen}|G_k(z,y_{gen}))] \\
\mathcal{L}_{GAN}^k &= \mathbb{E}_{X\backsim p_{local_k}(X)}[\log p_{D_k}(real|X)] + \mathbb{E}_{z\backsim\mathcal{N}(0,1),y_{gen}\backsim\mathcal{U}(0,c)}[\log p_{D_k}(fake|G_k(z,y_{gen}))]
\end{aligned}
\tag{4.4}
$$

In eq. (4.4) where there are two components, $\mathcal{L}_{GAN}^k$ constitutes the adversarial loss found in a vanilla GAN [14] which tests the discriminator to determine the correct source of the input (real or fake). The implementation accompanying the original paper [32] suggests passing the discriminator logits $D_k(X)$ through a sigmoid activation to compress the output to the range $[0,1]$ followed by using the binary cross entropy loss. This has been known to cause issues with stability of GAN training due to vanishing gradients [33]. In addition, there is the loss of the classification task $\mathcal{L}_{CLS}^k$, which tests the classifier C_k to predict the correct labels, where the cross-entropy loss is normally used to calculate the negative log-likelihood of the expected class. Instead we use a novel alternative formulation based on this implementation of the semi-supervised GAN [34], which we have modified to train a conditional GAN, that does not require the sigmoid and provides a stronger learning signal using raw logit outputs only. We implement the above losses concretely as a minimisation of (expectation distribution subscript omitted for brevity):

$$
\mathcal{L}_{CLS}^k = -\mathbb{E}[l_y^k(X)] + \mathbb{E}[logZ_k(X)] - \mathbb{E}[l_{y_{gen}}^k(X_{gen})] + \mathbb{E}[logZ_k(X_{gen})] \tag{4.5}
$$

$$
\mathcal{L}_{GAN}^k = -\mathbb{E}[logD_k(X)] + \mathbb{E}[log(1-D_k(X_{gen}))] \tag{4.6}
$$

$$
\text{where } X_{gen} = G_k(z, y_{gen})
$$

We can breakdown the classification loss (eq. (4.5)) into the average error between the logit corresponding to the ground truth label $l_y^k(X)$ and the smooth maximum of the logits $logZ_k(X)$, this error is minimised when the smooth maximum corresponds to the ground truth label. Additionally, $D_k(X)$ output close to 1 (real label) occurs when classifier logits $C_k(X)$ have low entropy i.e. skewed towards one class. Therefore, C_k should optimise to maximise this for real images X and minimise this for fake X_{gen}.

Notice also how the generated inputs and labels are considered in eq. (4.5). This is because as the generator G_k (in conjunction with generator aggregation - Section 4.2.2) approximates the global joint distribution $p_{global}(X,y)$, it can provide synthetic samples of classes that are under-represented or even not present. This should help tackle the issue of heterogeneity (non-iid) of the client's private data $\mathcal{D}_k$.

However, with the modification made to reformulate the discriminator output in eq. (4.2), we can see that there is also a contradiction. The classifier is trying to predict both the correct label and distinguish between real and fake using the same logits. eq. (4.2) suggests that a high realness score ($D_k(X)$ is large) requires that the logits have low entropy i.e. skewed toward one class (after softmax, one class has a high probability). Conversely, for a low realness score the classification logits should have high entropy (not skewed towards a single class). This suggests mutual exclusivity, where the classifier logits can only be used to predict either fake or real and a class - thus only being able to truly optimise one objective

on the synthetic data. This can act as regularisation for the classifier preventing it from dominating the generator in training i.e. always successfully predicting fake so the generator does not learn to improve.

Following this, we have the ACGAN formulation for the generator objective that G_k needs to minimise:

$$\mathcal{L}_{G_k} = \mathbb{E}_{z\sim\mathcal{N}(0,1),y_{gen}\sim\mathcal{U}(0,c)}[\log p_{D_k}(fake|G_k(z,y_{gen})) - \log p_{C_k}(y_{gen}|G_k(z,y_{gen}))] \tag{4.7}$$

In eq. (4.7), the generator needs to create realistic images that will fool the discriminator and produce features in the generated output that will match the conditioned class label y_{gen}. We can also provide the novel concrete formulation that G_k needs to minimise as:

$$\mathcal{L}_{G_k} = -\mathbb{E}[logD_k(X_{gen})] - \mathbb{E}[l^k_{y_{gen}}(X_{gen})] + \mathbb{E}[logZ_k(X_{gen})] \tag{4.8}$$

$$\text{where } X_{gen} = G_k(z, y_{gen})$$

4.2.2 Generator Aggregation

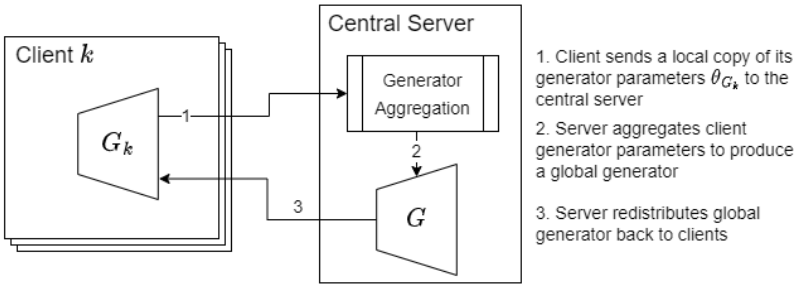


Figure 4.3: FedGDKD: Generator Aggregation Stage

Once the local adversarial training phase is complete, the central server will then average the model parameters of the client generator θ_{G_k}, using a weighted average like FedAvg[8], to produce the global generator G:

$$\theta_G = \sum_{k\in S_t} w_k \theta_{G_k} \quad w_k = \frac{|\mathcal{D}_k|}{\sum_{i\in S_k}|\mathcal{D}_i|} \tag{4.9}$$

where S_t is the set of sampled clients. The distribution of the generator G should approach that of the global distribution, that is, $p_G(X,y) \rightarrow p_{global}(X,y)$.

Unlike existing federated GAN training frameworks [35, 36, 37, 26], this proposed method only aggregates the generator parameters θ_{G_k} as opposed to the generator and discriminator (classifier) parameters θ_{C_k}. This novel approach further improves efficiency and privacy of federated GAN training.

Parameter aggregation is one method of resolving model drift in FL. Model drift occurs because the local objectives L_{C_k} differ from the global objective due to non-iid client data. During local training, the client models will be biased towards their local objective and as such be optimised to move towards a local extrema. The aggregation of parameters followed by the distribution of the global model G acts as a "reset". A simplified example of this can be seen in Figure 4.4, where the black cross and the line represent the change in the global model on the global objective and the white lines represent the client models drifting due to their local objectives. Once the global model parameters have been calculated, these are distributed to the clients again for local training.

An alternative method for training GANs in a distributed manner that allows personalised discriminators has been shown to work [38, 39]. However, usually this requires the generator G to be trained on a high-power central server, querying local discriminators for scores on the generated samples per batch (there can be hundreds or thousands of batches per epoch). This is impractical in the federated setting as this requires a much higher communication cost.

However, the model drift of the local classifier models C_k should also be taken into account. This is done in the next stage.

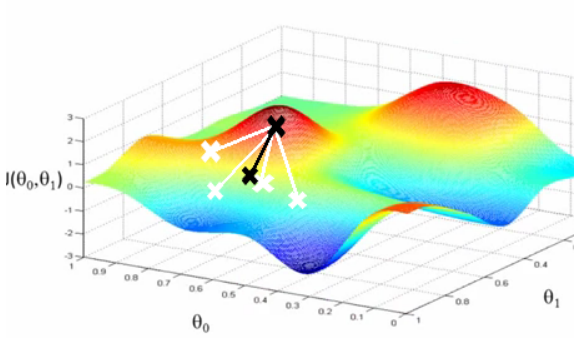


Figure 4.4: Example of model drift

4.2.3 Data-Free Knowledge Distillation

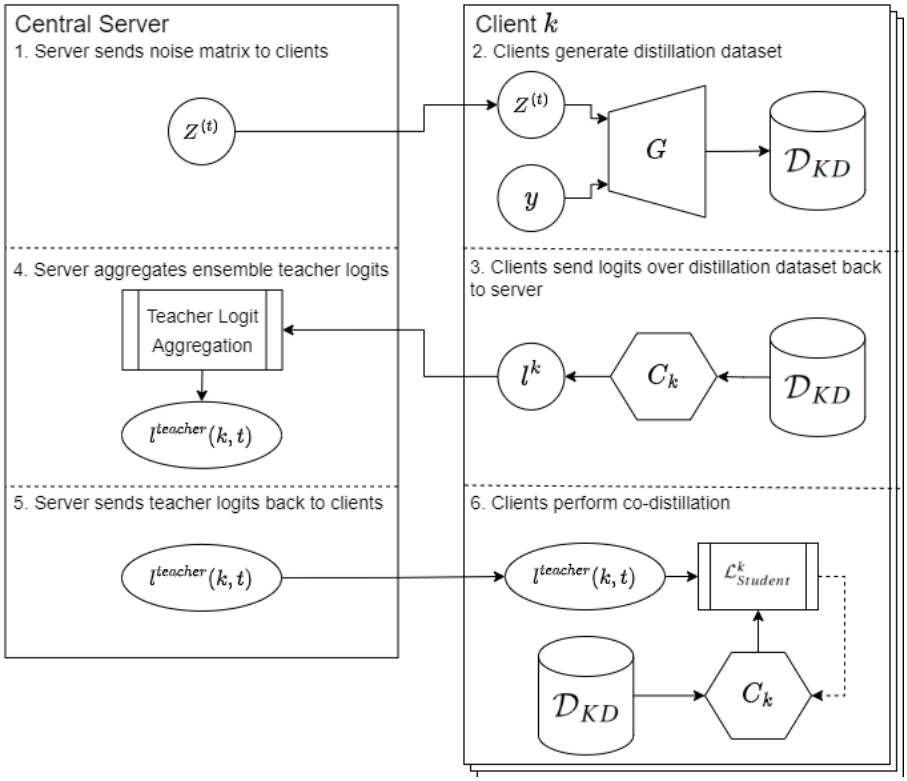


Figure 4.5: FedGDKD: Data-Free Knowledge Distillation Stage (dashed line refers to backpropagation)

In this stage, we correct the drift of the local classifier models C_k and impart global aggregated knowledge to them through co-distillation [21].

To do this, we need to construct a common distillation dataset $\mathcal{D}_{KD}$. Here shines our global generator G. We can generate a synthetic dataset of arbitrary size to act as our *distillation* dataset. To save communication costs, we send to each client a noise matrix $Z \in \mathbb{R}^{\lceil \frac{N}{c} \rceil xnz}$ consisting of $\lceil \frac{N}{c} \rceil$ noise vectors $z \backsim \mathcal{N}(0,1)$ to each client, where N is the target size of the distillation dataset, nz is the latent vector size of G and c the number of classes. The client will then produce $\lceil \frac{N}{c} \rceil$ generated samples X^i_{KD} conditioned on each class i (using label vector $y^i_{KD} \in \mathcal{Z}^{\lceil \frac{N}{c} \rceil}$) together forming a distillation dataset of size N. In addition, we update each client so that they have the same generator parameters θ_G such that identical distillation data sets are produced at each client.

$$\begin{gathered}\mathcal{D}_{KD} = (X_{KD}, y_{KD}), \quad \text{where} \\ X_{KD} = \{G_k(Z, y^i_{KD}|\theta_G)\}^c_{i=0} = \{G(Z, y^i_{KD})\}^c_{i=0} \quad \forall k = 1, .., K \\ y_{KD} = \{y^i_{KD}\}^c_{i=0}\end{gathered} \tag{4.10}$$

Each client classifier C_k will then calculate the logits for the distillation dataset $\mathcal{D}_{KD} \backsim p_G$ and return them to the server. The teacher logits for client k in the communication round t can then be calculated as follows:

$$l^{teacher}(k,t) \triangleq \frac{1}{|S_t - 1|} \sum_{i \in S_t}^{i \neq k} l^i \tag{4.11}$$

The teacher logits contains an ensemble of response-based knowledge (Section 2.3.2) from all other clients that can be used to transfer knowledge about classes that are not present or underepresented in local datasets $\mathcal{D}_k$. As explained in Section 2.3, the student model attempts to mimic the teacher. In this case, the student models are the client models C_k and aim to minimise the following objective:

$$\begin{gathered}\mathcal{L}^k_{student} = (1-\alpha)\,\mathcal{L}^k_{Task} + \alpha\,\mathcal{L}^k_{KD} \quad \alpha \in [0..1], \quad \text{where} \\ \mathcal{L}^k_{Task} = \mathbb{E}_{X_{KD}, y_{KD} \backsim p_G}[p_{C_k}(y_{KD}|X_{KD})] \\ \mathcal{L}^k_{KD} = KL(ST(C_k(X_{KD}), \tau) \,||\, ST(l^{teacher}(k,t), \tau)))\end{gathered} \tag{4.12}$$

where ST is the soft targets function eq. (2.9) with temperature T and KL is the Kullback–Leibler divergence. In this method, the use of knowledge distillation is used to correct the drift of the local model while imparting global knowledge. Therefore, we recommend choosing a high α to prioritise this realignment over correctly classifying the synthetic data.

A consideration that must be made is the case that not all clients will participate in a communication round - a common challenge in FL. Therefore, to account for this, we can perform co-distillation in the subsequent communication round $t+1$ only on those clients who did not participate in the current round, i.e. $k \notin S_t$ where S_t is the sample of clients from the current round. Doing this will "catch up" the clients with knowledge that it has missed out on learning and reduce the model drift, allowing for better performance. The teacher logits would then be the empirical mean:

$$l^{teacher}(k,t) = \frac{1}{|S_t|} \sum_{i \in S_t} l^i, \quad \forall k \in S_{t+1} \setminus S_t \tag{4.13}$$

4.3 Implementation

To implement the proposed method, we chose to use the FL research framework FedML [40]. This is because FedML offers boilerplate code that could be used to speed up development, as well as implementations of existing methods that can be used as baselines to evaluate against. We also opted to use the PyTorch machine learning framework [41] to implement the machine learning aspects of the algorithm.

FedML is an extensive framework that covers many paradigms of FL; however, it lacks refinement and often has complex code with a lot of redundancy. More importantly, it lacked the support for heterogeneous model FL algorithms and stateful clients, so a refactor was required to implement this functionality. The framework opted for an implementation pattern where there is a single client and model instance (since they are homogeneous), and its private data is replaced to emulate different clients.

After we added the functionality to allow for stateful clients, the workflow for implementing such an FL algorithm for single machine simulation is as follows:

4.3.1 `FedML.fedml_api.standalone`

In this package, the FL algorithm code is implemented. To complete an implementation, one needs to implement the following classes:

`StatefulModelBaseTrainerAPI`

This abstract class represents the central aggregation server in the algorithm and also governs the orchestration of the clients. The abstract base class includes implementations that will allow for client sampling, testing, and parameter aggregation, but can also be overriden. The only methods that need to be implemented are the following:

`train()` This method should implement the role of the aggregation server in the algorithm.

`_setup_clients(...)` This private method instantiates multiple stateful clients (Section 4.3.1) given their configuration. This includes individual models as well as their respective datasets.

`BaseClient`

This class represents the individual stateful client that holds the client dataset and a `ModelTrainer` (Section 4.3.1). It is a light layer of abstraction that mainly contains functionality that the aggregation server might call upon. If additional methods are required, this class should be extended. The important methods are the following:

`train(...)` Starts local training at the client.

`ModelTrainer`

Arguably the most important class, this abstract class implements the client-side training of local private models and has the least boilerplate code. This class should be extended to implement the following methods:

`__init__(...)` The constructor of this class should be overridden to allow for any configuration parameters or additional models.

`get_model_params()` If parameter aggregation is used, this method should return the local model parameters.

`set_model_params(...)` This method should update the parameters of a local model. In the case of the proposed method, this would be used to update the local copies of the generator G.

`train(...)` Trains the local models. In the proposed method, this would perform the local adversarial training.

`test(...)` Tests the local models for evaluation.

In addition to the above, for the proposed algorithm, we have implemented additional functionality to carry out the knowledge distillation.

4.3.2 `FedML.fedml_experiments.standalone`

In this package are the scripts for the experiments. Here, we can use the APIs defined previously (Section 4.3.1) in a parameterised experiment. To do so, one needs to extend the following class:

`ExperimentBase`

This abstract class contains the basic functionality to run an experiment. In particular, it will set up the environment by parsing command-line arguments (of which there are common ones), partitioning the selected dataset, and logging the experiment run to Weights and Biases[42] - an MLOps platform. One needs to implement the following methods:

`add_custom_args(...)` This method allows FL algorithm-specific arguments to be added to the command-line argument parser.

`experiment_start(...)` Instantiates the custom `StatefulModelBaseTrainerAPI` (Section 4.3.1) class and calls the `train` method.

4.4 Summary

In this chapter, we have introduced the novel FedGDKD algorithm which tackles the problem of heterogenous client model federated learning with data-free knowledge distillation. To facilitate the “data-free" aspect, a novel federated conditional GAN (cGAN) is trained, to generate realistic synthetic data that can be used as a common dataset for knowledge distillation. To further ease use of this method, we implement novel cGAN training objectives, to allow for unmodified client models to take part.

Algorithm 4: FedGDKD

Input: Private client datasets, $\mathcal{D}_k$, independently designed classifiers C_k, global generator G, local generators G_k $k = 1..K$

Output: trained global generator G, trained K local C_k

Initialise model parameters θ_G, θ_{C_k} for $k = 1..K$

for *each communication round* $t = 1, ..., T$ **do**

$S_t \leftarrow$ random subset of K clients.

for *each client* $k \in S_t$ ***in parallel*** **do**

send θ_G to client k

$\theta_{G_k} \leftarrow \theta_G$

`// Catch up Co-Distillation`

if *client* $k \notin S_{t-1}$ *and* $t > 1$ **then**

Send $l^{teacher}(k, t-1)$ eq. (4.13), $Z^{(t-1)}$ to client k

$\mathcal{D}_{KD}^{(t-1)} \leftarrow (X_{KD}^{(t-1)}, y_{KD}^{(t-1)})$ generate distillation dataset eq. (4.10)

`Co-distillation`$(\mathcal{D}_{KD}^{(t-1)}, l^{teacher}(k, t-1))$

end

`// Local GAN Training`

for *n local epochs* **do**

calculate losses $\mathcal{L}_{G_k}$ eq. (4.7), $\mathcal{L}_{C_k}$ eq. (4.4) over local data $\mathcal{D}_k \subset \mathcal{D}$

$\theta_{G_k} \leftarrow \theta_{G_k} - \nabla\mathcal{L}_{G_k}$

$\theta_{C_k} \leftarrow \theta_{C_k} - \nabla\mathcal{L}_{C_k}$

end

return θ_{G_k} to Aggregation Server

end

`// Generator Aggregation`

$\theta_G = \sum_{k \in S_t} w_k \theta_{G_k} \quad w_k = \frac{|\mathcal{D}_k|}{\sum_{j \in S_t} |\mathcal{D}_j|}$

`// Data-Free Knowledge Distillation`

Server generates noise matrix $Z^{(t)} \in \mathbb{R}^{\lceil \frac{N}{c} \rceil \times nz}$

for *each client* $k \in S_t$ ***in parallel*** **do**

Send $Z^{(t)}$, θ_G to client k

`// Client executes`

$\theta_G^k \leftarrow \theta_G$

$\mathcal{D}_{KD}^{(t)} \leftarrow (X_{KD}^{(t)}, y_{KD}^{(t)})$ generate distillation dataset eq. (4.10)

$l^k \leftarrow C_k(X_{KD}^{(t)} | \theta_{C_k})$ calculate logits

return l^k logits to Aggregation Server

end

for *each client* $k \in S_t$ ***in parallel*** **do**

Send $l^{teacher}(k, t)$ (eq. (4.11)) to client k

`Co-distillation`$(\mathcal{D}_{KD}^{(t)}, l^{teacher}(k, t))$

end

end

Function `Co-distillation`$(\mathcal{D}_{KD}, l_{teacher})$ **is**

for *d distillation epochs* **do**

calculate losses $\mathcal{L}_{Student}^k$ eq. (4.12) over $\mathcal{D}_{KD}$, $l_{teacher}$

$\theta_{C_k} \leftarrow \theta_{C_k} - \nabla\mathcal{L}_{Student}^k$

end

end

Chapter 5

Evaluation

In this section, we evaluate the proposed **FedGDKD** (Chapter 4) against our problem statement (Section 4.1). This is done by testing various scenarios to gauge performance, efficiency, privacy guarantees, and justify the stages of the proposed method.

5.1 Experiment Setup

5.1.1 Dataset

To evaluate the proposed method and compare it to state-of-the-art methods, we used two image classification datasets: MNIST[43] and EMNIST[44]. MNIST is a dataset composed of 28x28 greyscale images of handwritten digits. It has a training set size per digit (10 classes) of 6,000 and a testing set size of 1,000. EMNIST is a dataset composed of 28x28 greyscale images of handwritten digits and letters (0-9, a-z, A-Z) totalling 47 classes (balanced split) with a training set, testing set size per class of 2,400 and 400 respectively. There are 47 classes as they merge similar looking classes into one to avoid misclassification.

5.1.2 Configurations

Unless otherwise specified, we focus on 10 clients (10 models) with hyperparameters shared between algorithms remaining fixed. This was done without tuning for any particular scenario. Additionally, we fix random seeds (Table B.1) to provide determinism and isolate performance based on the change in algorithm. The main shared hyperparameters are:

Hyperparameter	Value
Communication Rounds (T)	50
Client Local Training Epochs (n)	5
Client Model (C_k) Optimiser	SGD(learning rate = 0.01)
Batch Size	32
Image Size (Resize Transform)	32
Knowledge Distillation Epochs (d) (if applicable)	5
Knowledge Distillation Weight (α_{KD}) (if applicable)	0.8
Soft Targets Temperature (τ) (if applicable)	4
Generator Architecture (if applicable)	Specified in Appendix B.3.1
Generator Optimiser (if applicable)	Adam(learning rate = 0.001)
Generator Latent Vector Dimensions (nz) (if applicable)	100
Discriminator Architecture (if applicable)	Specified in appendix Appendix B.3.2
Discriminator Optimiser (if applicable)	SGD(learning rate = 0.01)
Distillation Dataset Size (N) (if applicable)	10000

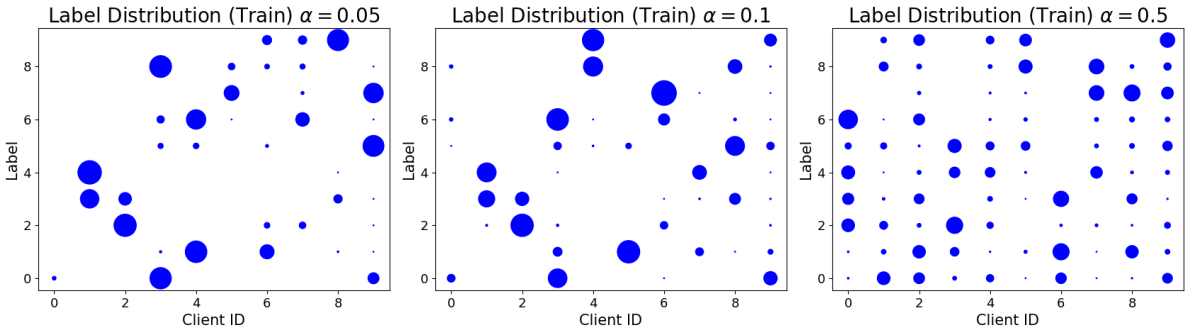
Table 5.1: Shared Hyperparameters

To represent the lack of data in a federated setting, we use at most $r = 25\%$ of the training dataset and distribute it in a non-iid fashion to clients (Section 5.1.3). As the goal of our proposed algorithm

FedGDKD is to optimise local models towards the global objective, we use an iid distribution of the test set at each client.

5.1.3 Client Data Heterogeneity

To achieve a non-iid client data distribution, we use a Dirichlet distribution **Dir**(α) [45], in which a smaller α indicates higher data heterogeneity. See Figure 5.1.



(a) Dir($\alpha = 0.05$) (b) Dir($\alpha = 0.1$) (c) Dir($\alpha = 0.5$)

Figure 5.1: MNIST ($r = 10\%$) client training label distribution with varying degree of heterogeneity

The size of the point (blue) in Figure 5.1 represents the frequency of the label/class in the training dataset or private data of each client.

