TU Delft

MULTI-FLGANs: Multi Distributed Adversarial Networks for Non-IID distributed datasets

Akash Amalan
Supervisor(s): Kaitai Liang, Rui Wang
EEMCS, Delft University of Technology, The Netherlands
22-6-2022

**A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering**

Abstract

Federated learning is an emerging concept in the domain of distributed machine learning. This concept has enabled GANs to benefit from the rich distributed training data while preserving privacy. However, in a non-iid setting, current federated GAN architectures are unstable, struggling to learn the distinct features and vulnerable to mode collapse. In this paper, we propose a novel architecture MULTI-FLGAN to solve the problem of low-quality images, mode collapse and instability for non-iid datasets. Our results show that MULTI-FLGAN is four times as stable and performant (i.e. high inception score) on average over 20 clients compared to baseline FLGAN.

1 Introduction

General Adversarial Networks (GANs)[1] [2] have been used in applications ranging from style transfer to image-to-image translation. While they are widely used, they suffer from three main problems:

- A lack of data affects their ability to generate high-quality images [3][4].

- They are vulnerable to mode collapse, a case in which a GAN only generates a specific subset of the real images. [5][6].

- They can be unstable (i.e. fluctuations in IS score), failing to converge or visibly improve.[6].

Several architectures[7][3][4][8] were proposed to address