5.1.4 Baselines

The goal of **FedGDKD** is to allow effective training of heterogeneous models in a federated setting without the need for a proxy/shared dataset. Therefore, we consider **Baseline**, that is, each client model is trained on their private data, to determine that the algorithm indeed imparts global knowledge to local models (a lower bound); **Centralised**, where each client model is trained on the combined client datasets (if all data were to be aggregated in one data centre), this is an upper bound of performance; **FedAvg** [8], the state-of-the-art and most widely used federated learning (FL) algorithm, we compare the performance with homogeneous client models; **FedMD** (Section 3.1.1), **FD + FAug** (Section 3.1.2) and **FedDTG** (Section 3.2.1) state-of-the-art FL algorithms that allow heterogeneous client models.

Note that for both **FedMD** and **FD + FAug**, each client creates a public and augmented data set, respectively, sharing a random 5% of their private data. This is based on the fact that in **FedMD**, the proxy dataset can be difficult to find (in the same domain); in **FD + FAug**, the generator is trained on this shared data (for 200 epochs) and distributed to each client prior to the commencement of communication rounds. Additionally, we introduce another baseline: **Baseline (5% shared)**, that is similar to **Baseline** but augments each client's private data with the shared dataset detailed earlier. This baseline is to see the additional benefit of **FedMD** and **FD + FAug** in comparison to just sharing the data.

5.2 Performance

To properly evaluate the performance of the proposed method **FedGDKD**, we evaluated it in various scenarios.

5.2.1 Homogeneous Client Models

The first scenario in which we evaluate performance is when the client models are identical $\forall k, j.C_k = C_j$. For this experiment, the architecture of the homogeneous client model has a convolutional neural network feature extractor with three convolutional layers of filter size 8, 16 and 16 with an instance normalisation layer and ReLU activation between each one. Then the classification task head is two fully connected linear layers with feature sizes 128 and the number of classes as output, respectively.

This is the simplest circumstance under which to perform an evaluation, as we can then compare it to the **FedAvg** [8] the most popular federated learning algorithm. Doing so will allow us to measure basic proficiency.

Since the goal is to train the client models C_k to perform well in the global objective, we can measure this with the average top-1 accuracy across clients in an iid test set for each client (balanced in all classes). We have collected the top-1 test accuracy across both the MNIST and EMNIST datasets while varying the degree of heterogeneity of the client data ($Dir(\alpha)$) with values $\alpha = [0.05, 0.1, 0.5]$. The smaller α the more heterogeneous the client training distributions will be. Furthermore, we vary the proportion of the training dataset used: sampling ratio $r = [10\%, 25\%]$ (uniform random selection). This is to test the robustness under varying amounts of training data. The results we gather using the parameters defined in Section 5.1.2 are in the Table 5.2 and you can see the effect of the dataset setting in Figure B.1.

Average Top-1 Test Accuracy (%)									
Dataset	Setting	Baseline	Baseline (5% shared)	Centralised	FedAvg	FD + FAug	FedMD	FedDTG	FedGDKD
MNIST $r = 25\%$	$\alpha = 0.5$	64.0	93.5	97.3	**93.0**	66.6	92.9	63.6	91.7
	$\alpha = 0.1$	38.1	92.5	96.9	85.2	47.6	**90.1**	38.9	72.0
	$\alpha = 0.05$	21.6	91.3	97.2	56.6	52.0	**81.4**	22.0	51.1
MNIST $r = 10\%$	$\alpha = 0.5$	58.6	89.5	96.6	**89.8**	53.8	89.7	59.4	88.5
	$\alpha = 0.1$	27.4	87.0	96.6	67.2	37.9	**85.0**	28.4	57.6
	$\alpha = 0.05$	27.9	85.8	96.1	65.7	41.1	**87.0**	28.4	61.2
EMNIST $r = 25\%$	$\alpha = 0.5$	50.4	68.7	78.3	**71.1**	49.1	69.7	46.9	69.7
	$\alpha = 0.1$	28.6	66.2	70.5	52.9	28.6	54.7	26.8	**54.8**
	$\alpha = 0.05$	22.6	65.0	65.0	37.9	27.8	**41.1**	17.4	38.7
EMNIST $r = 10\%$	$\alpha = 0.5$	42.5	60.2	75.8	**68.9**	39.6	63.2	37.4	67.1
	$\alpha = 0.1$	24.1	55.5	76.5	50.5	16.0	47.7	21.0	**53.5**
	$\alpha = 0.05$	21.4	52.1	72.3	42.8	21.2	42.7	18.9	**45.1**

Table 5.2: Summary of top-1 test accuracies over varying settings.

In Table 5.2, we see an interesting result. First there is a general trend that as data heterogeneity increases (α decreases), the final average top-1 test accuracy decreases. Likewise, as the dataset sampling ratio r decreases so does the performance. Both of which are expected, lower α means that more clients will lack good representation of many classes, simiilarly with lower r again the number of samples per class is reduced meaning the local model is likely to not generalise well.

Second, in cases of low data heterogeneity (high α), we see that FedAvg performs the best. This is likely due to directly being able to aggregate client model parameters. However, it seems to lack robustness to increased heterogeneity (lower α) as it tends to perform worse than either FedMD or the proposed method FedGDKD under these scenarios. FedMD, consistently performs well on the MNIST dataset but not as well on the EMNIST dataset, performing worse than Baseline (5% shared). This is likely due to an increased complexity in the task, as EMNIST has 47 classes as opposed to MNIST's 10. Therefore, the proxy dataset that is made up of 5% of client private data may not have enough representation of all classes to perform effectively. Finally, we see that our proposed method FedGDKD also performs reasonably well. Its performance is similar to FedAvg, although generally worse. However, in the more complex EMNIST dataset, it is more robust to higher data heterogeneity than FedAvg.

However, we should also consider the perspective that since in the cases of utilising algorithms FedMD and FD + FAug, we assume that clients are willing to share 5% of their private data. Therefore, to show the additional benefit each algorithm has, we should also consider the improvement over their respective baselines. For FedMD and FD + FAug - Baseline (5% shared); for FedAvg, FedDTG and FedGDKD - Baseline. We present the improvements over these baselines in Table 5.3.

Average Top-1 Test Accuracy Improvement (%)						
Dataset	Setting	FedAvg	FD + FAug	FedMD	FedDTG	FedGDKD
MNIST $r = 25\%$	$\alpha = 0.5$	**29.0**	-26.9	-0.58	-0.44	27.7
	$\alpha = 0.1$	**47.1**	-45.0	-2.45	0.832	34.0
	$\alpha = 0.05$	**35.0**	-39.3	-9.89	0.321	29.5
MNIST $r = 10\%$	$\alpha = 0.5$	**31.1**	-35.6	0.183	0.808	29.9
	$\alpha = 0.1$	**39.9**	-49.1	-1.94	1.04	30.3
	$\alpha = 0.05$	**37.8**	-37.3	-23.9	-5.28	33.3
EMNIST $r = 25\%$	$\alpha = 0.5$	**20.7**	-19.6	1.02	-3.54	19.3
	$\alpha = 0.1$	24.3	-37.6	-11.5	-1.81	**26.2**
	$\alpha = 0.05$	15.2	-37.3	-23.9	-5.28	**16.1**
EMNIST $r = 10\%$	$\alpha = 0.5$	**26.4**	-20.7	2.98	-5.12	24.7
	$\alpha = 0.1$	26.4	-39.6	-7.84	-3.11	**29.4**
	$\alpha = 0.05$	21.4	31.0	-9.40	-3.07	**23.7**

Table 5.3: Summary of top-1 test accuracy improvement over respective baselines.

We can see in Table 5.3 that this clarifies the additional benefit of each algorithm. We see that similar to before FedAvg consistently has the most improvement over the baseline. However, now FedGDKD proves to provide the second most benefit. Note that both data-based methods FedMD and FD + FAug actually have little benefit over simply sharing the 5% of private data, even proving to be worse than this. In the case of FedMD, this is likely due to the regularisation of knowledge distillation adding too much bias and worsening performance. This could be similarly said for FD + FAug, but is more likely due to a bad generative model trained on a small amount of data, causing the client models to fit to this unrepresentative synthetic samples. Finally, FedDTG does not provide much benefit over the baseline, this was due to the complex three-player adversarial training that seems not be working effectively in this experimental setup.

In summary, we have shown that the proposed method FedGDKD provides the most benefit to performance compared to state-of-the-art heterogeneous model federated learning algorithms. Additionally, it even shows potential for higher robustness to increased data heterogeneity for more complex tasks than FedAvg but this needs to be explored further.

5.2.2 Heterogeneous Client Models

Previously, we have tested the case of homogeneous client models (Section 5.2.1), now we look at the primary scenario of this thesis: heterogeneous client models. In this study, we examine the performance of heterogeneous model federated learning algorithms and their ability to transfer global knowledge.

In particular, we investigate the performance of 10 heterogeneous clients which have different client model architectures. Each client model's feature extractor is made up of blocks of a 2D convolution layer, 2D instance normalisation layer and ReLU activation. The number and size of the filter of these blocks differ. The classification task head is made up of two dense linear layers whose penultimate layer has 128 neurons. However, this is not a restriction, just for easier parameterisation. In reality the architecture of these models only need to have the same output layer. We summarise the architectures used in the following table:

Client Model Architectures	
Client Index	Feature Extractor Block Configuration
0	[16, 32]
1	[16, 32, 16]
2	[8, 16, 16]
3	[8, 8, 8]
4	[32, 64, 64]
5	[32, 32, 32]
6	[16, 16]
7	[32, 32]
8	[16, 16, 16, 16]
9	[16, 32, 64, 32]

Table 5.4: Summary of heterogeneous client model architectures.

where “Feature Extractor Block Configuration" refers to the configuration of the feature extractor blocks where the length of the list is the number of blocks and each item in the list is the number of convolutional filters.

We then fix the setting under which to run the algorithms to MNIST with non-iid parameter $Dir(\alpha = 0.5)$ and training set sample ratio of $r = 25\%$ (Figure 5.2). A summary of the results per client can be seen below:

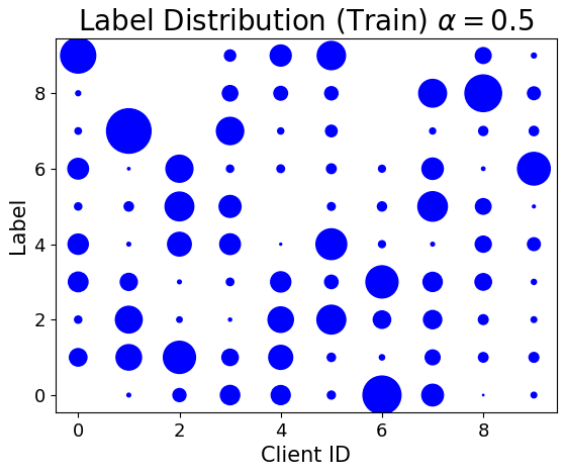


Figure 5.2: Heterogeneous Models: Client training label distribution (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

Client Index	Top-1 Test Accuracy (%)						
	Baseline	Baseline (5% shared)	Centralised	FD + FAug	FedMD	FedDTG	FedGDKD
0	73.5	92.3	97.8	77.1	**95.0**	87.9	92.2
1	59.4	90.6	96.3	65.2	**91.3**	79.9	90.2
2	58.0	91.5	96.8	66.1	**91.8**	84.5	90.6
3	80.9	90.8	97.2	78.8	**93.7**	85.4	88.8
4	72.9	93.6	97.9	76.4	**94.2**	85.8	91.7
5	94.4	97.0	98.8	92.3	**96.7**	93.4	96.5
6	63.9	94.4	98.2	72.5	**96.2**	88.1	94.6
7	83.2	98.2	100	86.0	**98.3**	95.8	97.1
8	77.7	96.2	98.7	83.5	**96.5**	93.4	96.1
9	70.6	93.6	98.5	77.6	**94.6**	84.9	91.6

Table 5.5: Heterogeneous Client Models: Top-1 test accuracies (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

In Table 5.5, we can see that FedMD consistently performs best for each client and FedGDKD consistently performs second best. However, we must also consider that FedMD assembles its proxy dataset

from 5% of shared private data. Therefore let us consider the improvements from the corresponding baselines. Algorithms that use client data: FedMD and FD + FAug will be compared to the Baseline (5% shared) performance and those that do not: FedDTG and FedGDKD will be compared to Baseline. The improvements over respective baselines is presented in Table 5.6.

Client Index	Top-1 Test Accuracy Improvement (%)			
	FD + FAug	FedMD	FedDTG	FedGDKD
0	-15.2	2.71	14.4	**18.7**
1	-25.4	0.71	20.5	**30.8**
2	-25.4	0.30	26.5	**32.5**
3	-11.9	2.91	4.52	**7.83**
4	-17.1	0.610	13.0	**18.8**
5	-4.72	-0.450	-1.01	**3.12**
6	-21.9	1.80	24.2	**30.7**
7	-12.2	0.102	12.6	**13.9**
8	-12.7	0.310	15.7	**18.4**
9	-16.0	1.01	14.3	**30.0**

Table 5.6: Heterogeneous Client Models: Top-1 test accuracy improvement over baselines (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

As can clearly be seen in Table 5.6, FedGDKD improves the most upon its baseline consistently. This suggests that it is the most effective at global knowledge extraction and distribution. It is interesting to note a consistent decrease in performance from FD + FAug (Section 3.1.2), this is due to a bad generator that is only trained on a small subset of the training set. Similar reasoning can be given for FedDTG, as due to the complexity of its three-player GAN, under these circumstances its generator also performs worse. This is not the case in FedGDKD, as the novel GAN training proves to be more stable and produces higher quality synthetic data. We recorded the Frechet Inception Distance (FID) [46] at corresponding stages, which is a metric for measuring the distance between the feature distributions of two image samples. Therefore, the lower score the better. We found that the generators at the end of their training had scores of: 315 for FD + FAug; 188 for FedDTG and 39.4 for FedGDKD. Further confirming this theory.

In summary, we have shown that FedGDKD is an effective heterogeneous model federated learning algorithm and should be selected out of the presented algorithms clients are not willing to share data. However, if the client is willing to share some data, then FedMD is the best choice.

5.2.3 Active-User Ratio

A common challenge in FL is that a client may not participate in all communication rounds. This could be due to not meeting requirements, e.g. connected to power and the Internet, or technical difficulties such as the Internet going down. Therefore, FL algorithms should be robust to such scenarios and still perform well.

In this experiment, we adjust the active-user ratio so that we can emulate such a scenario. We chose active-user ratios of: 30%, 50% and 100% (full participation) of 10 homogeneous model clients to see the effects. This was carried out on the MNIST dataset with non-iid parameter $Dir(\alpha = 0.5)$ and the training set sample ratio $r = 25\%$. The client sampling per round was performed randomly from a uniform distribution, so the likelihood of being selected per client matches that of the active-user ratio (see Figure 5.3 and Figure B.2).

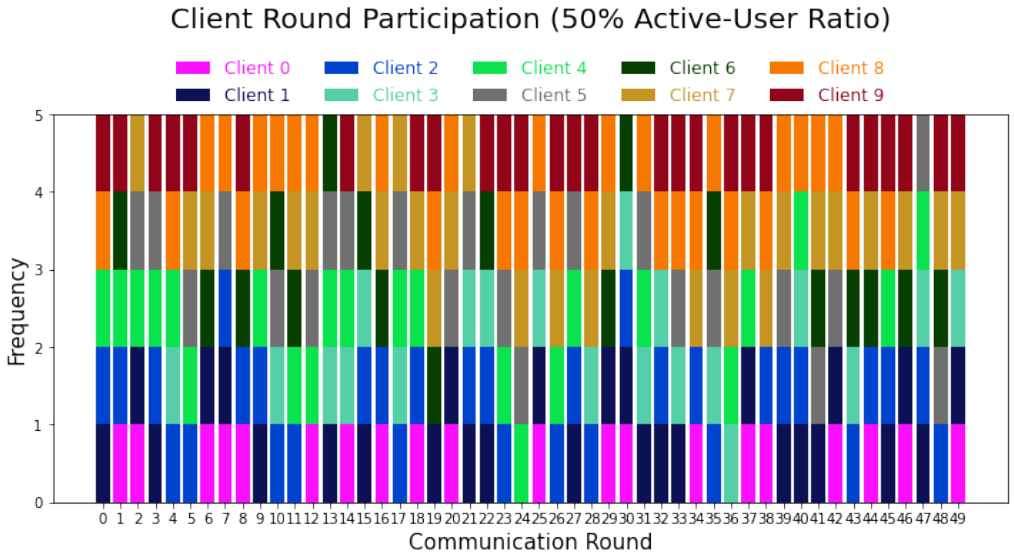
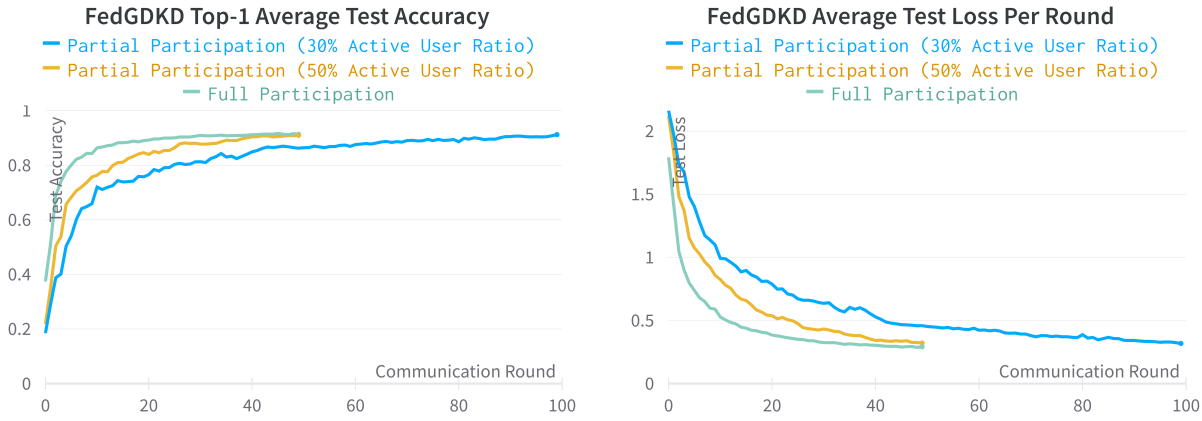


Figure 5.3: Client participation per round (active-user ratio 50%)



(a) Top-1 Average Test Accuracy

(b) Average Test Loss

Figure 5.4: Test curves when varying the active-user ratio (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

Figure 5.4 depicts the average test performance (mean) of client models given varying active-user ratios. We can see that the final performance of the local models is not affected by user sampling. This can be seen in the final test accuracy of each run (~91% test accuracy).

However, the number of communication rounds required to achieve the said performance increases as the active-user ratio decreases. This can be seen where the different runs reach within 1% of the final top-1 average test accuracy:

Active-User Ratio	Communication Rounds
30%	88
50%	39
100% (full participation)	25

Table 5.7: Communication rounds required to reach within 1% of final average top-1 test accuracy

we can see in Table 5.7 a super-linear decay, but, to be sure of this, further active-user ratios should be considered. In relation to this observation, we can also see in Figure 5.4, that the curves are more unstable. Both observations are likely due to the clients not participating in the communication round,

who will not improve, performing worse than those participating. We can see this in Figure 5.5, where the flat sections are, where the client is not participating in the rounds. This results in a negative skew in average performance. The lower the active-user ratio, the greater the skew. This is, of course, unavoidable and expected.

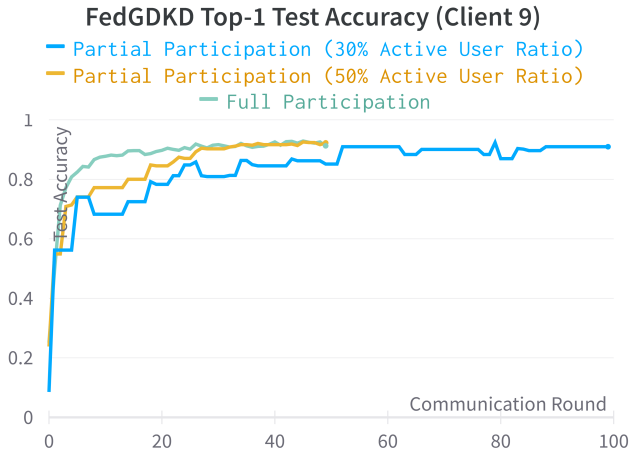


Figure 5.5: Client 9 Top-1 Test Accuracy

The robustness to this scenario is most likely attributed to the “catch up" distillation described in Section 4.2.3, where if a client has not participated in the previous round, they must undergo knowledge distillation on the distillation dataset constructed in the previous round. This imparts knowledge that would otherwise be missing from the client model and effectively "catches" the model up to be suitable for the current round. Further investigation of this is done in Section 5.3.3. In general, we can see that the proposed method FedGDKD is robust to scenarios where clients may not participate in every round having little effect on final performance given enough communication rounds.

5.2.4 Knowledge Distillation Parameter Study

FedGDKD employs knowledge distillation (KD), in particular co-distillation, as a means of aggregating and transferring knowledge between heterogeneous client models. This feature is extremely important for the success of the algorithm, as has been discussed in Section 5.3.2. Indeed, there are three hyperparameters that dictate the impact of co-distillation in FedGDKD, namely: the knowledge distillation weight parameter α_{KD}, the distillation epochs d, and the size of the distillation dataset N. Therefore, in this study, we examine the effects of changing these parameters on the performance of the algorithm.

To do so, we pick a problem setting in which there can be a clear improvement in performance and fix this so that only the change in parameter will affect the change in performance. We choose MNIST with a non-iid setting of $Dir(\alpha = 0.1)$ and dataset sampling ratio of $r = 10\%$. Unless specified, for each study the default values for these hyperparameters are noted in Table 5.8 and the corresponding client training label distribution can be seen in Figure 5.6.

Hyperparameter	Value
Knowledge Distillation Weight α_{KD}	0.8
Knowledge Distillation Epochs d	5
Distillation Dataset Size N	10000

Table 5.8: Knowledge Distillation Parameter Study: Default hyperparameters.

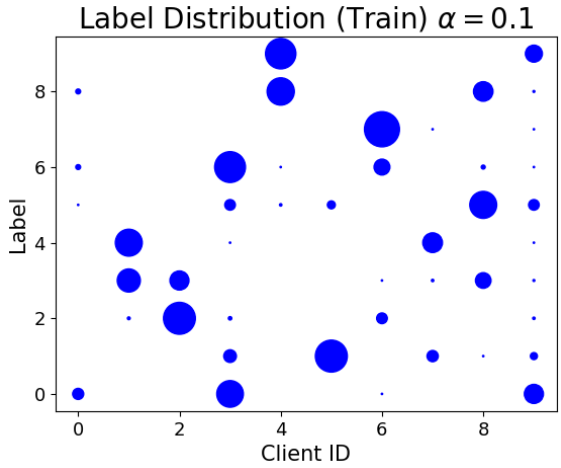


Figure 5.6: Client Training Label Distribution (MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$)

Knowledge Distillation Weight Parameter α_{KD}

First, let us consider studying the effects of the KD weight parameter α_{KD}. This parameter controls the importance of the KD regulariser in the student objective Equation (4.12), where a higher value indicates prioritising alignment of student and teacher outputs, and a lower value prioritising classification performance on the distillation dataset. Given that the distillation data set is synthetic and produced by the learnt generator, a low α_{KD} could lead to worse performance due to low quality samples. We can then test this hypothesis by varying the values of $\alpha_{KD} = [0.1, 0.5, 0.8, 1]$.

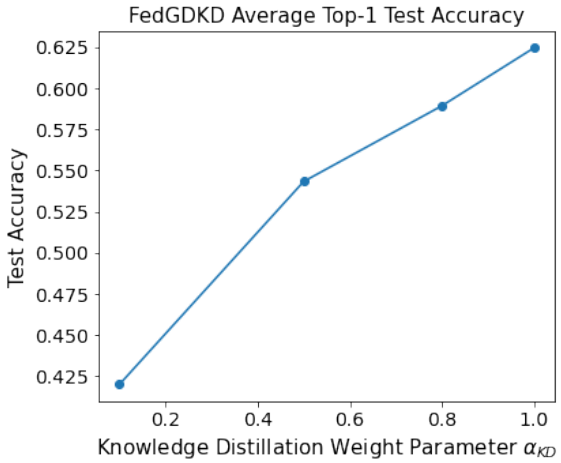
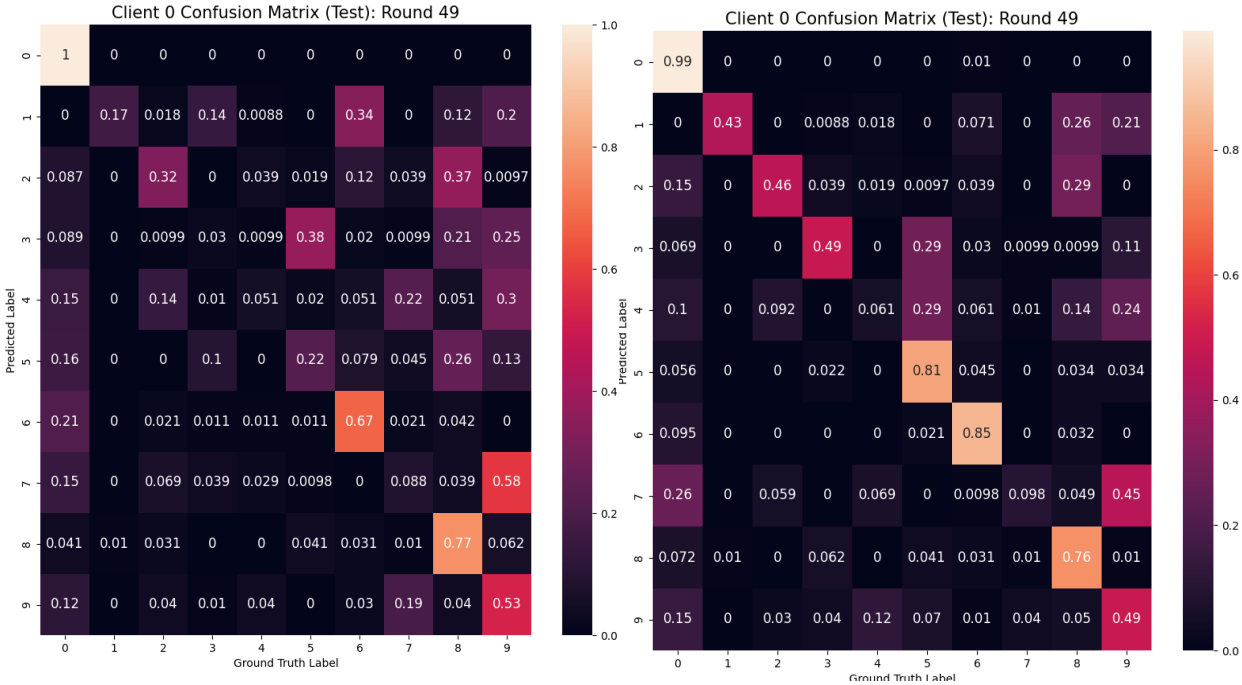


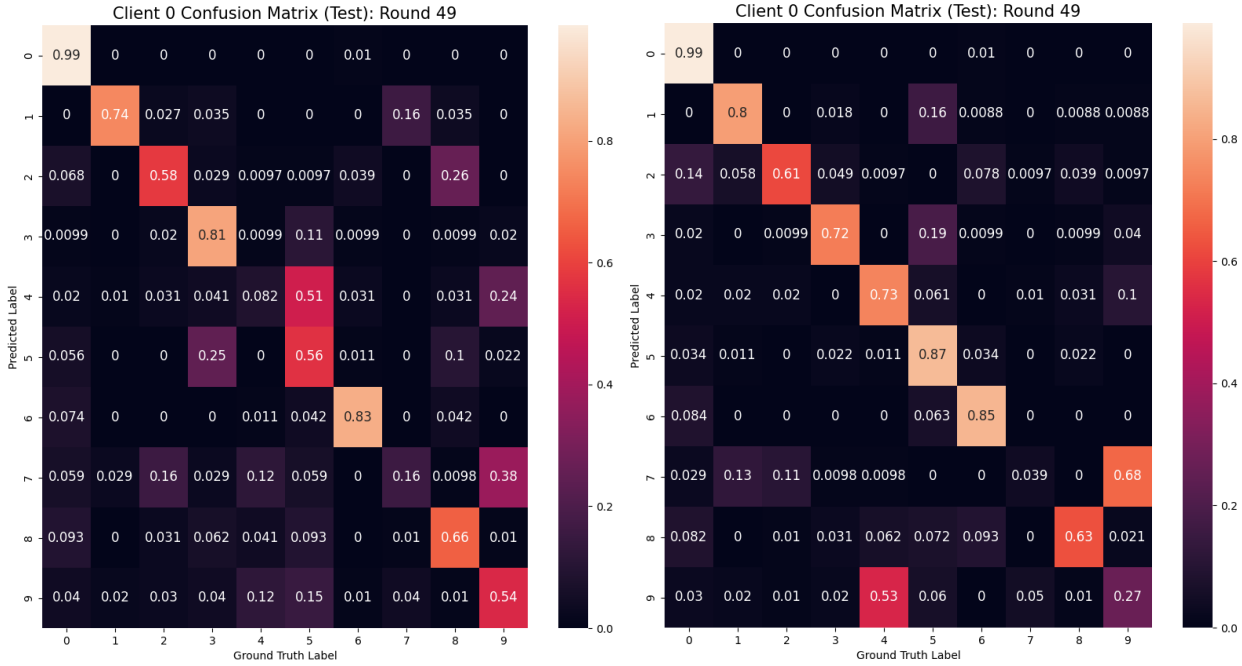
Figure 5.7: Average top-1 accuracy with varying α_{KD}

In Figure 5.7, there is an almost linear improvement in the top-1 average test accuracy as α_{KD} increases. This is likely due to the strength of the learning signal from the teacher logits increasing with α_{KD} in the student's objective Equation (4.12). This increase in learning signal, causes the local model to align more with the ensemble teacher logits and further ignoring classifying the potentially low quality synthetic data correctly. This allows more global knowledge to be transferred to the otherwise isolated local model.



(a) $\alpha_{KD} = 0.1$

(b) $\alpha_{KD} = 0.5$



(c) $\alpha_{KD} = 0.8$

(d) $\alpha_{KD} = 1$

Figure 5.8: KD Parameter Study: Client 0 confusion matrices (x-axis: Ground Truth Label; y-axis: Predicted Label) after training when varying α_{KD} (MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$).

We can confirm that this is the case by looking at the client with index 0 (Client 0) whose training label distribution is very sparse (Figure 5.6). In particular, we look at the final confusion matrix after training on the test data Figure 5.8. A perfect classification model will have high values across the diagonal (individual class precision). We can see that as α_{KD} increases, the precision per class increases. In summary, we have confirmed that a high α_{KD} results in better performance due to prioritising alignment with teacher logits.

Knowledge Distillation Epochs d

Next, we investigate the effect of varying the number of knowledge distillation (KD) epochs d. This hyperparameter has the role of determining to what degree the student's objective Equation (4.12) is optimised. The degree refers to how close to the optimal value can be reached. This is because gradient-based optimisation (in particular, stochastic gradient descent Section 2.1.5) is used. At each epoch, the gradient is approximated and then a step is taken towards a local optima. The more steps that can be taken, the closer the local optima that can be reached (assuming that divergence does not occur). Therefore, we hypothesise that increasing d will allow further global (teacher) knowledge to be transferred to the local model. We can test this by looking at the performance while varying $d = [1, 5, 10, 20]$

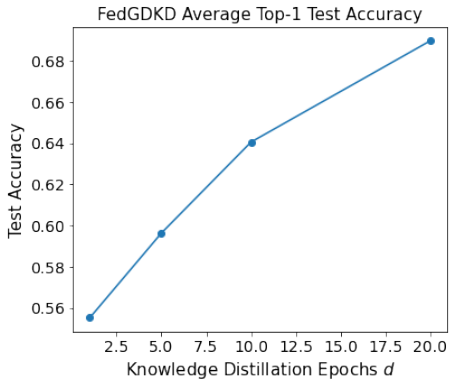
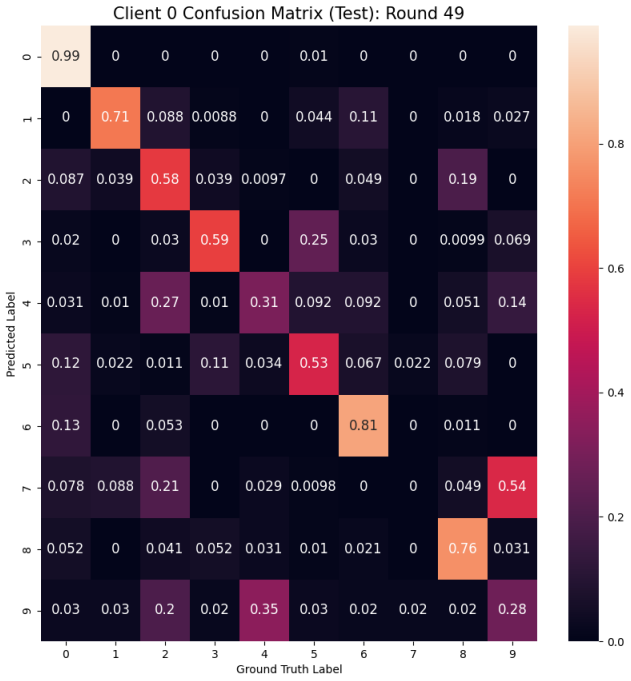
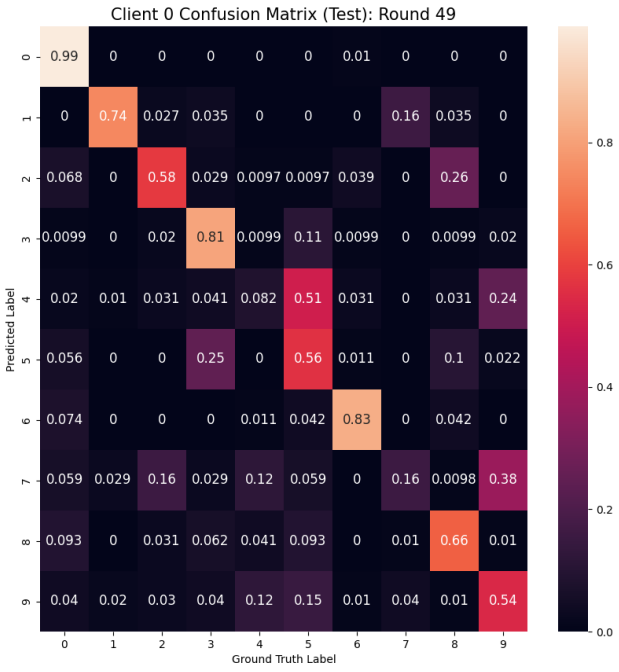


Figure 5.9: Average top-1 accuracy with varying d

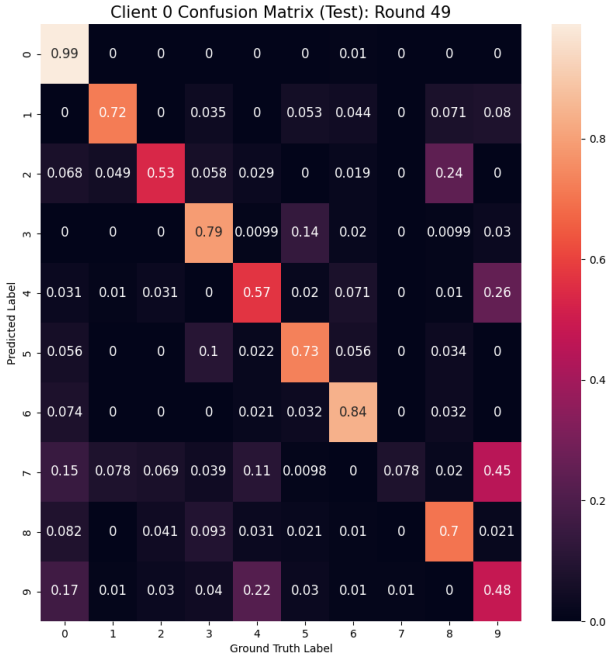
We can see in Figure 5.9, that this hypothesis is indeed correct as there is indeed an almost linear positive trend between d and the average top-1 test accuracy. However, future work would be to test larger values of d, to see if a divergence occurs in the local objective and negatively affects the result. We can further confirm that more global knowledge is transferred by looking again at the confusion matrices for Client 0.



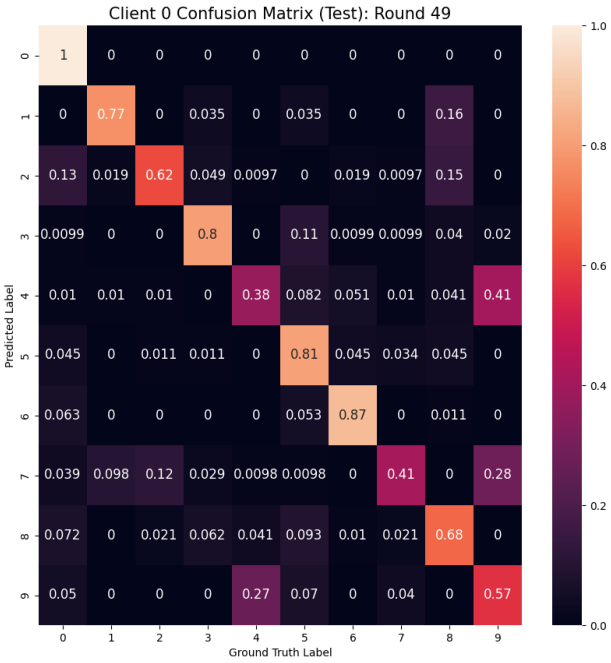
(a) $d = 1$



(b) $d = 5$



(c) $d = 10$



(d) $d = 20$

Figure 5.10: KD Parameter Study: Client 0 confusion matrices (x-axis: Ground Truth Label; y-axis: Predicted Label) after training when varying d (MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$).

As can be seen in Figure 5.10, there is again a stronger diagonal pattern as d increases. This suggests that the per-class precision increases and the more global knowledge, that is otherwise inaccessible, has been transferred to Client 0. In summary, we see that increasing d, should increase the performance of the algorithm. However, a balance must be struck with the additional computational cost incurred, as discussed in Section 5.4.1.

Distillation Dataset Size N

Finally, we examine the effects of the size of the distillation data set N. For knowledge distillation to work effectively, a common dataset with a good representation per class needs to be used. The representation of classes is key to effectively teaching the student model (client local model), as if a greater number (and more diverse set) of examples are given for each class, more knowledge can be transferred due to inter-class diversity. Therefore, we hypothesise that a larger N, will allow for better global knowledge transfer. We test this by examining the final performance when varying $N = [100, 1000, 5000, 10000]$.

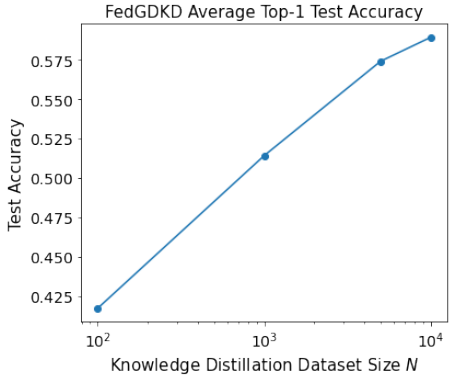


Figure 5.11: KD Parameter Study: Average top-1 accuracy with varying N

In Figure 5.11, the average top-1 test accuracy (y-axis) has been plotted against the varying values of N (x-axis) on a logarithmic scale (base 10). We can see that there is a logarithmic trend and that an

increase in N results in diminishing gains in performance. This can likely be attributed to the lack of diversity in samples at higher N, that is, similar samples are repeated. To determine whether this is indeed the case, we can examine the final Frechet Inception Distance (FID) score [46] reached by the generators in each run. The FID also includes a component that scores the diversity of the generated images. Additionally, the lower the score, the better, as the features of the generated images and the real images therefore match more closely. Note that each score is calculated from 10,000 real images and 10,000 generated images.

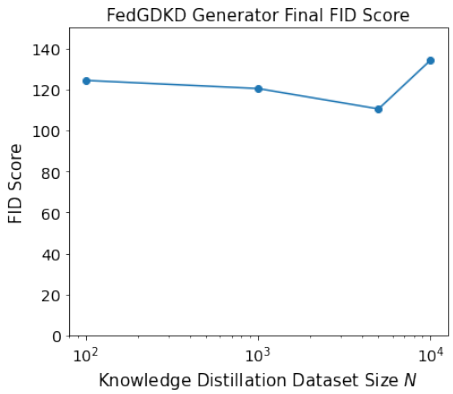
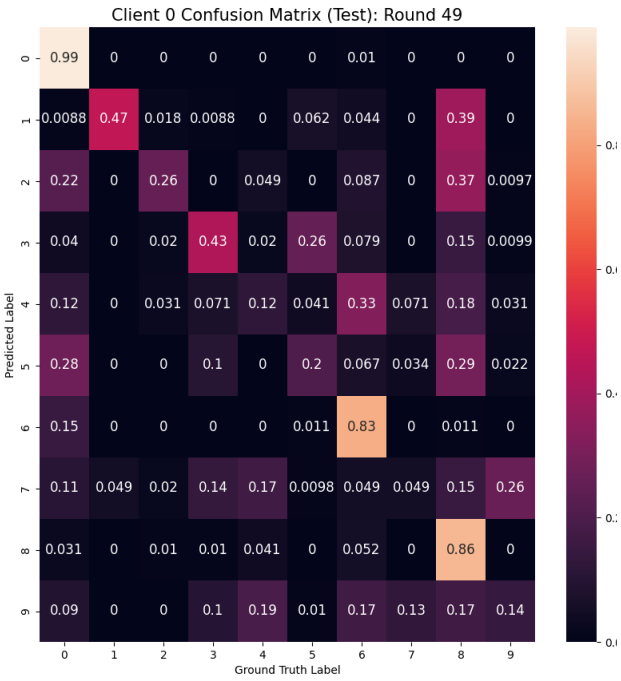


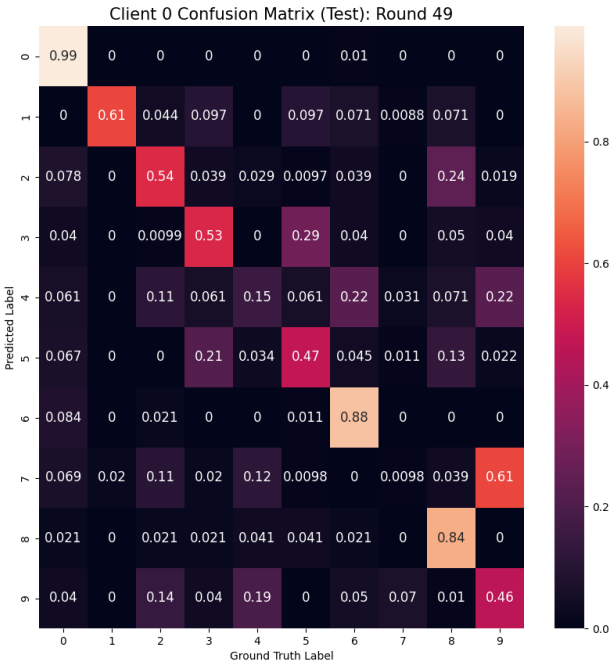
Figure 5.12: KD Parameter Study: FedGDKD final generator score with varying N

The final FID scores of the generators for each N (Figure 5.12) do not differ significantly. Therefore, it is reasonable to say that all the generators have similar performance and so the diversity will be similar - so there is a threshold N reaches before the generator will produce similar samples with no additional benefit.

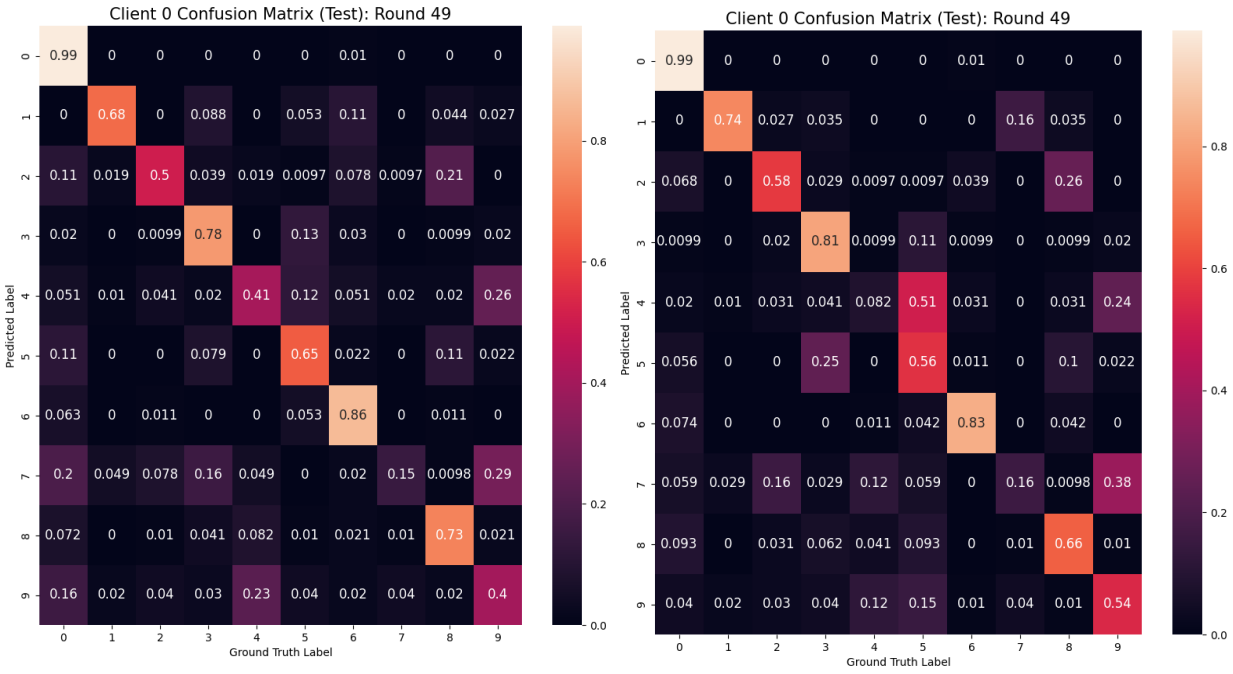
Finally, we can examine Client 0 again to ensure that there is indeed a diminishing increase in the amount of global knowledge transferred as N increases.



(a) $N = 100$



(b) $N = 1000$



(c) $N = 5000$

(d) $N = 10000$

Figure 5.13: KD Parameter Study: Client 0 confusion matrices (x-axis: Ground Truth Label; y-axis: Predicted Label) after training when varying N (MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$).

We can see in Figure 5.13, that the per class precision has a reduction in increase as N increases, suggesting the logarithmic increase in the global knowledge transferred.

In summary, we see an increase in global knowledge transferred as N increases. However, there is a diminishing increase that should be taken into account as N greatly affects the efficiency of FedGDKD (Section 5.4). Further work can be done to improve the diversity of synthetic samples produced by the generator. Doing so should increase the threshold of N where the returns diminish. This improvement would also allow a smaller N to be selected as more knowledge can be distilled in a smaller, more diverse dataset, further improving efficiency.

Overall Summary

Overall, we have explored the effect that each KD hyperparameter has and have seen that generally bigger is better! However, a balance must be struck between the increase in performance and the potential decrease in efficiency.

5.2.5 Generator Performance

In this thesis, we solve the problem of heterogeneous model federated learning using GAN-based (Section 2.1.3) data-free knowledge distillation in our proposed method FedGDKD (Chapter 4). FedGDKD's primary purpose is to train an accurate classifier that performs well on a global objective, however in doing so we train a generator that aims to produce realistic synthetic data. The generator itself, is a vehicle for global knowledge transfer as discussed in Section 5.3.1 and in fact its training is not reliant on correcting model drift through the data-free distillation stage (Section 5.3.2). Therefore, could the proposed method also be used as a means of GAN training in a federated setting. In this section of the evaluation, we investigate just that by comparing performance to the state-of-the-art [35] algorithm.

To perform the comparison, we use the same generator and a classifier architecture as mentioned in Section 5.2.1 but with an additional discriminative task head. This is because FedGAN implements an ACGAN [5]. For FedGDKD, this discriminative head is not used as it is not required. To measure how close the synthetic samples produced by the trained generators are to the real data, we use the Frechet Inception Distance (FID) [46]. This is a measure of how close the feature distributions match between two image-like datasets, therefore the lower the FID score the, the more realistic the generated samples

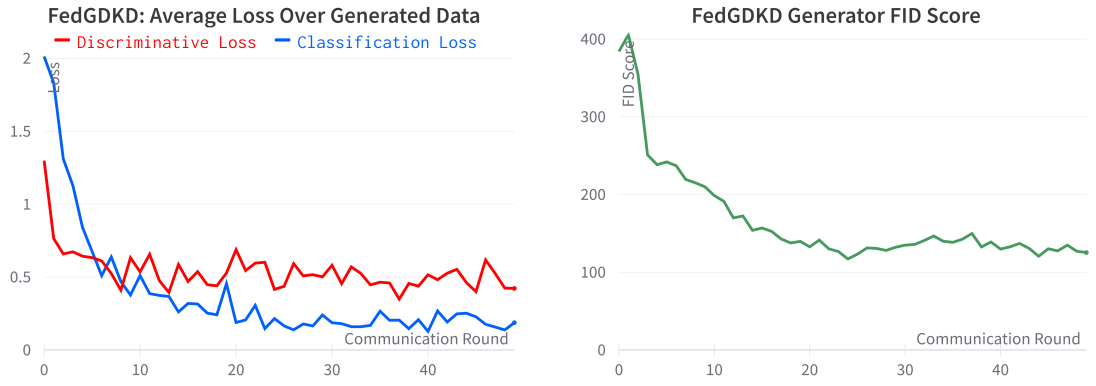
are. We test both algorithms with the same hyperparameters as mentioned in Section 5.1.2 and use a sample size of 10000 real and synthetic images to calculate the FID score over. We do this for several dataset settings:

Generator Final FID Score			
Dataset	Setting	FedGAN	FedGDKD
MNIST $r = 25\%$	$\alpha = 0.5$	191	**43.9**
	$\alpha = 0.1$	131	**76.9**
	$\alpha = 0.05$	157	**89.5**
MNIST $r = 10\%$	$\alpha = 0.5$	272	**49.0**
	$\alpha = 0.1$	327	**125**
	$\alpha = 0.05$	383	**106**
EMNIST $r = 25\%$	$\alpha = 0.5$	153	**48.5**
	$\alpha = 0.1$	149	**58.6**
	$\alpha = 0.05$	166	**103**
EMNIST $r = 10\%$	$\alpha = 0.5$	**173**	185
	$\alpha = 0.1$	144	**57.0**
	$\alpha = 0.05$	**143**	200

Table 5.9: Generator Performance: FID score summary

We can see in Table 5.9, that FedGDKD generally outperforms FedGAN in most settings. This is a very interesting result as GANs are difficult to train, an explanation is that the novel GAN loss implemented in FedGDKD (Section 4.2.1) improves stability of training through stronger learning signals. The class logits are not funneled through a sigmoid activation and so its gradients will not saturate for very high and low values, therefore mitigating the vanishing gradient problem.

Additionally, the classifier loss (Equation (4.4)) includes a contradictory term which can act like an additional regulariser which prevents the discriminator (classifier) from dominating the generator i.e. the discriminator always classifies all synthetic samples as fake causing the generator to not improve. Whereas, since there is separation in FedGAN's discriminator due to the separate task heads for the discrimination task and classification task, a sample can be both fake and be classified. This means that the discriminator is not battling itself to optimise one task or the other and can optimise for both allowing it to dominate the generator. We can see this by observing the discriminative and classification loss (eq. (4.5)) of FedGDKD over its generated data:



(a) Average component-wise loss over generated data

(b) Generator FID Score

Figure 5.14: Generator Performance: FedGDKD GAN Metrics (MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$)

Figure 5.14a, shows that indeed the classifiers (client models) tend to optimise one of the two losses over generated data. Towards the beginning of training, the discriminative loss is generally lower, as it is apparent that the synthetic images are of worse quality in Figure 5.14b. We then see as the images get more realistic the classification component of the loss decreases whereas the discriminative component of the loss stagnates. This discriminative component will then act as a regulariser as previously

mentioned (Section 4.2.1), preventing the classifier from dominating the generator.

This additional regularisation also prevents the client models from overfitting on their private data. Overfitting can cause worse generalisation, leading to worse test performance as well as lead to further privacy leakage if access to the model is given (discussed in Section 5.5.1).

In summary, we have shown that the proposed method FedGDKD is a more robust and private method of conditional GAN training under a federated setting. Also note that FedGAN requires both the discriminator and generator parameters to be shared and aggregated with the central aggregation server, whereas FedGDKD only shares the generator. This is both cheaper in terms of communication as well as more private as only one model is shared.

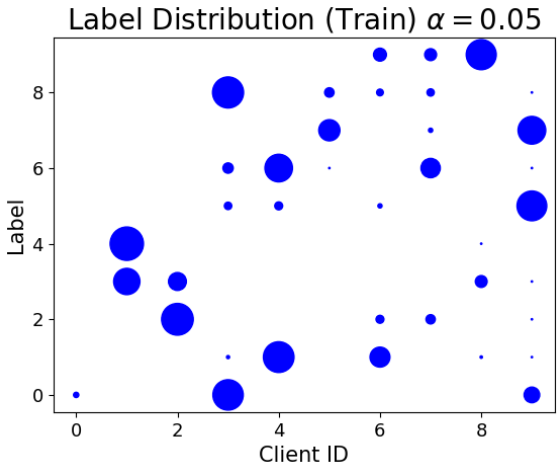
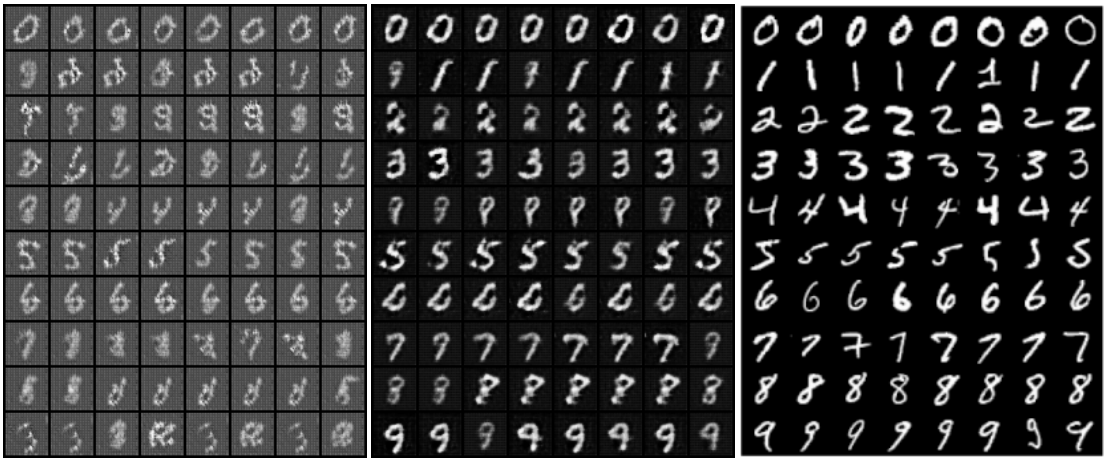


Figure 5.15: Client training label distribution



(a) FedGAN (b) FedGDKD (c) Ground Truth

Figure 5.15: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.05)$, $r = 10\%$)

Note: additional examples of generator outputs can be seen in Appendix B.5.

5.3 Ablation Study

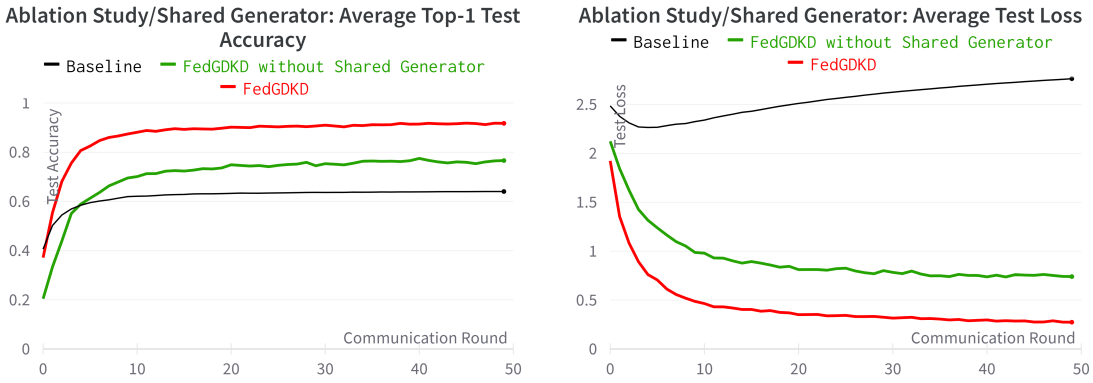
In this evaluation section, we discuss and investigate the importance of the main features in our proposed method FedGDKD. We ablate each feature one by one to see the effect this has and comment on the initial reasoning behind its use.

5.3.1 Shared Generator

In our proposed FedGDKD method, the clients share a generator whose parameters are aggregated on the central server. This generator is hypothesised to model the global joint distribution of the training samples and their labels. It plays an extremely important part in global knowledge extraction and transfer, where clients will be exposed to realistic samples from under-represented classes and, as such, generalise better.

To study the effects in isolation, we can ablate the shared generator by removing shared updates and allowing each client's clone of the generator to diverge on their private data. The difficulty in this study is that we are unable to generate an identical distillation dataset at each client. Unfortunately, there is no way around this, so the noise vector is sent as is. This will result in an identical number of generated samples per class; however, nonidentical samples will be generated.

To create the comparison, we ran the algorithm with and without a shared generator on the same dataset, MNIST, and setting, Dir($\alpha = 0.5$), $r = 25\%$. The results are as follows:



(a) Top-1 average test accuracy

(b) Average test loss

Figure 5.16: Ablation study on shared generator: test curves (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

Evidently in Figure 5.16, there is a significant difference when the generator model is not shared. We see a significant drop in accuracy from 91.7% (with shared generator, red) to a peak of 77.5% (without shared generator, green). This is likely due to the local copies of the generator diverging (model drift) towards the optima of the non-iid distribution in the client data. Attempting to replicate the joint distribution of the unbalanced private data, it will be biased towards well-represented classes and, as such, produce low-quality synthetic samples for under-represented ones. The effect of this is "bad" teacher logits for the subsequent data-free knowledge distillation stage Section 4.2.3. This supports the hypothesis that the shared generator better approximates the global distribution of the data.

Continuing the previous statement that local generators diverge to the local optima of private client datasets, we can investigate the FID[46] scores on average and for a specific client.

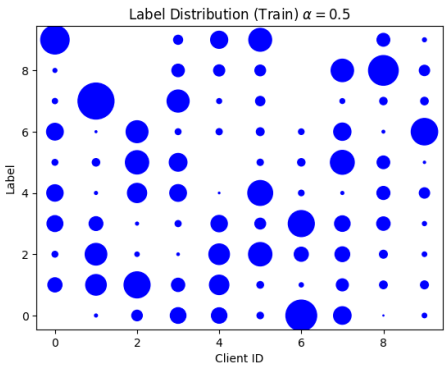
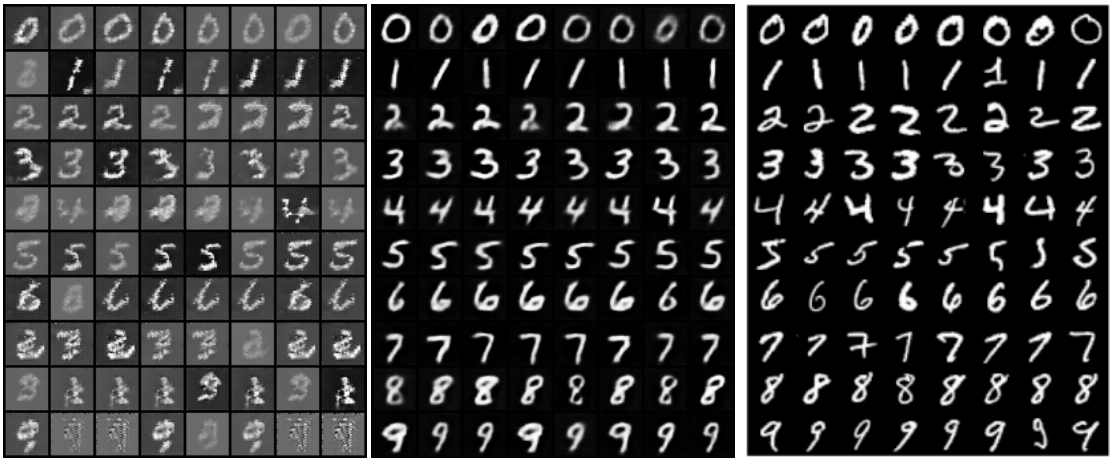


Figure 5.17: Training Label Distribution Between Clients



(a) FedGDKD without shared generator (Client 6)

(b) FedGDKD

(c) Ground truth

Figure 5.18: Ablation study on shared generator: generator output (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

We can see in Figure 5.17, that the client with ID 6 (Client 6) does not have classes/labels (in the case of MNIST the label directly corresponds to the digit) 7,8 and 9 present in their private training dataset. Therefore, it should be fair to assume that the client 6 generator will not be able to produce realistic samples for these classes. Looking at the output of the generator of client 6 Figure 5.18a and comparing it with the output of the shared generator Figure 5.18b, this is indeed the case. This provides further confirmation of the hypothesis.

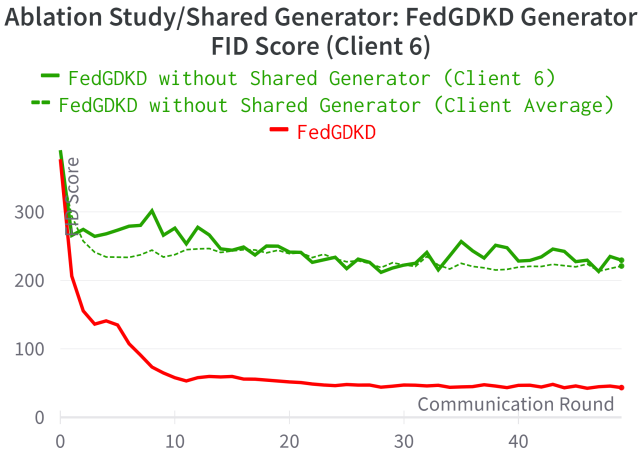
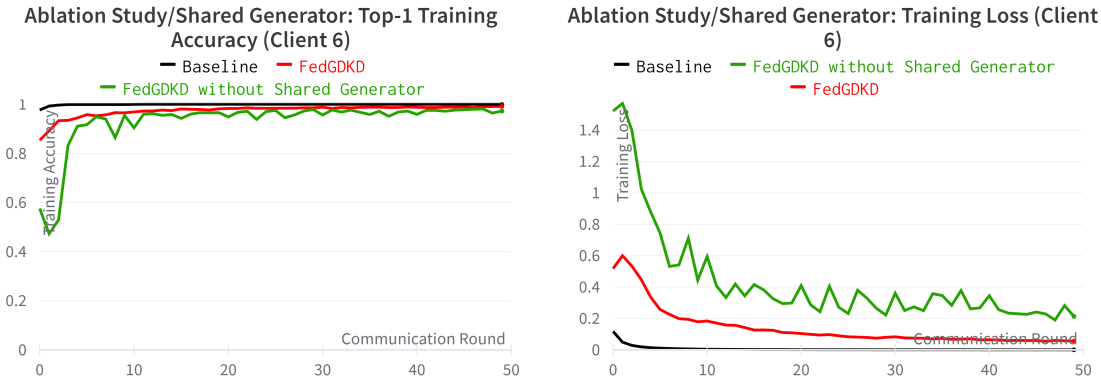


Figure 5.19: Ablation study on shared generator: Client 6 generator FID score (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

Looking deeper into the FID scores [46] of the client 6 generator (Section 5.3.1). The score represents the distance between the feature distributions of the generated data and the real data. Therefore, the lower the score the better. First, note that the average client generator scores (dotted green) are consistently and significantly higher without a shared generator than with(red). This is again most likely due to each client generator diverging. Also, note that the scores have an unstable trend for the client 6 generator (solid green). This is likely due to ineffective data-free knowledge distillation, where inconsistent lower-quality distillation datasets are being used. Thus, noisy perturbations in training, as can be seen in Figure 5.20a and Figure 5.20b.



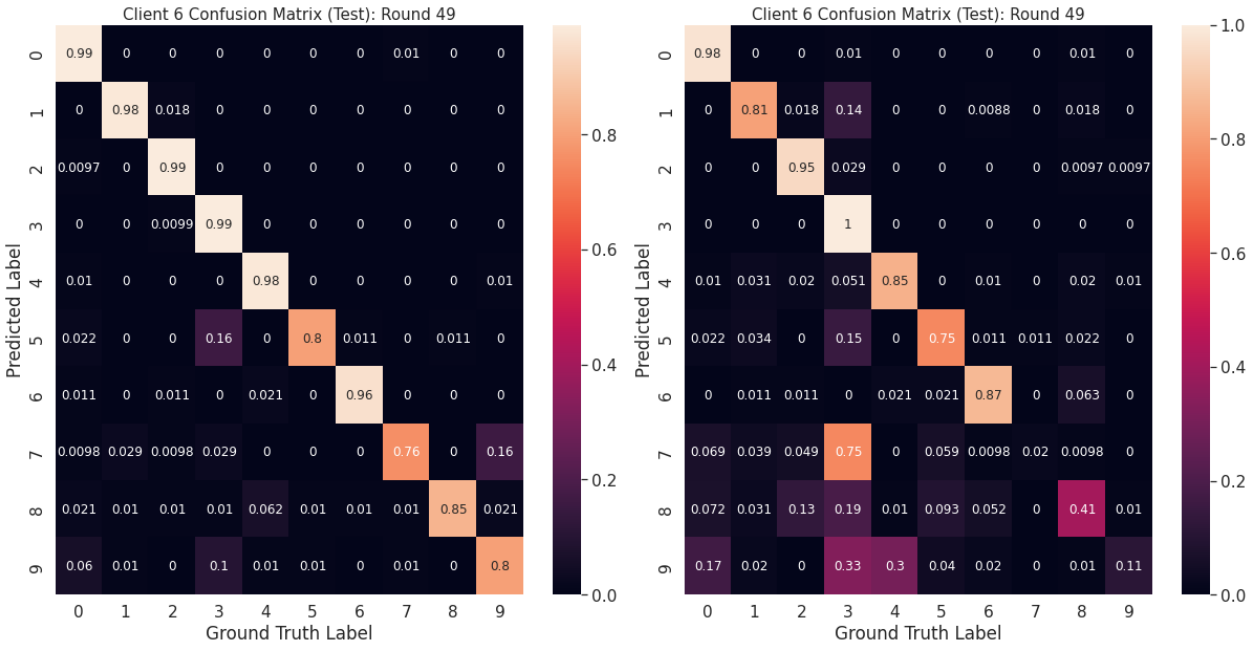
(a) Training Top-1 Accuracy

(b) Training Loss

Figure 5.20: Ablation study on shared generator: Client 6 training metrics (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

Finally, we can examine the confusion matrices of Client 6's local model to determine the level of global knowledge transferred. In alignment with the hypothesis that the shared generator enables effective global knowledge transfer, we should see that the missing labels are classified correctly on the test set and misclassified without it. In fact, this is the case, as can be seen in Figure 5.21. Interestingly, label 7 (digit 7) is misclassified with a high probability as label 3 (digit 3) in the case without the generator Figure 5.25b. This is due to a 3-like sample being generated by Client 6's generator (Figure 5.18a). Again, this further highlights the importance of a good generator in the proposed method **FedGDKD**.

In summary, we have shown that sharing the generator model between clients allows effective global knowledge transfer and more closely approximates the true joint global distribution of the data. However, this also highlights a tight coupling between the generator and the client model, where the generator must produce realistic samples to increase the accuracy of the client model.



(a) FedGDKD

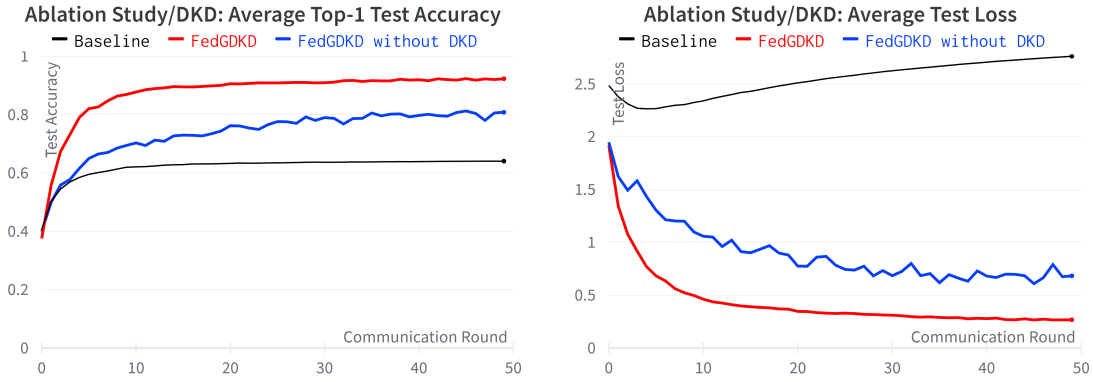
(b) FedGDKD without Shared Generator

Figure 5.21: Ablation study on shared generator: Client 6 confusion matrices at the end of training (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

5.3.2 Data-Free Knowledge Distillation

To study the effects that the data-free knowledge distillation (DKD) stage (Section 4.2.3) has on our proposed method FedGDKD, we perform an ablation of the stage. This stage was hypothesised to help impart global knowledge to each client model, as well as reduce model drift to improve the training of the generator.

To create the comparison, we ran the algorithm with and without the stage on the same dataset, MNIST, and setting, Dir($\alpha = 0.5$), $r = 25\%$. To give a perspective on performance, the baseline (client models train only on client private data) is provided. The test performance can be seen in Figure 5.22:



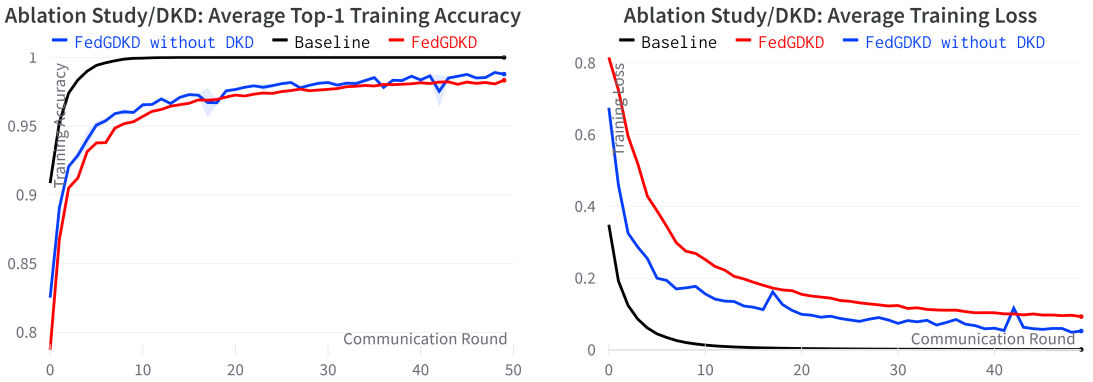
(a) Top-1 average test accuracy

(b) Average test loss

Figure 5.22: Ablation study on data-free knowledge distillation (DKD): test curves (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

As can be seen in Figure 5.22, we can clearly see the performance benefit that DKD has. The final average top-1 test accuracy (Figure 5.22a) of FedGDKD with DKD (red) is 91.7% compared to without DKD (blue) is 81.2% at its peak. This is a significant drop of more than 10%. Indeed, this supports the claim that DKD is imparting global knowledge to each client model.

We can further investigate this claim by looking at the effect of ablation on the training performance:



(a) Top-1 average training accuracy

(b) Average training loss

Figure 5.23: Ablation study on data-free knowledge distillation (DKD): training curves (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

In Figure 5.23, two observations can be made. The first is that DKD's regularising effect is clearly present (as knowledge distillation is added as a regularisation term). This is evident as the training accuracies are consistently lower and losses are higher throughout the communication rounds when DKD is being used (red). The convergence seems to be quite similar, if not slightly slower, than without DKD (blue). This suggests that DKD helps to reduce overfitting on the small private datasets, further reducing bias towards the local objective. However, also notice that without DKD, the test curves do not match

the baseline (black) in Figure 5.23. This is most likely due to the novel classification loss used in local GAN training acting as additional regularisation (Section 5.2.5).

The second observation is that the training curves are smoother with DKD than without it. This can be attributed again to the regularisation effect, which leads to model alignment (as this is the average accuracy, meaning that the variance between model training accuracies is low). The goal of DKD is to align the model with a shared teacher model (an ensemble of other clients) that has global knowledge. This suggests that as the generator produces samples from under-represented classes (when compared to a client's own private dataset), they will have less of an effect as the client model has been exposed to them through DKD.

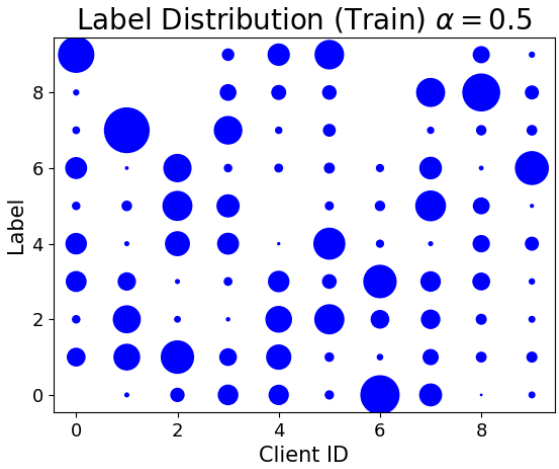


Figure 5.24: Client training label distribution (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

Both observations in the training data set further support the claim that DKD imparts global knowledge on the client model and reduces model drift. Another, more concrete investigation is to see the effect DKD has on a client whose private data is completely missing samples from some classes. Looking at the distribution of training labels between clients (Figure 5.24), we can see that the client with ID 6 (Client 6) lacks labels 7,8 and 9. Therefore the final test performance on these labels will be telling as to how well the global knowledge is imparted onto Client 6.

There is a clear distinction between the confusion matrices when using DKD (Figure 5.25a) versus without (Figure 5.25b). We can see that in fact the missing labels are more likely to be misclassified when the DKD is removed. This solidifies the claim that DKD plays an important role in global knowledge transfer.

To explore the hypothesis that DKD also improves generator training, we can look at FID[46] scores during communication rounds of the shared generator. The FID score is a metric based on how close the feature distributions are to each other between two image sets. In this study, we examine the distance between the generated distillation dataset and a sample of MNIST images (both of size 10,000). The lower the metric, the closer the distributions are, i.e. the more realistic the generated data are.

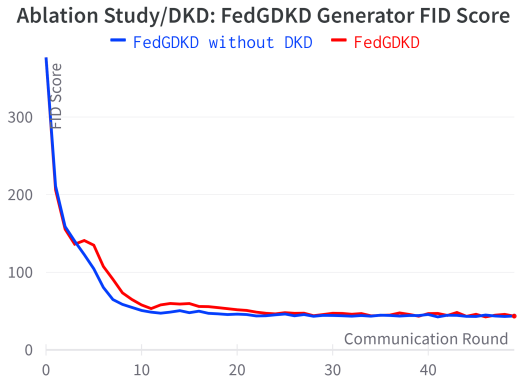
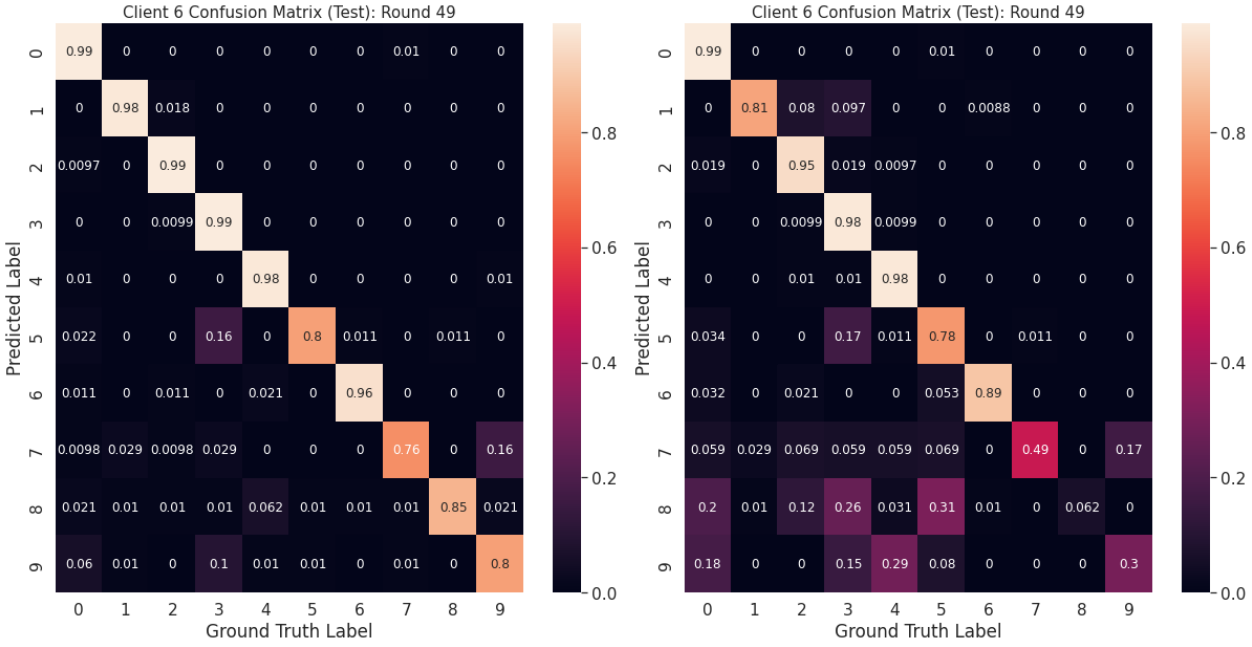


Figure 5.26: Ablation study on data-free knowledge distillation (DKD): FID scores (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

In Figure 5.26 and (visually in Figure 5.27), it is actually not the case that DKD (red) improves generator training, in fact, it makes it slightly worse in early communication rounds. This could be due to the double-edged effect of the knowledge distillation regularisation, where the low-quality synthetic samples will cause "bad teaching" because the client models will likely lack consensus on what the sample's class could be leading potentially high realness scores for low-quality images. This could in turn provide the generator with positive feedback for synthesising low-



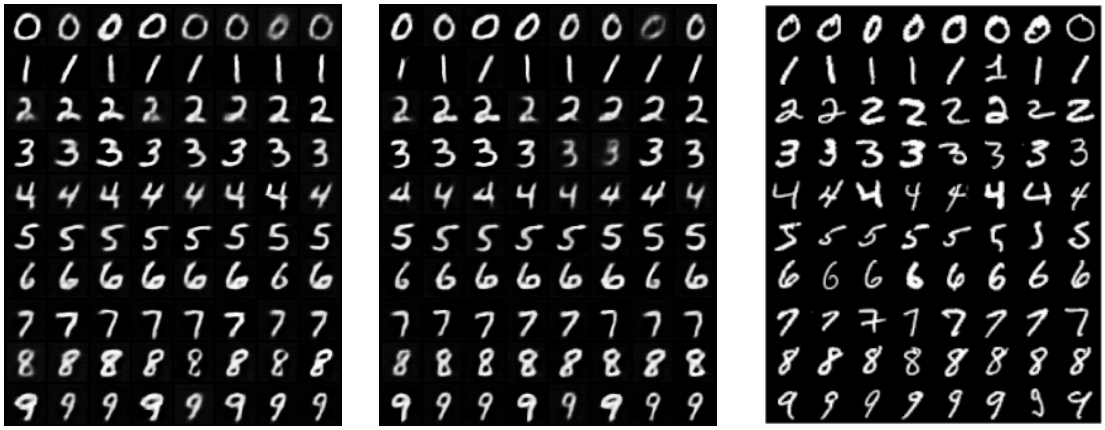
(a) FedGDKD

(b) FedGDKD without DKD

Figure 5.25: Ablation study on data-free knowledge distillation (DKD): Client 6 confusion matrices at the end of training (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$).

quality samples, and so increases the FID metric. Therefore, an improvement could be to set a schedule for the knowledge distillation weight α_{KD} that starts low to avoid the effect of bad synthetic data and increases as the generator improves. Investigating this improvement is left for future work.

In summary, this ablation study has been worthwhile. It shows both the importance of DKD in effective global knowledge transfer and model alignment (model drift resolution). However, it disproves the hypothesis that DKD improves GAN training. The insight derived from this study has opened a path of further research into scheduling the knowledge distillation weight parameter α_{KD}.



(a) FedGDKD

(b) FedGKDK without DKD

(c) Ground Truth

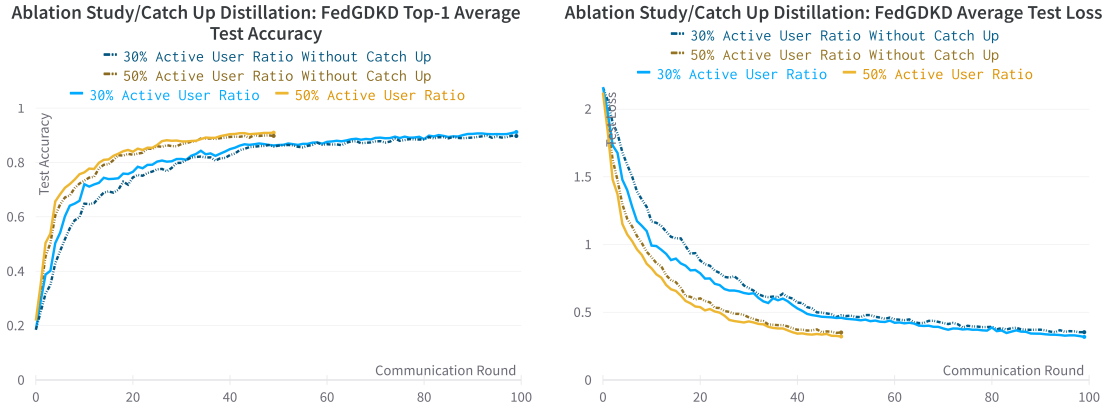
Figure 5.27: Ablation study on data-free knowledge distillation (DKD): a comparison of the generated data produced at the end of training $Dir(\alpha = 0.5)$, $r = 25\%$).

5.3.3 Catch Up Distillation

In this ablation study, we see the effectiveness of the "catch up" distillation step in the proposed method FedGDKD (Section 4.2.3). The idea behind this step was to address model drift in client models when

they have not participated in the previous communication round. Before starting local GAN training they will undergo co-distillation where ensemble teacher knowledge from the previous round is imparted onto the client's model.

To test the effect this step has, we perform an ablation study under the partial client participation scenario as explored in (Section 5.2.3). We explore active user ratios of 30% and 50% (not full participation as the catch up step will never occur). This was performed under the same conditions as (Section 5.2.3): on the MNIST dataset with non-iid parameter $Dir(\alpha = 0.5)$ and the training set sample ratio $r = 25\%$. Therefore, the client participation will be the same as well.



(a) Top-1 average test accuracy

(b) Average test loss

Figure 5.28: Ablation study on catch up distillation: test curves (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)

We can see in fig. 5.28, that the removal of the catch up distillation step has an almost negligible negative impact on the final performance of the algorithm as can be seen in Table 5.10.

Active User Ratio (%)	Catch Up Distillation	Communication Rounds T	Final Average Top-1 Test Accuracy (%)
50	✓	50	91.0
50	✗	50	89.8 (-1.20)
30	✓	100	91.2
30	✗	100	89.7 (-1.47)

Table 5.10: Ablation study on catch up distillation: summary of final top-1 test accuracies

However, another observation that can be seen is a drop in convergence rate towards the beginning of the runs (smaller gradient magnitude). The lower the active user ratio, the slower the initial convergence. This suggests that catch up improves the convergence of the algorithm, and in cases where there are limitations on the number of communication rounds, it could be an important feature.

We can weigh up the additional overhead of the catch up step with performance increase. Let us assume a stationary (does not change over time) active user ratio which can be uniformly sampled $p \in (0..1]$ and let us assume a binomial distribution. Then we can say that this step will be executed each round by a client with a probability of $(1-p)$, as this is the probability that they will not participate in the previous round. Let us go through an example, in the case where $p = 0.3$ and $T = 100$, the expected number of rounds a client will participate in is $\mathbb{E}_p[participate] = p \times T = 30$ and the expected number of rounds the client will execute catch up distillation: $\mathbb{E}_p[catchup] = (1-p) \times \mathbb{E}_p[participate] = 21$. So we can clearly see that the lower the active user rate, the higher the likelihood per round where the client is participating of additional overhead, but also the faster the convergence. Therefore, this additional overhead could be tolerable for clients not wanting to stop early, i.e., stop participating completely after a number of rounds.

Alternatively, for clients prepared to participate in all T communication rounds. The expected execution of the catch-up step occurs at most in 25% of rounds and will involve this additional overhead as $(1-p)p \leq 0.25 \forall p \in (0..1]$. This may be unfavourable for some as even without this additional overhead the final performance is just as good.

Overall, these observations justify the use of catch up distillation in cases where convergence speed is of high priority and the removal in cases where clients are willing to participate for extended periods of time.

5.4 Efficiency

5.4.1 Computational Complexity

An ideal algorithm should be as cheap to compute as possible; this is an obvious statement, as it improves the speed of execution. In the context of federated learning, this factor greatly affects client selection. If the algorithm is expensive to run, it could cause clients with low computing resources to be unable to complete the round in time or not be able to participate at all. This is an active area of research [1, 47].

Therefore, we should investigate the computational cost for the clients in our proposed method FedGDKD and compare this with other state-of-the-art heterogeneous model FL algorithms. This can be done through analysing the running time of each algorithm. We do this using asymptotic notation [48].

To do this, we introduce some simplifying assumptions. First, we assume that floating-point operations (flops) take a constant time to execute. We can then estimate the time complexity as the number of flops. Then let us assume that the number of flops for forward and backward passes through a (deep learning) model M for a single input (e.g. single image) is proportional to the number of parameters θ_M. Therefore, we can derive the time complexity by the number of forward and backward passes through a model. The summary of time complexities is as follows

Algorithm	Initialisation	Communication Round
FedMD	$\mathcal{O}((n_{public}\|\mathcal{D}_{public}\| + n_{private}\|\mathcal{D}_k\|)\theta_{C_k})$	$\mathcal{O}((n_{digest}\|\mathcal{D}_{public}\| + n_{revisit}\|\mathcal{D}_k\|)\theta_{C_k})$
FD + FAug	$\mathcal{O}(\|\mathcal{D}_{aug}\|\theta_G)$	$\mathcal{O}(n(\|\mathcal{D}_{aug}\| + \|\mathcal{D}_k\|)\theta_{C_k})$
FedDTG	0	$\mathcal{O}((n\|\mathcal{D}_k\| + N)(\theta_{C_k} + \theta_G) + n\|\mathcal{D}_k\|\theta_D)$
FedGDKD	0	$\mathcal{O}((n\|\mathcal{D}_k\| + (d+1)N)\theta_{C_k} + (n\|\mathcal{D}_k\| + N)\theta_G)$

Table 5.11: Summary of algorithm time complexities at client

where $|*|$ is a length function that in the case of datasets $|\mathcal{D}|$ refers to its length. We have estimated the number of passes that have passed through the client models.

First, we give an explanation of how these figures were derived. In the case of FedMD (Section 3.1.1), the complexity of the initialisation (before the communication rounds) comes from the transfer learning step. In this step, the client model C_k is trained (forward and backward passes) first on the public/proxy dataset $\mathcal{D}_{public}$ until convergence (in n_{public} epochs) and then on the private data set $\mathcal{D}_k$ until convergence (in $n_{private}$ epochs). We can then estimate the total number of inputs passed through C_k as proportional to the sum of the size of these datasets multiplied by the epochs required for training: $2(n_{public}|\mathcal{D}_{public}| + n_{private}|\mathcal{D}_k|)$ (the constant 2 can be removed). We can then multiply this by the operations C_k performed on a single input sample, which is proportional to the number of parameters in the model $|C_k|$. Similarly, for communication rounds, the client executes two stages: digest, where the distillation of knowledge occurs for C_k on the public dataset $\mathcal{D}_{public}$ for n_{digest} epochs; revisit, where C_k trains further on the private dataset $\mathcal{D}_k$.

Using the same logic, we can analyse FD + FAug (Section 3.1.2). Before communication rounds, all that occurs is the optional FAug step, where clients share undersampled labels from their private dataset to train a generative model G on another server. This incurs no cost to the client. However, G is then used to augment the training data with $\mathcal{D}_{aug}$ to alleviate data heterogeneity (degree of non-iid). This comprises a forward pass over G proportional to the size of this augmented dataset. During a communication round, the client C_k is trained (by knowledge distillation) on this augmented dataset $\mathcal{D}_{aug} \cup \mathcal{D}_k$ for n epochs.

Again, we can use this logic to derive the computational complexity for our proposed method FedGDKD (Chapter 4). The initialisation complexity of the client is 0 as no computation occurs prior to commencing communication rounds. During a communication round, there are two stages that involve computation at the client. First local GAN training; this is where the clients train a copy of the shared generator G adversarially with their local model C_k. During this stage, the private dataset $\mathcal{D}_k$ is iterated over for n epochs in mini-batches. For each mini-batch, the generator G produces synthetic data of the

same batch size, resulting in $|\mathcal{D}_{gen}| = |\mathcal{D}_k|$. The synthetic data passes forward and backward through G and $2|\mathcal{D}_k|$ samples are passed forward and backward through C_k. This gives a time complexity of

$$\mathcal{O}(n(2|\mathcal{D}_k|\theta_G + 4|\mathcal{D}_k|\theta_{C_k})) = \mathcal{O}(n|\mathcal{D}_k|(\theta_{C_k} + \theta_G)) \tag{5.1}$$

The second stage involving client computation is the data-free knowledge distillation stage. Here, a synthetic dataset for distillation ($\mathcal{D}_{KD}$) of size N is generated by G (only forward pass) and passed through the client model C_k retrieve class logits (only forward pass). Then during co-distillation, D_{KD} is passed forward and backward through the client model C_k for d epochs. Additionally, the client may not have participated in the previous communication round, so the cost of co-distillation can be doubled in the worst case. This leads to a time complexity of

$$\mathcal{O}(2[N(\theta_G + \theta_{C_K}) + 2dN|\theta_{C_k}]) = \mathcal{O}(N(\theta_G + \theta_{C_K}) + dN\theta_{C_k}) \tag{5.2}$$

where summing eq. (5.1) and eq. (5.2) produces the result in Table 5.11:

$$\begin{aligned} &\mathcal{O}(n|\mathcal{D}_k|(\theta_G + \theta_{C_k}) + N(\theta_G + \theta_{C_k}) + dN\theta_{C_k}) \\ &= \mathcal{O}((n|\mathcal{D}_k| + N + dN)\theta_{C_k} + (n|\mathcal{D}_k| + N)\theta_G) \\ &= \mathcal{O}((n|\mathcal{D}_k| + (d+1)N)\theta_{C_k} + (n|\mathcal{D}_k| + N)\theta_G) \end{aligned} \tag{5.3}$$

We can also derive a similar result for FedDTG. However, we account for the additional discriminator only taking part in GAN training and only one distillation epoch, resulting in:

$$\begin{aligned} &\mathcal{O}((n|\mathcal{D}_k| + N)\theta_{C_k} + (n|\mathcal{D}_k| + N)\theta_G + n|\mathcal{D}_k|\theta_D) \\ &= \mathcal{O}((n|\mathcal{D}_k| + N)(\theta_{C_k} + \theta_G) + n|\mathcal{D}_k|\theta_D) \end{aligned} \tag{5.4}$$

To compare the time complexities, let us analyse the added overhead in a consistent setting where the client participates in T communication rounds. We can assume that the three algorithms run on the same datasets ($|\mathcal{D}_k|$ is constant). Furthermore, during communication rounds, the normal training epochs (without knowledge distillation) are the same and constant $n = n_{revisit}$. Also, the datasets used for knowledge distillation are of the same size $|\mathcal{D}_{aug}| = |\mathcal{D}_{public}| = N$. Finally, the client models C_k are the same so θ_{C_k} is constant. The time complexity can then be simplified to

Algorithm	Initialisation	Communication Round	Overhead (T Rounds)
FedMD	$\mathcal{O}(n_{public} + n_{private})$	$\mathcal{O}(n_{digest})$	$\mathcal{O}(n_{public} + n_{private} + Tn_{digest})$
FD + FAug	$\mathcal{O}(\theta_G)$	$\mathcal{O}(1)$	$\mathcal{O}(\theta_G + T)$
FedGDKD	0	$\mathcal{O}(d + \theta_G)$	$\mathcal{O}(T(d + \theta_G))$
FedDTG	0	$\mathcal{O}(\theta_G + \theta_D)$	$\mathcal{O}(T(\theta_G + \theta_D))$

Table 5.12: Summary of time complexities for algorithm additional overhead for client ranked lowest to highest (under a consistent setting)

In Table 5.12, we can clearly see that the additional overhead of the FedDTG has the highest cost. This is because the number of parameters for the generator θ_G (millions) is orders of magnitude larger than the number of epochs (tens), that is, $\theta_G >> n_{private} + n_{public}$. We can also say the same for θ_D, hence coming to the conclusion that FedDTG is the most expensive. With similar reasoning and lack of discriminator, our proposed FedGDKD method is the second most expensive as $\theta_D >> d$. Following this, is FD + FAug, this cost is due to the production of synthetic data at initialisation, where also $\theta_G >> Tn_{digest}$. Therefore, the computationally cheapest method is FedMD where the proxy dataset is assembled in advance.

In summary, our proposed FedGDKD method is more computationally efficient than the closely related FedDTG but not as cheap as the data-based alternatives. However, we must also consider the difficulty in producing this proxy dataset in FedMD and the privacy issues associated with data sharing in FD + FAug. Therefore, in the circumstances where this is the case, FedGDKD is a good alternative.

5.4.2 Communication Cost

Another challenge in Federated Learning (FL) is the cost of network communication. As FL is a distributed algorithm, relying on the network is of utmost importance. There could be heterogeneity in

network bandwidth, download and upload speeds at each client, so minimising the overhead of communication is highly important. Therefore, we consider comparing communication costs across algorithms that allow heterogeneous models to align with the initial aim of this thesis.

Let us assume that the cost of transmitting a single byte of information value over a network is constant. Then we can estimate the cost of communication through the number of bytes transmitted. In the case of neural network models M, the number of bytes is proportional to the number of parameters θ_M, for a logit vector, the cost is proportional to the number of entries in the vector. Finally, for a dataset $\mathcal{D}$, it is proportional to the length of the dataset $|\mathcal{D}|$ multiplied by the size of each entry.

Algorithm	Initialisation	Communication Round
FedMD	$\mathcal{O}(\mathcal{D}_{public})$	$\mathcal{O}(\lvert\mathcal{D}_{public}\rvert * [l_{consensus}]^0)$
FD + FAug	$\mathcal{O}(\mathcal{D}_{GAN}+\theta_G)$ (FAug)	$\mathcal{O}(c * l_{class\_average})$
FedDTG	0	$\mathcal{O}(\theta_G+\theta_D+N([Z_{KD}]^0+[l^k]^0))$
FedGDKD	0	$\mathcal{O}(\theta_G+\lceil N/c\rceil[Z_{KD}]^0+N[l^k]^0)$

Table 5.13: Summary of communication costs For heterogeneous model FL algorithms

In Table 5.13 we have summarised the communication costs per algorithm (i.e. data sent between client and aggregation server) and broken it down into: initialisation costs (before communication rounds start) and the cost per communication round. Note that $|*|$ refers to a length operation and $[*]^x$ refers to an indexing operation in a matrix (rowwise) or vector at position x. These values are derived from the algorithms: FedMD (Algorithm 1), FD + FAug (Algorithm 2), FedDTG (Algorithm 3) and our proposed method - FedGDKD (Algorithm 4). Here, we provide a quick overview of the derivations of the terms used.

First, in the case of FedMD, the initial cost comes from the sharing of the proxy / public data set between clients. The cost per communication round comes from sending and receiving logits (only these logits are shared) from client to server and vice versa. The size of this logit matrix is determined by the size of the public dataset $|\mathcal{D}_{public}|$.

Second, in the case of FD + FAug, the initial cost is attributed to **FAug**. Here, clients will share a small subset of private data $D_{GAN} \subset D_k$ with a high-power server to train a generative network whose parameters θ_G are then shared with all clients. The cost per round of communication comes from sharing the average logits per label $l_{class\_average}$, the number of which depends on the number of classes c.

Then, analysing FedDTG, where there is no upfront cost for the client. The cost of communication per round comes in part from the use of a shared generator and discriminator whose parameters θ_G and θ_D are, in fact, updated twice during a communication round. The other part comes from aligning the client models through data-free distillation. This consists of first generating the distillation data set of size N, by sharing a noise vector Z_{KD} whose first dimension is of size N. Furthermore, the cost of sharing logit vectors l^k calculated on the distillation dataset (size N) to ensemble teacher logits, as well as receiving these teacher logits to perform knowledge distillation, must be taken into account. This is similar for our proposed method FedGDKD with the differences that it only shares the generator parameters θ_G and a smaller noise vector whose first dimension is of size $\lceil \frac{N}{c} \rceil$.

To compare each algorithm, let us analyse the additional overhead in a consistent setting where the client participates in T communication rounds. We can assume that the sizes of the logit vectors (for a single input) are the same $[l_{consensus}]^0 = l_{class\_average} = [l^k]^0$. Also, the task is the same and so the number of classes c is constant and equivalent. Additionally we can assume for now that identical generators are used for FD + FAug, FedDTG and FedGDKD. Then we can summarise the additional overhead incurred by the algorithm as

Algorithm	Initialisation	Communication Round	Overhead (T rounds)
FedMD	$\mathcal{O}(\mathcal{D}_{public})$	$\mathcal{O}(\lvert\mathcal{D}_{public}\rvert)$	$\mathcal{O}(\mathcal{D}_{public}+T\lvert\mathcal{D}_{public}\rvert)$
FD + FAug	$\mathcal{O}(\mathcal{D}_{GAN}+\theta_G)$ (FAug)	$\mathcal{O}(1)$	$\mathcal{O}(\mathcal{D}_{GAN}+\theta_G+T)$
FedGDKD	0	$\mathcal{O}(\theta_G+N[Z_{KD}]^0)$	$\mathcal{O}(T(\theta_G+N[Z_{KD}]^0))$
FedDTG	0	$\mathcal{O}(\theta_G+\theta_D+N[Z_{KD}]^0)$	$\mathcal{O}(T(\theta_G+\theta_D+N[Z_{KD}]^0))$

Table 5.14: Summary of additional communication overhead cost per client for T rounds ranked lowest to highest (under a consistent setting).

When comparing the costs noted in Table 5.14, one must consider the context of the problem. The

initialisation of FedMD and FD + FAug both involve transmitting entire data sets. Depending on the size of their entries, the cost could change dramatically, e.g. number vs. video file. Let us also assume that the size of the entry is proportional to the complexity of the problem, i.e. a larger dataset entry will need larger model capacities (more parameters).

Using the previous assumptions, FedDTG incurs the highest additional overhead. This is because the number of parameters in the generator and discriminator (millions) (shared in every communication round) probably outnumbers the length of the public dataset (hundreds/thousands) by a considerable margin, that is, $\theta_G >> |\mathcal{D}_{public}|$. The second highest is our proposed algorithm FedGDKD for the same reason, however, without the discriminator parameters. Next, $\mathcal{D}_{GAN}$ is smaller than $\mathcal{D}_{public}$, as the former refers to the data shared between clients and the server, and the latter the complete proxy data set. This suggests that unless the number of rounds T or the length of the public dataset $\mathcal{D}_{public}$ is very large $\theta_G > T|\mathcal{D}_{public}|$, FedMD has the lowest additional overhead cost, otherwise, FD + FAug.

In summary, we have shown that our proposed method, FedGDKD, is more communication efficient than the related data-free FedDTG. However, there is still a trade-off that one needs to consider when choosing a suitable algorithm. If the communication cost must be kept as low as possible, then FedMD should be chosen. If they cannot obtain a proxy data set but are willing to share some data then FD + FAug should be chosen. Finally, if there are (most likely) difficulties in acquiring a proxy dataset and restrictions on data sharing, then our proposed algorithm FedGDKD should be chosen.

Methods to further reduce the communication costs of the proposed algorithm will be considered in future work, but a few suggestions are as follows:

- Quantisation of the generator model.
- Compression methods.
- Reduce client participation per communication round.
- Reduce the size of the distillation dataset N.

5.5 Privacy

Privacy is the fundamental driver behind federated algorithms. Therefore in this section, we examine the privacy of the proposed method. This is not an exhaustive study over the vulnerabilties in federated learning e.g. [49], but examines the exposure introduced by the algorithms themselves.

5.5.1 Privacy Leakage

Privacy leakage refers to any vulnerabilities in the algorithm that can lead to access to private information. For the federated learning (FL) algorithms covered in this evaluation, we discuss these vulnerabilities and compare levels of privacy. We consider private information to be that related to the exposure of client private data as well for our use case the information regarding the client model C_k e.g. architecture, parameters, etc.

First, let us discuss the vulnerabilities that each federated learning algorithm exposes. We also include the evaluation of FedAvg [8], to compare against the most common FL algorithm. We have assumed that both the clients and aggregation server are honest-but-curious - follows the algorithm as outlined but also looks to learn all information legitimately - presenting a summary of findings in the following table:

Algorithm	Access (Client / Server)		
	Client Data	Model	
		White-box	Black-box
FedAvg	✗/✗	C/C_k	✗/✗
FedMD	?/?	✗/✗	✗/C_k
FD + FAug	✗/✓	G/G	✗/✗
FedDTG	✗/✗	G, D/G_k, D_k	✗/C_k
FedGDKD	✗/✗	G/G_k	✗/C_k

Table 5.15: Privacy: Summary of privacy leakage present in each federated learning algorithm.

where white-box refers to full access to a model (architecture, parameters, etc.) and black-box refers to the ability to query the model i.e. pass an input and receive an output. In terms of severity, access to client private data directly is the most severe for obvious reasons, following this is white-box access as this provides full access to a model and finally black-box which provides limited access. In particular, white-box access to the client models C_k is more severe than other models. The attacks that can be performed with each level of access can be found here [50]. Next, we give explanation to how Table 5.15 was produced.

Firstly, Federated Averaging (FedAvg) [8], which does not require sharing of any client private data. FedAvg uses parameter aggregation of client models C_k at the central server. This means that the server has white-box access to all of the client models C_k; the client has white-box access to the global model C which is the weighted average of client models. This is the only federated learning algorithm discussed that exposes the client model to white-box access.

Following this is FedMD (Section 3.1.1), which firstly requires a proxy dataset to carry out knowledge distillation. In the circumstance that the proxy dataset can be sourced from a similar domain without restriction then there is no leakage of client data. However, in the circumstance where no proxy dataset can be sourced, the clients will have to share some private data hence the "?" meaning depends. We have assumed the latter for preceding sections of evaluation. Additionally, the central server has access to the proxy dataset (as a means of distribution) and queries each client model C_k to ensemble teacher logits for knowledge distillation. This means that the server has black-box access to the client models. However, the aggregation of the teacher logits could also be insecure as the server knows the individual client logits so it can learn some information about the structure of the training data i.e. if it has greater representation of some classes over others etc.

Next, FD + FAug (Section 3.1.2), to initialise the algorithm each client shares private data to a high-powered server (which we assume to be the same as the aggregation server) to train a generative model G as part of FAug. This means that the server has access to the shared private data of the clients albeit partially. Once G is trained, it is distributed to the clients so both the server and clients have white-box access to it. Note, most attacks [50] work to uncover the private training data. So if the the proportion of shared training data is small so is the privacy leakage. Additionally, in FD + FAug, the clients share averages of class-wise logits which are then aggregated and redistributed at the server. One could argue that this does not strictly count as black-box access as neither the server nor client have knowledge about the data that was used to produce these logits and the averaging protects finer-grained information leakage. However, the aggregation of these could be insecure again through similar logic mentioned for FedMD.

Let us then discuss FedDTG (Section 3.2.1) which also does not require clients to share any private data. We can break down the algorithm into its main stages. First, GAN training, this stage requires that each client trains local copies of a shared generator G and discriminator D then sharing these back to the server. Therefore, the server has white-box access to the client copies G_k, D_k and the clients have white-box access to the shared copies G and D. Following this, a shared dataset for knowledge distillation is constructed and the server queries each client model C_k for its outputs. This suggests that as the server knows both the input and the output, it has black-box access to client models C_k. Following similar logic used in FedMD, the aggregation of logits could again be insecure allowing the server to uncover the structure of each client's training data.

Finally, our proposed method FedGDKD (Chapter 4), which again does not require clients to share any private data. Similarly to FedDTG, for GAN training the clients collaboratively train a global generator G sharing local copies G_k with the server - so white-box access. Additionally, again similar to FedDTG, a shared dataset is constructed for knowledge distillation by using G whereby the server queries client models C_k giving it black-box access. We also note the additional regularisation given by the novel cGAN training objectives (Section 4.2.1), this prevents the client models overfitting on the private data which can further reduce privacy leakage [51]. Finally, similar to FedMD the aggregation of logits could also cause some privacy leakage.

To compare the above, we focus on the severity of each vulnerability present (as discussed earlier). First, the most secure algorithm is FedMD when an applicable proxy dataset can be found without the need for clients to share private data. Following this as the next most secure is our proposed method FedGDKD, this is because of limited black-box access to client models and only white-box access to a shared generator. Then we have FedDTG for similar reasoning however, exposing an additional shared discriminator D which directly interacts with client data. After this is FD + FAug, where a small amount of client private data is shared in addition to a generator G trained on it but the client models are mostly protected. Then FedAvg, as although it does not expose any client data directly, white-box access to

the client models C_k is given. Finally the least secure is FedMD where a proxy dataset cannot be found and must be constructed from a proportion of client private data. The proxy dataset is shared amongst clients and so each client will have access to other's private data.

In summary, we have found that our proposed method FedGDKD, has higher privacy guarantees than most other federated learning algorithms present, further reinforcing its applicability. However, it still has vulnerabilities that can lead to privacy leakage of which we discuss solutions to in the next section.

5.5.2 Vulnerability Mitigation

In Section 5.5.1, we discussed the vulnerabilities present in our proposed algorithm FedGDKD that could lead to privacy leakage. In this section, we look at how we can combat this with implementation and investigation left as future work.

The two main vulnerabilities that FedGDKD has is the shared generator model G which approximates the joint distribution of the collective client private data. The second vulnerability is the information leakage that could be present in sharing logits calculated on the distillation set using client models C_k. Next we offer suggestions for further research as to how to tackle them.

Local Differential Privacy

Differential privacy (DP) has been used extensively in deep learning and in analytics to preserve privacy of individuals in datasets [52, 53]. Therefore, we can use differential privacy local to each client in two places: when training the local copy of the generator model G_k and when returning the logits on the distillation dataset.

Using ideas from Differentially Private GANs (DPGAN) [54] and its federated implementation [55], we can kill two birds in one stone by training the client model C_k in a differentially private manner i.e. adding noise to gradients and clipping updates. Due to the post-processing property of DP, where a function on something that is differentially private is also differentially private, the generator G and logits will be differentially private too. This is because the generator G does not directly interact with client private data but interacts with outputs of C_k which has. Additionally, the logits of C_k will also be inherently differentially private for the same reason.

However, there is a privacy-utility trade-off that comes into play when using DP. This is where in place of privacy the performance of the algorithm will be negatively affected. Therefore, directly applying DP to the client model C_k will worsen its performance. Could there be a way to reduce this effect while maintaining the same privacy guarantees? Potentially, we could apply DP directly to the shared generator and to the logits instead of to the client model. This means that C_k will not be hampered by DP and may perform better. Unfortunately, this can add further complexity as DP occurs separately in two places now.

Secure Aggregation

Alternatively, we can employ secure aggregation [56] at the central server which will hide individual contributions of each client's generator parameters or knowledge distillation logits. This can be used to prevent the server from having black-box access to client models and increase privacy without the expense of utility. This would be used to aggregate the client copies of the generator G_k and for aggregating client logits for knowledge distillation.

However, while protecting individual contributions and updates of clients, secure aggregation will not alone protect against the privacy leakage from white-box access to the generator G. Therefore, a solution could be to use secure aggregation with DP on just the generator. This, in theory, could also maintain utility better than just local DP on the client model C_k.

Overall Summary

In summary we have presented three different approaches on mitigating privacy leakage in our proposed algorithm FedGDKD. The evaluation of such improvements is left for further research but indeed provides interesting discussion.

5.6 Summary

In summary, in conjunction with the problem statement and challenges set out in Section 4.1, we have explored the various aspects of our proposed method FedGDKD compared to the state-of-the-art. In particular, we can give an overall score to each heterogeneous model federated learning algorithm based on their relative rankings of performance (Section 5.2), efficiency (Section 5.4), privacy (Section 5.5) and whether they are data-based (0 for no; 1 for yes). The lower the overall total, the better the overall algorithm.

Algorithm	Data-based	Performance	Efficiency	Privacy	Total
FedMD	1	1(2*)	1	4	7(8*)
FD + FAug	1	4	2	3	10
FedDTG	0	3	4	2	9
FedGDKD	0	2(1*)	3	1	**6(5*)**

Table 5.16: Ranking algorithms over evaluation areas.

In this project, we have assumed that a proxy dataset is impossible to assemble (clients share 5% of their data) and so for FedMD we have accordingly ranked it in Table 5.16. The performance rankings were purely based on final top-1 test accuracies reached, whereas, if improvement over baselines is considered, as denoted by the "*", then FedGDKD would be ranked first and FedMD second (leading to an overall score of 5 and 8, respectively). In either case however, the results show that overall, the proposed method FedGDKD is the best when compared to the state-of-the-art when meeting the solution criteria. Additionally, we have shown that the novel GAN training method present in FedGDKD proves to also be more robust than the state-of-the-art federated GAN algorithm. However, this evaluation has also highlighted areas of improvement as well as further research.

Chapter 6

Conclusions and Future Work

In this project, we have presented a novel algorithm for tackling the problem of heterogeneous model federated learning. A summary of our contributions is as follows:

- We introduced *Federated GAN-based Data-Free Knowledge Distillation* (FedGDKD), a novel federated learning algorithm that allows clients to collaborate effectively while protecting the architecture and parameters of their own model. The algorithm is made up of three stages: local GAN training, generator aggregation and data-free knowledge distillation which are all necessary for best performance. We evaluated our algorithm across both the MNIST and EMNIST datasets with various settings. This showed that FedGDKD performs well under various amounts of data heterogeneity (degree of non-iid) and even shows higher robustness to these settings than state-of-the-art methods; is more efficient than other state-of-the-art heterogeneous model federated learning algorithms using data-free knowledge distillation; is more private than other state-of-the-art heterogeneous model federated learning algorithms under.
- In the process of developing FedGDKD, we developed a novel cGAN loss formulation for training conditional generators that removes the use of a discriminator task. This allows clients to not have to modify their private models to use FedGDKD.
- Continuing with the previous contribution, FedGDKD also introduces a novel federated cGAN training method, whereby only the generator must be shared and the discriminator's role is absorbed into the client's private classifier (can be heterogeneous in architecture too). We evaluated this against the state-of-the-art method FedGAN and found it in fact generally performs better, is more private and is more communication efficient.

Additionally, from evaluating our method we had found areas of improvement and that of further work that could be done:

Theoretical Convergence Analysis In this project we empirically evaluated the convergence of FedGDKD under various scenarios. For more rigorous evaluation, a theoretical convergence analysis should be performed obtaining an upper bound.

Evaluation Under More Complex Datasets In this project we focussed on simple image-classification datasets: MNIST and EMNIST. Further evaluation should be done to see the performance on image datasets that have more complex features e.g. CIFAR10[57] or ImageNet[58].

Evaluation Under More Random Seeds In this project due to limitations in time, we were only able to perform a single run for some algorithms over a single seed per experiment. This is because of the runtime being quite large (up to 9 hours). Therefore, to test further and to further isolate performance from a lucky/unlucky random seed, we suggest running more experiments over multiple seeds to find the a better approximation of performance.

Evaluate FedMD With Suitable Proxy Datasets In this project we assumed that the proxy dataset was impossible to assemble for the FedMD algorithm (Section 3.1.1). It would be interesting to explore the case where one could be assembled. This could be done by perturbing or applying transformations on the dataset to different degrees to simulate covariate shift and elicit a difference in domain. We can then

see the effect on performance this has, which we hypothesise will make FedMD perform worse.

Extension To Other Tasks As current methods in knowledge distillation Section 2.3 mainly focus on classification tasks, it would be interesting to see extensions to other tasks such as regression.

Knowledge Distillation Weight Scheduling In our evaluation of the hyperparameters related to knowledge distillation (Section 5.2.4), we had found evidence that having a schedule for the weight parameter α_{KD} could improve overall performance of FedGDKD. Therefore, different schedules should be explored to see what works best.

GAN Training Our novel GAN training objectives were evaluated to perform better than the state-of-the-art (Section 5.2.5). However, further research can be done to improve the diversity of samples generated. As this can be used to further improve the efficiency of knowledge distillation. This was discussed in the study when varying the distillation dataset size N Section 5.2.4.

Communication Efficiency Enhancement When evaluating the communication efficiency of our proposed method (Section 5.4.2), we had found that FedGDKD has a relatively large cost. Therefore, we proposed methods of reducing these communication costs through methods such as: model quantisation, compression and modifying hyperparameters. Other such methods should be implemented and evaluated in future work.

Privacy Enhancement In the privacy evaluation of our proposed method (Section 5.5), we had found potential vulnerabilities that could lead to privacy leakage. We additionally propose solutions to mitigate this vulnerabilities however further work should be done to implement and evaluate such improvements.

We also consider that the constituent methods and techniques used in FedGDKD may have orthogonal enhancements that complement well. One such example is more robust parameter aggregation like used in FedProx[59] or SCAFFOLD[60]. It is also worth stating that any individual method or technique used in FedGDKD can be switched out for one that is more effective, such as a better version of response-based knowledge distilliaton. Any of these changes could positively impact the overall performance of our proposed method.

Finally, to finish, this project proved difficult at times due to the expansive fields of Federated Learning, GANs and Knowledge Distillation. However, we feel that the final method presented is one that tackles the problem of heterogeneous model federated learning effectively and hope that it opens further questions into the field.

Bibliography

[1] Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, et al. Advances and open problems in federated learning. arXiv preprint arXiv:191204977. 2019. Available from: https://arxiv.org/abs/1912.04977.

[2] Gonzalo Medina TT. Diagram of an artificial neuron; 2017. Last accessed 07 January 2022. Available from: https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network.

[3] Zhou Y, Song S, Cheung NM. On classification of distorted images with deep convolutional neural networks. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2017. p. 1213-7. Available from: https://ieeexplore.ieee.org/document/7952349.

[4] Overview of GAN Structure; Generative Adversarial Networks; Google Developers. Google;. Available from: https://developers.google.com/machine-learning/gan/gan_structure.

[5] Odena A, Olah C, Shlens J. Conditional image synthesis with auxiliary classifier gans. In: International conference on machine learning. PMLR; 2017. p. 2642-51. Available from: https://arxiv.org/abs/1610.09585v3.

[6] Jumper J, Evans R, Pritzel A, Green T, Figurnov M, Ronneberger O, et al. Highly accurate protein structure prediction with AlphaFold. Nature. 2021;596(7873):583-9. Available from: https://www.nature.com/articles/s41586-021-03819-2.

[7] Al-Rubaie, Mohammad and Chang, J Morris. Privacy-Preserving Machine Learning: Threats and Solutions. IEEE Security Privacy. 2019;17(2):49-58. Available from: https://ieeexplore.ieee.org/abstract/document/8677282.

[8] Brendan McMahan H, Moore E, Ramage D, Hampson S, Agüera y Arcas B. Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. PMLR; 2017. p. 1273-82. Available from: http://proceedings.mlr.press/v54/mcmahan17a.

[9] Hinton G, Vinyals O, Dean J. Distilling the Knowledge in a Neural Network. In: NIPS Deep Learning and Representation Learning Workshop; 2015. Available from: http://arxiv.org/abs/1503.02531.

[10] Mireshghallah F, Taram M, Vepakomma P, Singh A, Raskar R, Esmaeilzadeh H. Privacy in deep learning: A survey. arXiv preprint arXiv:200412254. 2020. Available from: https://arxiv.org/abs/2004.12254.

[11] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436-44. Available from: https://www.nature.com/articles/nature14539.

[12] Stutz D. Illustrating (Convolutional) Neural Networks in LaTeX with TikZ; 2020. Last accessed 10 January 2022. Available from: https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/.

[13] Weisstein EW. Convolution.. MathWorld–A Wolfram Web Resource.;. Available from: https://mathworld.wolfram.com/Convolution.html.

[14] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative Adversarial Nets. In: Ghahramani Z, Welling M, Cortes C, Lawrence N, Weinberger KQ, editors. Advances in Neural Information Processing Systems. vol. 27. Curran Associates, Inc.; 2014. Available from: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[15] Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science; 1985. Available from: https://ieeexplore.ieee.org/document/6302929.

[16] Nishio T, Yonetani R. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC); 2019. p. 1-7. Available from: https://arxiv.org/abs/1804.08333.

[17] Zhu H, Xu J, Liu S, Jin Y. Federated Learning on Non-IID Data: A Survey. arXiv preprint arXiv:210606843. 2021. Available from: https://arxiv.org/abs/2106.06843.

[18] Tan AZ, Yu H, Cui L, Yang Q. Towards personalized federated learning. arXiv preprint arXiv:210300710. 2021. Available from: https://arxiv.org/abs/2103.00710.

[19] Gou J, Yu B, Maybank SJ, Tao D. Knowledge distillation: A survey. International Journal of Computer Vision. 2021;129(6):1789-819. Available from: https://link.springer.com/article/10.1007/s11263-021-01453-z.

[20] Lee SH, Kim DH, Song BC. Self-supervised knowledge distillation using singular value decomposition. In: Proceedings of the European Conference on Computer Vision (ECCV); 2018. p. 335-50. Available from: https://arxiv.org/abs/1807.06819.

[21] Anil R, Pereyra G, Passos A, Ormandi R, Dahl GE, Hinton GE. Large scale distributed neural network training through online distillation. arXiv preprint arXiv:180403235. 2018. Available from: https://arxiv.org/abs/1804.03235.

[22] Zhang L, Song J, Gao A, Chen J, Bao C, Ma K. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2019. p. 3713-22. Available from: https://arxiv.org/abs/1905.08094.

[23] Stanton S, Izmailov P, Kirichenko P, Alemi AA, Wilson AG. Does Knowledge Distillation Really Work? arXiv preprint arXiv:210605945. 2021. Available from: https://arxiv.org/abs/2106.05945.

[24] Li D, Wang J. FedMD: Heterogenous federated learning via model distillation. arXiv preprint arXiv:191003581. 2019. Available from: https://arxiv.org/abs/1910.03581.

[25] Jeong E, Oh S, Kim H, Park J, Bennis M, Kim SL. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. arXiv preprint arXiv:181111479. 2018. Available from: https://arxiv.org/abs/2103.00710.

[26] Zhang Z. FedDTG: Federated Data-Free Knowledge Distillation via Three-Player Generative Adversarial Networks. arXiv preprint arXiv:220103169. 2022. Available from: https://arxiv.org/pdf/2201.03169.

[27] Odena A, Olah C, Shlens J. Conditional image synthesis with auxiliary classifier gans. In: International conference on machine learning. PMLR; 2017. p. 2642-51. Available from: https://arxiv.org/abs/1610.09585.

[28] Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X. Improved techniques for training GANs. Advances in neural information processing systems. 2016;29. Available from: https://arxiv.org/pdf/1606.03498.

[29] Odena A. Semi-supervised learning with generative adversarial networks. arXiv preprint arXiv:160601583. 2016. Available from: https://arxiv.org/abs/1606.01583.

[30] torch.logsumexp. PyTorch;. Available from: https://pytorch.org/docs/stable/generated/torch.logsumexp.html.

[31] softplus. PyTorch;. Available from: https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html.

[32] OpenAI. Code for the paper "Improved Techniques for Training GANs"; 2018. [Online; accessed 10-June-2022]. https://github.com/openai/improved-gan.

[33] Common Problems; Generative Adversarial Networks; Google Developers. Google;. Available from: https://developers.google.com/machine-learning/gan/problems.

[34] Petrova O. Semi-Supervised Learning with GANs: a Tale of Cats and Dogs; 2020. [Online; accessed 10-June-2022]. https://blog.scaleway.com/semi-supervised/.

[35] Rasouli M, Sun T, Rajagopal R. FedGAN: Federated generative adversarial networks for distributed data. arXiv preprint arXiv:200607228. 2020. Available from: https://arxiv.org/pdf/2006.07228.

[36] Guerraoui R, Guirguis A, Kermarrec AM, Merrer EL. FeGAN: Scaling Distributed GANs. Proceedings of the 21st International Middleware Conference. 2020. Available from: https://dl.acm.org/doi/pdf/10.1145/3423211.3425688.

[37] Fan, Chenyou and Liu, Ping. Federated generative adversarial learning. In: Chinese Conference on Pattern Recognition and Computer Vision (PRCV). Springer; 2020. p. 3-15. Available from: https://arxiv.org/pdf/2005.03793.

[38] Hardy C, Le Merrer E, Sericola B. MD-GAN: Multi-Discriminator Generative Adversarial Networks for Distributed Datasets. In: 2019 IEEE international parallel and distributed processing symposium (IPDPS). IEEE; 2019. p. 866-77. Available from: https://ieeexplore.ieee.org/iel7/8804711/8820774/08821025.

[39] Yonetani R, Takahashi T, Hashimoto A, Ushiku Y. Decentralized learning of generative adversarial networks from non-iid data. arXiv preprint arXiv:190509684. 2019. Available from: https://arxiv.org/pdf/1905.09684.

[40] He C, Li S, So J, Zhang M, Wang H, Wang X, et al. FedML: A Research Library and Benchmark for Federated Machine Learning. Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop. 2020. Available from: https://arxiv.org/abs/2007.13518.

[41] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R, editors. Advances in Neural Information Processing Systems 32. Curran Associates, Inc.; 2019. p. 8024-35. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[42] Biewald L. Experiment Tracking with Weights and Biases; 2020. Software available from wandb.com. Available from: https://www.wandb.com/.

[43] LeCun Y, Cortes C, Burges C. MNIST handwritten digit database. ATT Labs [Online]. 2010;2. Available from: http://yann.lecun.com/exdb/mnist.

[44] Cohen G, Afshar S, Tapson J, Schaik AV. EMNIST: Extending MNIST to handwritten letters. 2017 International Joint Conference on Neural Networks (IJCNN). 2017.

[45] Kotz S, Balakrishnan N, Johnson NL. 49. In: Continuous multivariate distributions, Volume 1: Models and applications. vol. 1. John Wiley & Sons; 2004. .

[46] Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, et al., editors. Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc.; 2017. Available from: https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.

[47] Wang S, Tuor T, Salonidis T, Leung KK, Makaya C, He T, et al. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. IEEE Journal on Selected Areas in Communications. 2019;37(6):1205-21. Available from: https://ieeexplore.ieee.org/document/8664630/.

[48] Knuth DE. Big omicron and big omega and big theta. ACM Sigact News. 1976;8(2):18-24.

[49] Bouacida N, Mohapatra P. Vulnerabilities in Federated Learning. IEEE Access. 2021;9:63229-49. Available from: https://ieeexplore.ieee.org/document/9411833.

[50] Rigaki M, Garcia S. A survey of privacy attacks in machine learning. arXiv preprint arXiv:200707646. 2020. Available from: https://arxiv.org/abs/2007.07646.

[51] Yeom S, Giacomelli I, Fredrikson M, Jha S. Privacy risk in machine learning: Analyzing the connection to overfitting. In: 2018 IEEE 31st computer security foundations symposium (CSF). IEEE; 2018. p. 268-82.

[52] Dwork C. Differential Privacy: A Survey of Results. In: Agrawal M, Du D, Duan Z, Li A, editors. Theory and Applications of Models of Computation. Berlin, Heidelberg: Springer Berlin Heidelberg; 2008. p. 1-19. Available from: https://link.springer.com/chapter/10.1007/978-3-540-79228-4_1.

[53] Abadi M, Chu A, Goodfellow I, McMahan B, Mironov I, Talwar K, et al. Deep Learning with Differential Privacy. In: 23rd ACM Conference on Computer and Communications Security (ACM CCS); 2016. p. 308-18. Available from: https://arxiv.org/abs/1607.00133.

[54] Xie L, Lin K, Wang S, Wang F, Zhou J. Differentially private generative adversarial network. arXiv preprint arXiv:180206739. 2018. Available from: https://arxiv.org/abs/1802.06739.

[55] Zhang L, Shen B, Barnawi A, Xi S, Kumar N, Wu Y. FedDPGAN: federated differentially private generative adversarial networks framework for the detection of COVID-19 pneumonia. Information Systems Frontiers. 2021;23(6):1403-15. Available from: https://link.springer.com/article/10.1007/s10796-021-10144-6.

[56] Bonawitz K, Ivanov V, Kreuter B, Marcedone A, McMahan HB, Patel S, et al. Practical secure aggregation for privacy-preserving machine learning. In: proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security; 2017. p. 1175-91. Available from: https://dl.acm.org/doi/abs/10.1145/3133956.3133982.

[57] Krizhevsky A, Nair V, Hinton G. Learning multiple layers of features from tiny images. 2009. Available from: http://www.cs.toronto.edu/~kriz/cifar.html.

[58] Recht B, Roelofs R, Schmidt L, Shankar V. Do ImageNet Classifiers Generalize to ImageNet? In: International Conference on Machine Learning; 2019. p. 5389-400. Available from: http://proceedings.mlr.press/v97/recht19a/recht19a.pdf.

[59] Li T, Sahu AK, Zaheer M, Sanjabi M, Talwalkar A, Smith V. Federated Optimization in Heterogeneous Networks. In: Dhillon I, Papailiopoulos D, Sze V, editors. Proceedings of Machine Learning and Systems. vol. 2; 2020. p. 429-50. Available from: https://proceedings.mlsys.org/paper/2020/file/38af86134b65d0f10fe33d30dd76442e-Paper.pdf.

[60] Karimireddy SP, Kale S, Mohri M, Reddi S, Stich S, Suresh AT. SCAFFOLD: Stochastic controlled averaging for federated learning. In: International Conference on Machine Learning. PMLR; 2020. p. 5132-43. Available from: https://proceedings.mlr.press/v119/karimireddy20a.html.

Appendix A

Additional Background Information

	Datacenter distributed learning	**Cross-silo federated learning**	**Cross-device federated learning**
Setting	Training a model on a large but "flat" dataset. Clients are compute nodes in a single cluster or datacenter.	Training a model on siloed data. Clients are different organizations (e.g. medical or financial) or geo-distributed datacenters.	The clients are a very large number of mobile or IoT devices.
Data distribution	Data is centrally stored and can be shuffled and balanced across clients. Any client can read any part of the dataset.	**Data is generated locally and remains decentralized.** Each client stores its own data and cannot read the data of other clients. Data is not independently or identically distributed.	
Orchestration	Centrally orchestrated.	**A central orchestration server/service organizes the training,** but never sees raw data.	
Wide-area communication	None (fully connected clients in one datacenter/cluster).	Typically a hub-and-spoke topology, with the hub representing a coordinating service provider (typically without data) and the spokes connecting to clients.	
Data availability	——— All clients in one datacenter/cluster ———		Only a fraction of clients are available at any one time, often with diurnal or other variations.
Distribution scale	Typically 1 - 1000 clients.	Typically 2 - 100 clients.	Massively parallel, up to 10^{10} clients.
Primary bottleneck	Computation is more often the bottleneck in the datacenter, where very fast networks can be assumed.	Might be computation or communication.	Communication is often the primary bottleneck, though it depends on the task. Generally, cross-device federated computations use wi-fi or slower connections.
Addressability	Each client has an identity or name that allows the system to access it specifically.		Clients cannot be indexed directly (i.e., no use of client identifiers).
Client statefulness	Stateful—each client may participate in each round of the computation, carrying state from round to round.		Stateless — each client will likely participate only once in a task, so generally a fresh sample of never-before-seen clients in each round of computation is assumed.
Client reliability	——— Relatively few failures ———		Highly unreliable — 5% or more of the clients participating in a round of computation are expected to fail or drop out (e.g. because the device becomes ineligible when battery, network, or idleness requirements are violated).
Data partition axis	Data can be partitioned / repartitioned arbitrarily across clients.	Partition is fixed. Could be example-partitioned (horizontal) or feature-partitioned (vertical).	Fixed partitioning by example (horizontal).

Table A.1: Typical characteristics of federated learning settings vs. distributed learning in the datacenter. [1]

Appendix B

Evaluation

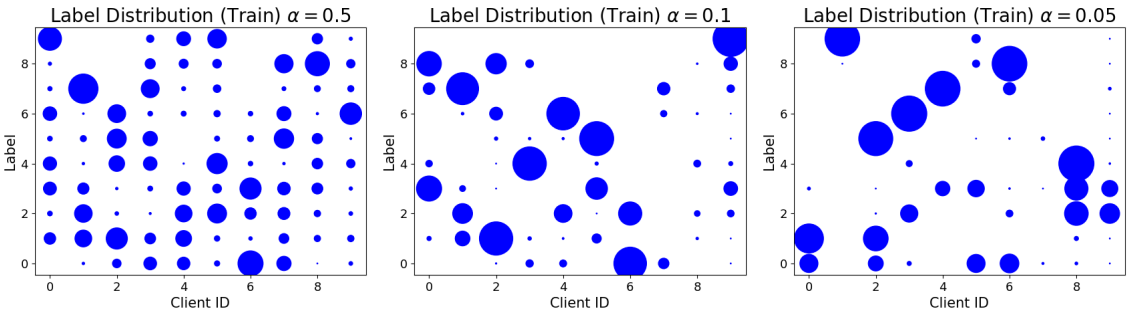
B.1 Random Seeds

We picked seeds based on the performance of FedAvg from the range 0-4 to use for each case and fixed it for each experiment.

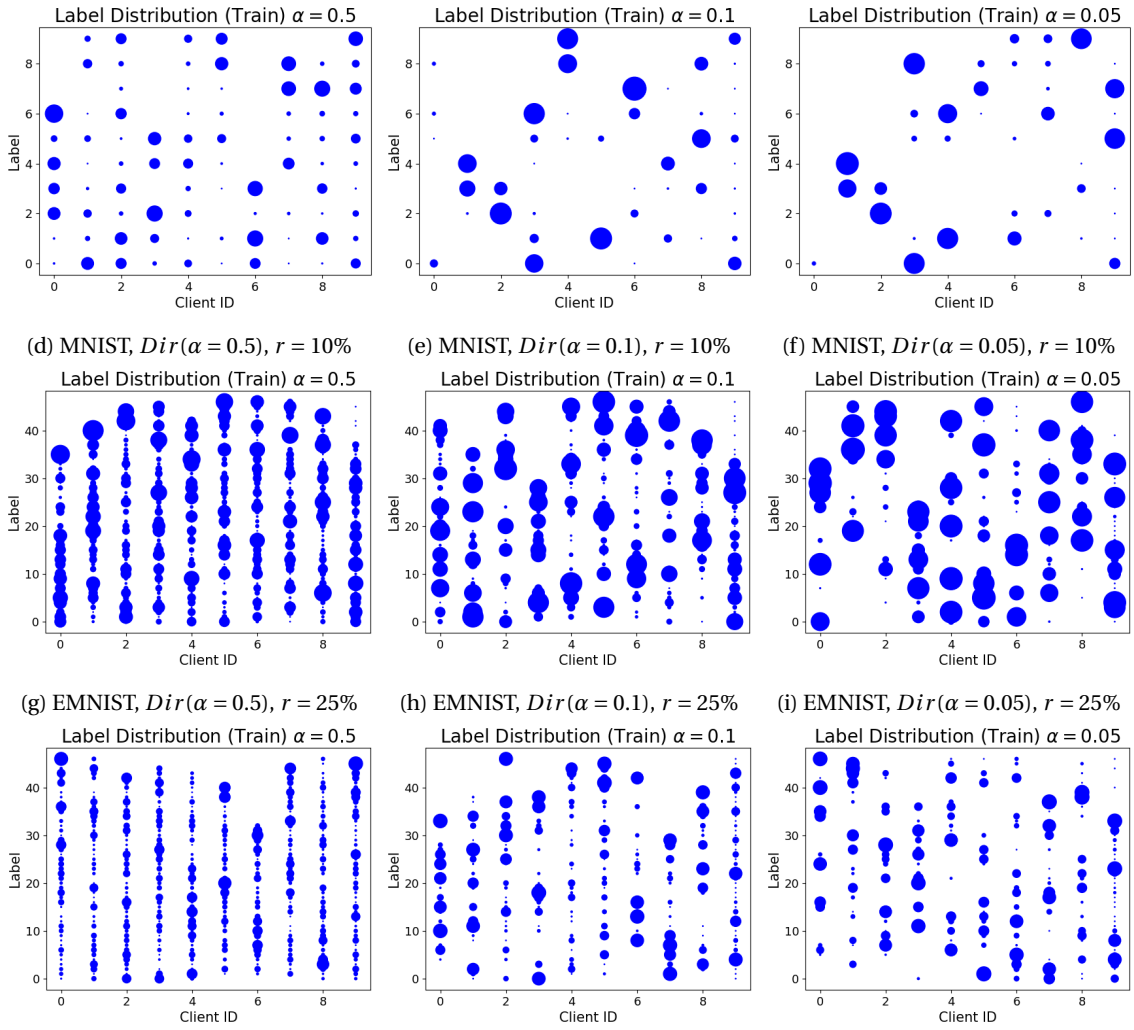
Random Seeds		
Dataset	Setting	Seed
MNIST $r = 25\%$	$\alpha = 0.5$	4
	$\alpha = 0.1$	4
	$\alpha = 0.05$	2
MNIST $r = 10\%$	$\alpha = 0.5$	3
	$\alpha = 0.1$	0
	$\alpha = 0.05$	0
EMNIST $r = 25\%$	$\alpha = 0.5$	1
	$\alpha = 0.1$	4
	$\alpha = 0.05$	1
EMNIST $r = 10\%$	$\alpha = 0.5$	4
	$\alpha = 0.1$	1
	$\alpha = 0.05$	3

Table B.1: Experiment Random Seeds Used

B.2 Client Data Heterogeneity



(a) MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$ (b) MNIST, $Dir(\alpha = 0.1)$, $r = 25\%$ (c) MNIST, $Dir(\alpha = 0.05)$, $r = 25\%$



(d) MNIST, $Dir(\alpha = 0.5)$, $r = 10\%$ (e) MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$ (f) MNIST, $Dir(\alpha = 0.05)$, $r = 10\%$

(g) EMNIST, $Dir(\alpha = 0.5)$, $r = 25\%$ (h) EMNIST, $Dir(\alpha = 0.1)$, $r = 25\%$ (i) EMNIST, $Dir(\alpha = 0.05)$, $r = 25\%$

(j) EMNIST, $Dir(\alpha = 0.5)$, $r = 10\%$ (k) EMNIST, $Dir(\alpha = 0.1)$, $r = 10\%$ (l) EMNIST, $Dir(\alpha = 0.05)$, $r = 10\%$

Figure B.1: Client training label distributions (x-axis: client ID; y-axis: label; size of point: frequency of label in client's training data)

B.3 Model Architectures

For both MNIST and EMNIST, we transformed the images to size 32 x 32. Allowing us to use the same architectures for both.

B.3.1 Generator Model Architecture

Pytorch:

```
ConditionalImageGenerator(
  (label_emb): Embedding(10, 100)
  (l1): Sequential(
    (0): Linear(in_features=100, out_features=6400, bias=True)
  )
  (main): Sequential(
    (block 0): Sequential(
      (0): ConvTranspose2d(400, 200, kernel_size=(4, 4),
           stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(200, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
```

```
    )
    (block 1): Sequential(
      (0): ConvTranspose2d(200, 100, kernel_size=(4, 4),
           stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(100, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (end): Sequential(
      (0): ConvTranspose2d(100, 1, kernel_size=(4, 4),
           stride=(2, 2), padding=(1, 1), bias=False)
      (1): Tanh()
    )
  )
)
```

B.3.2 ACGAN Discriminator Architecture

Pytorch:

```
(CNNParameterised(
  (net): Sequential(
    (layer_0): Sequential(
      (0): Conv2d(1, 8, kernel_size=(3, 3),
           stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(8, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=False)
      (2): ReLU(inplace=True)
    )
    (layer_1): Sequential(
      (0): Conv2d(8, 16, kernel_size=(3, 3),
           stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(16, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=False)
      (2): ReLU(inplace=True)
    )
    (layer_2): Sequential(
      (0): Conv2d(16, 16, kernel_size=(3, 3),
           stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(16, eps=1e-05, momentum=0.1,
           affine=True, track_running_stats=False)
      (2): ReLU(inplace=True)
    )
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=256, out_features=128, bias=True)
    (2): Linear(in_features=128, out_features=10, bias=True)
  )
  (discriminator): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=256, out_features=128, bias=True)
    (2): Linear(in_features=128, out_features=1, bias=True)
    (3): Sigmoid()
  )
```

B.4 Active-User Ratio

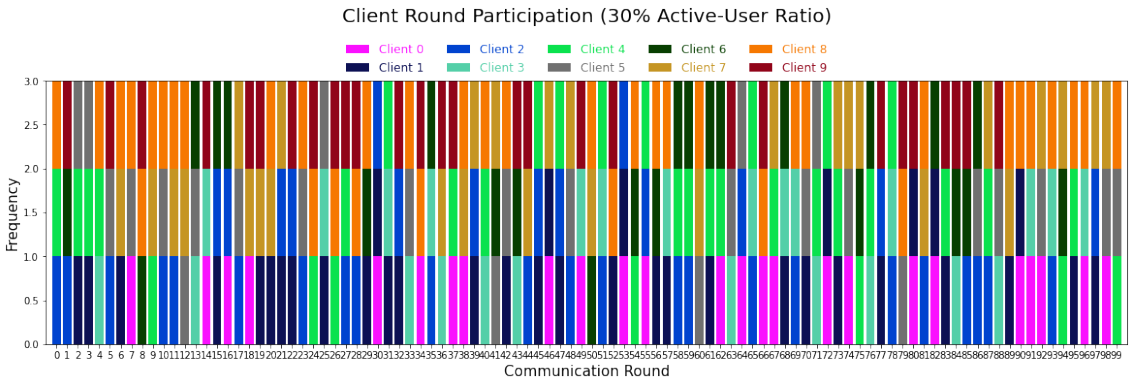
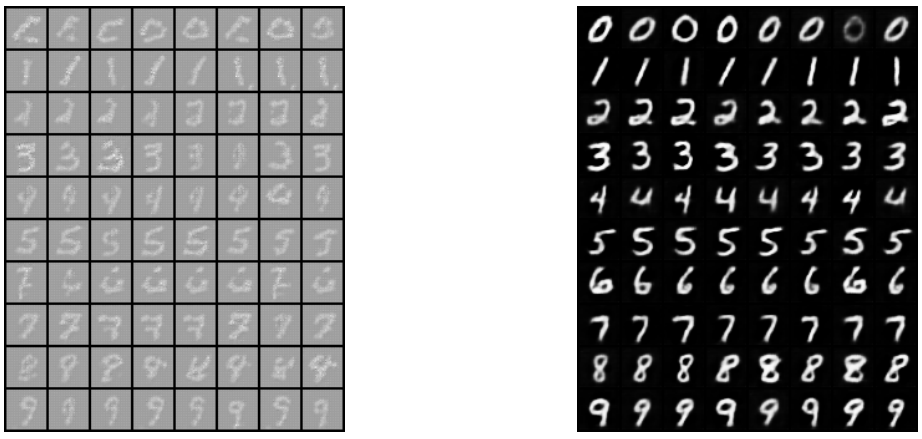


Figure B.2: Client participation per round with active-user ratio 30% (MNIST, Dir($\alpha = 0.5$), $r = 25\%$)

B.5 Generator Performance: Generator Output Comparison

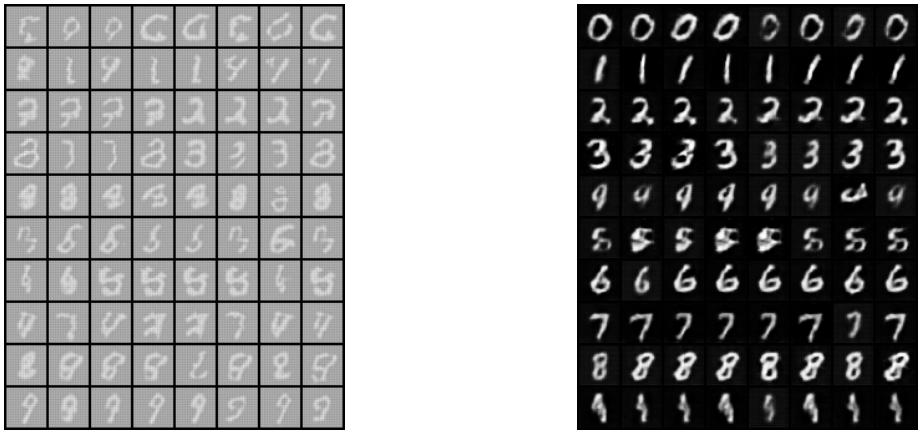
B.5.1 MNIST

Figure B.3: MNIST Ground Truth



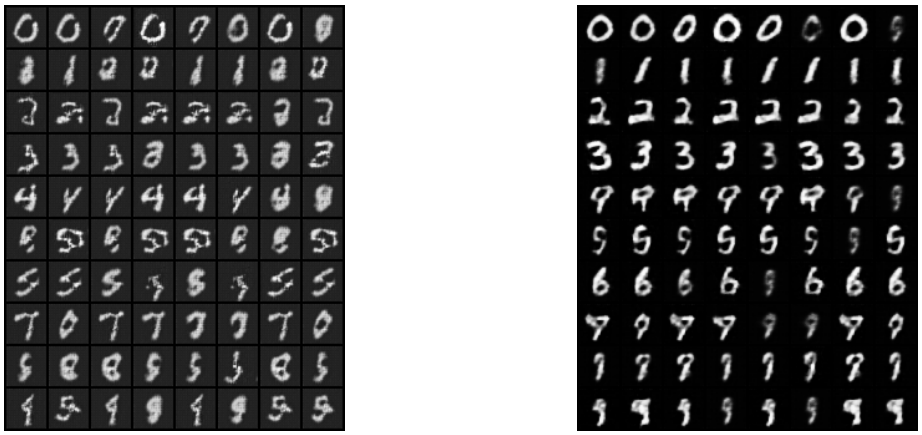
(a) FedGAN (b) FedGDKD

Figure B.4: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)



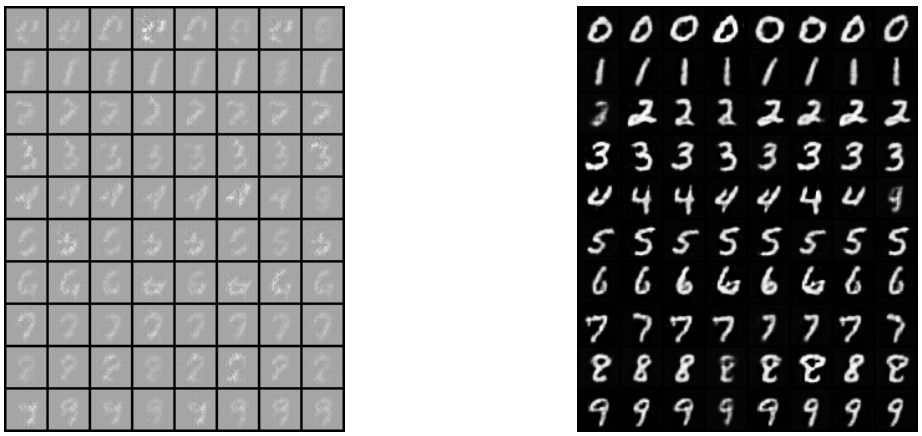
(a) FedGAN (b) FedGDKD

Figure B.5: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.1)$, $r = 25\%$)



(a) FedGAN (b) FedGDKD

Figure B.6: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.05)$, $r = 25\%$)



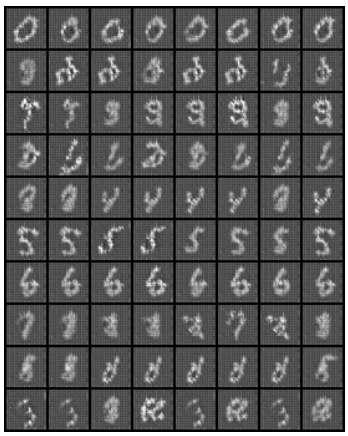
(a) FedGAN (b) FedGDKD

Figure B.7: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.5)$, $r = 10\%$)

(a) FedGAN

(b) FedGDKD

Figure B.8: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.1)$, $r = 10\%$)



(a) FedGAN

(b) FedGDKD

Figure B.9: Generator Performance: Generator final output (MNIST, $Dir(\alpha = 0.05)$, $r = 10\%$)

B.5.2 EMNIST

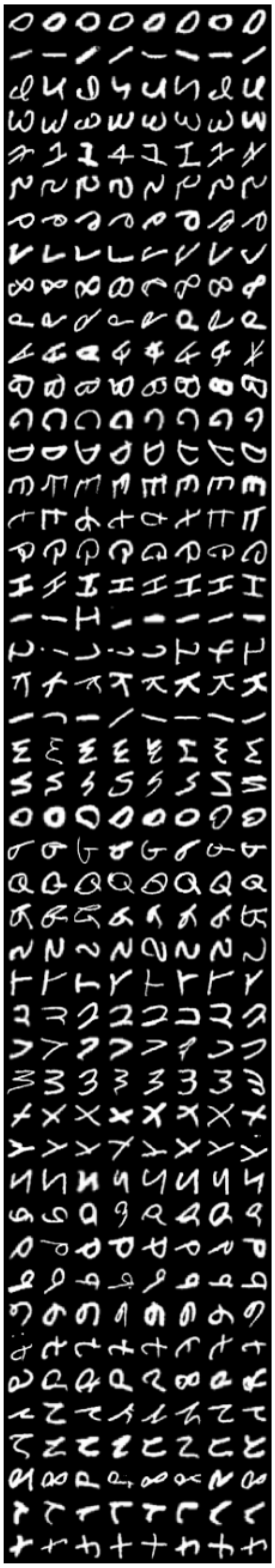
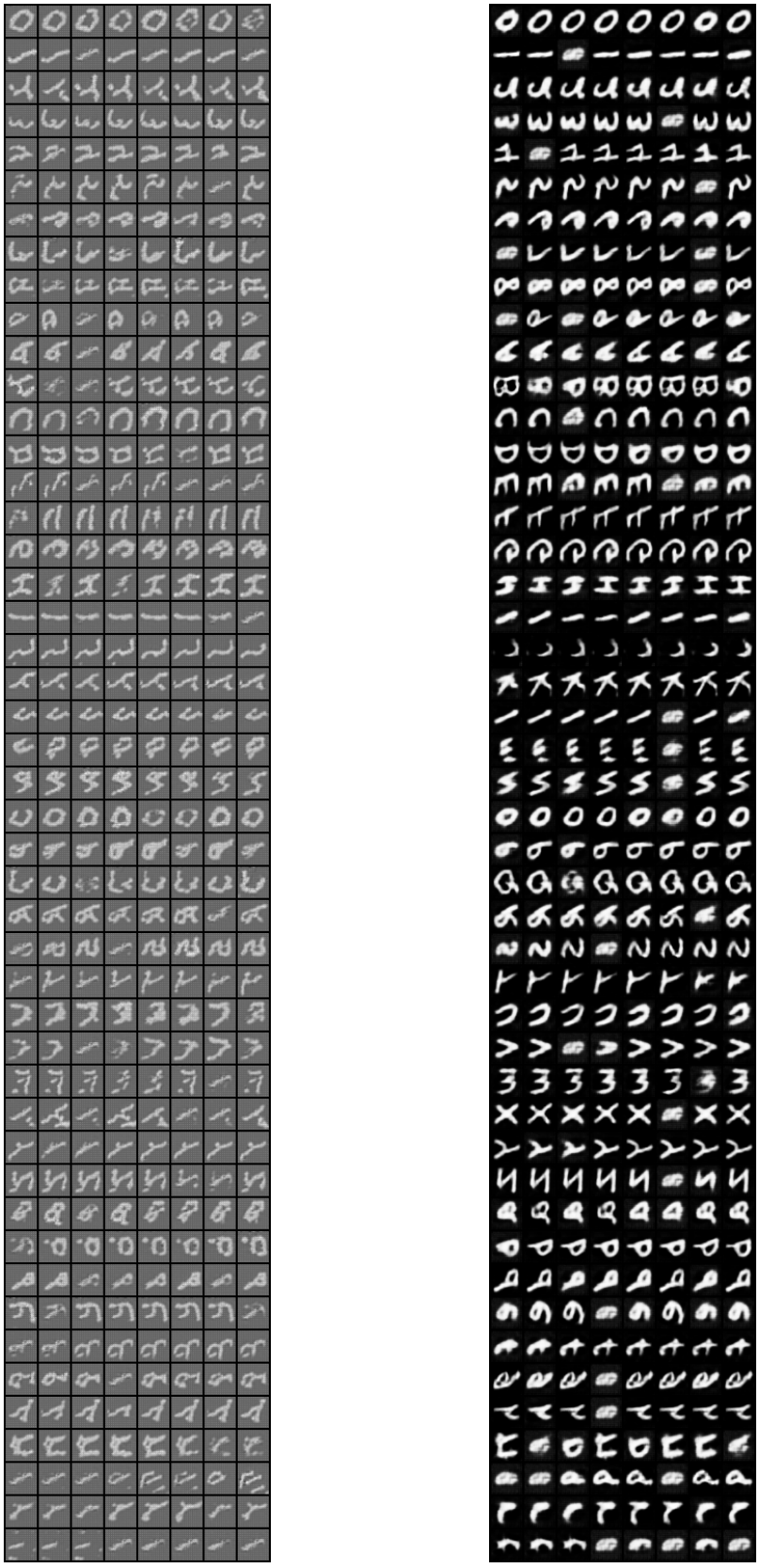
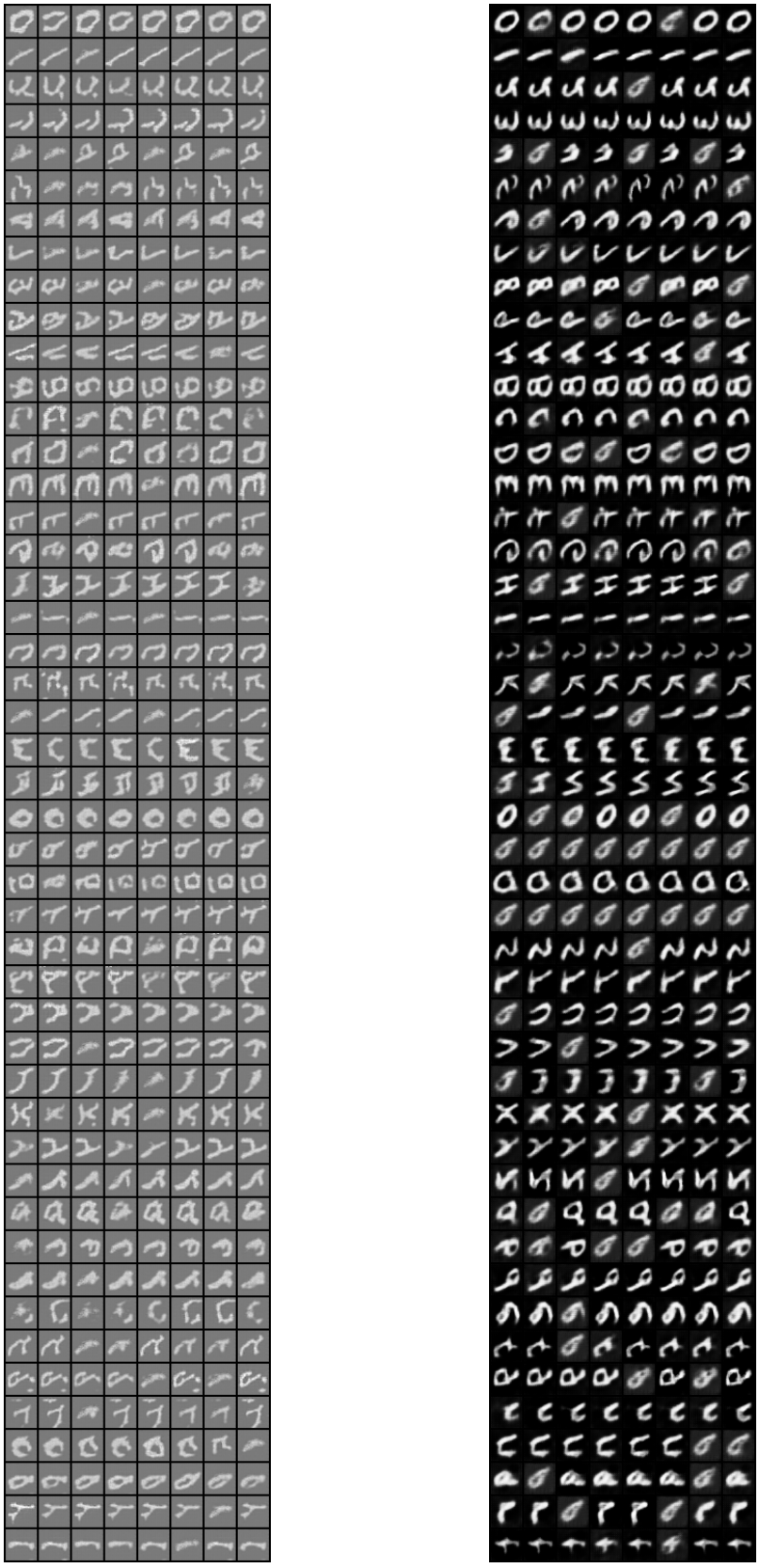


Figure B.10: EMNIST Ground Truth



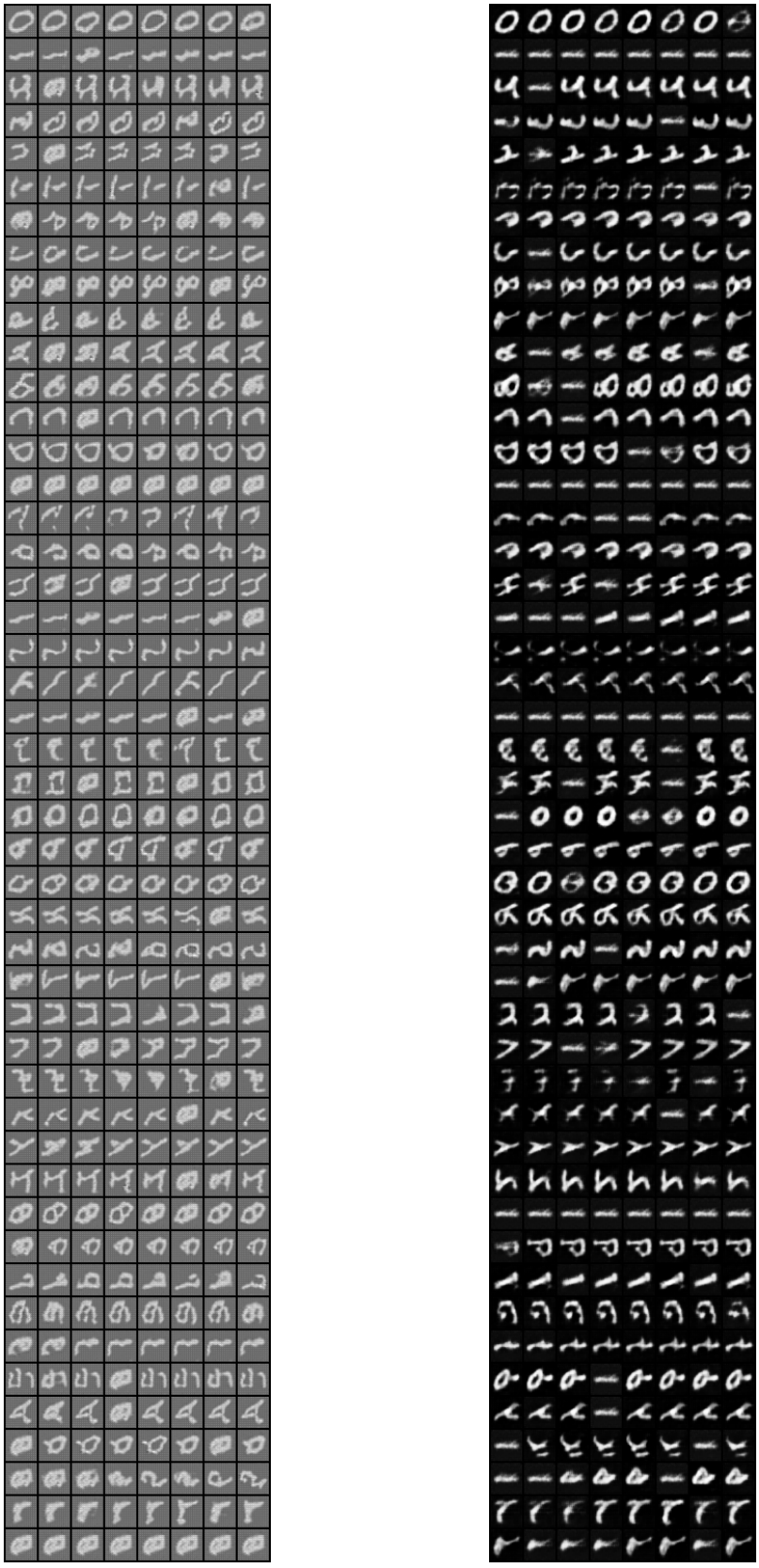
(a) FedGAN (b) FedGDKD

Figure B.11: Generator Performance: Generator final output (EMNIST, $Dir(\alpha = 0.5)$, $r = 25\%$)



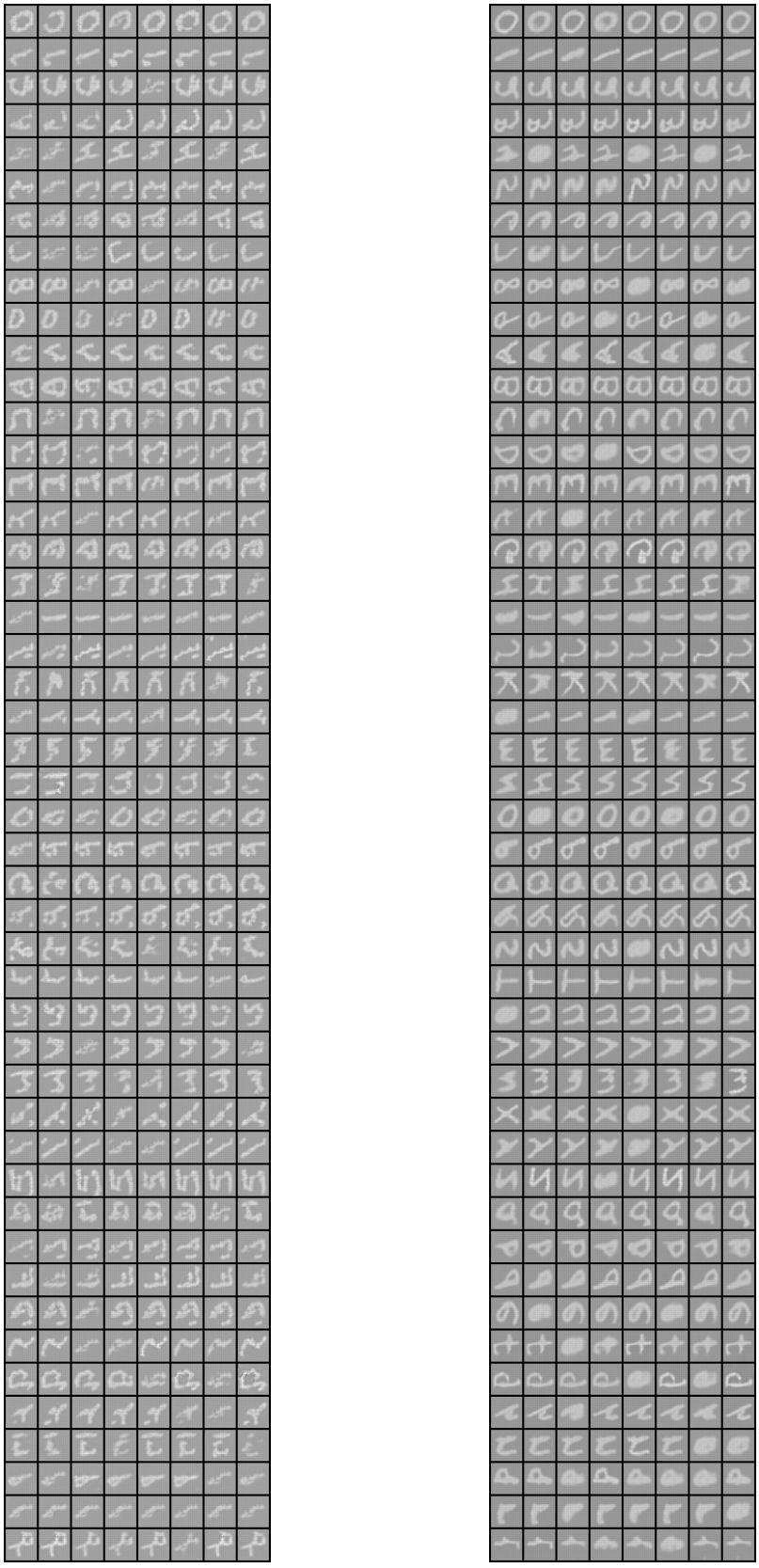
(a) FedGAN (b) FedGDKD

Figure B.12: Generator Performance: Generator final output (EMNIST, $Dir(\alpha = 0.1)$, $r = 25\%$)



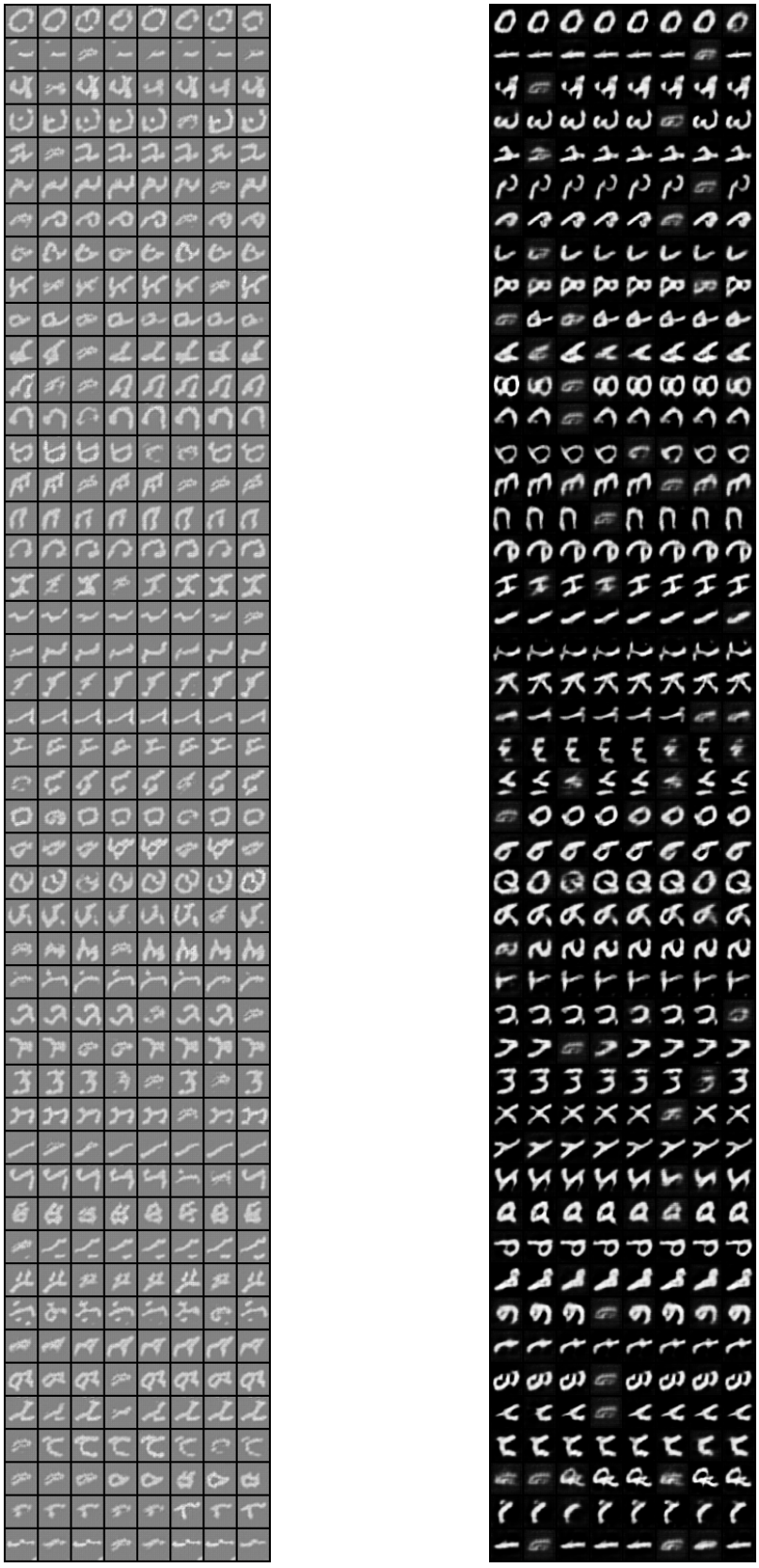
(a) FedGAN (b) FedGDKD

Figure B.13: Generator Performance: Generator final output (EMNIST, $Dir(\alpha = 0.05)$, $r = 25\%$)



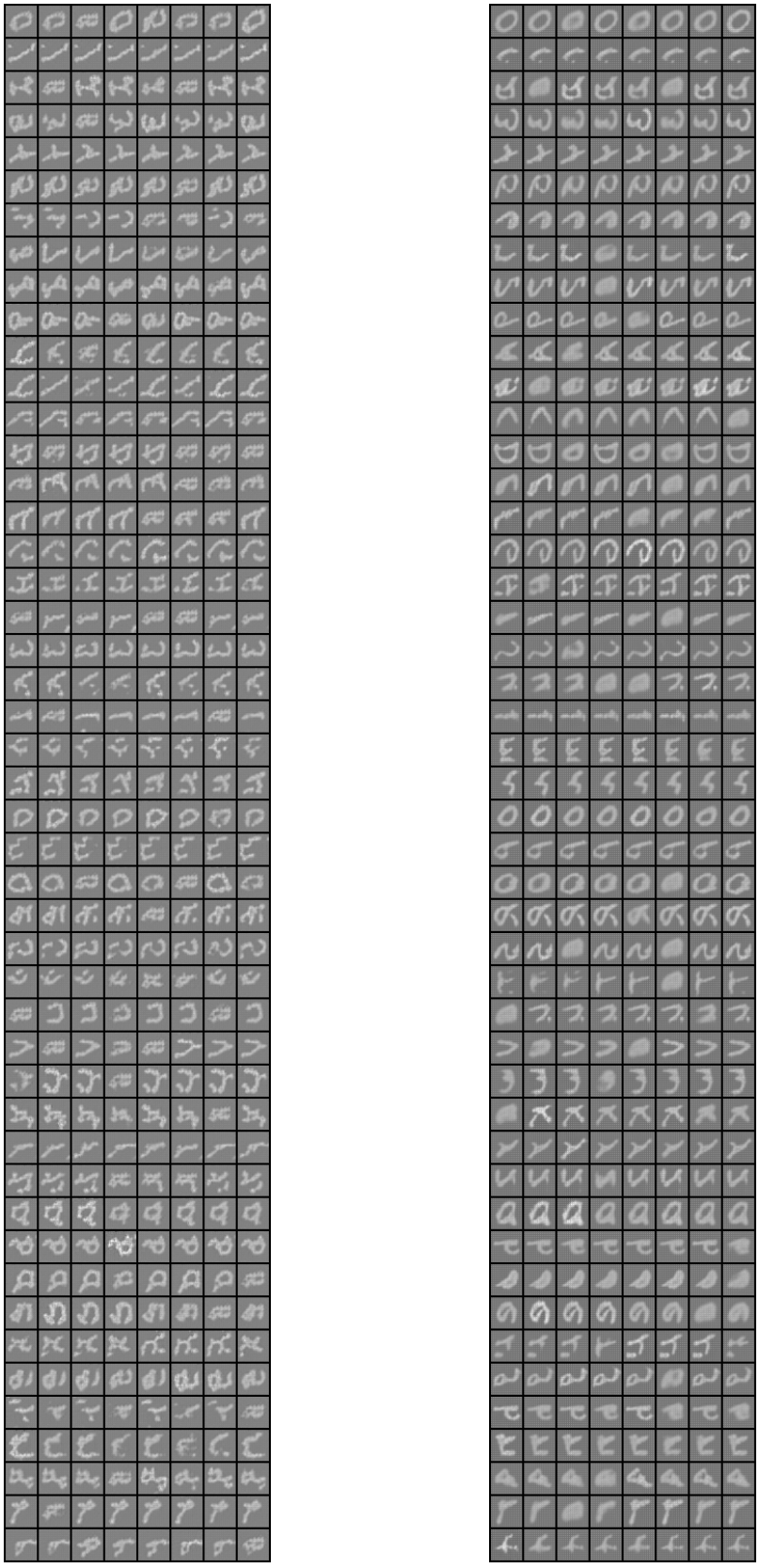
(a) FedGAN (b) FedGDKD

Figure B.14: Generator Performance: Generator final output (EMNIST, $Dir(\alpha = 0.5)$, $r = 10\%$)



(a) FedGAN (b) FedGDKD

Figure B.15: Generator Performance: Generator final output (EMNIST, $Dir(\alpha = 0.1)$, $r = 10\%$)



(a) FedGAN (b) FedGDKD

Figure B.16: Generator Performance: Generator final output (EMNIST, $Dir(\alpha = 0.05)$, $r = 10\%$)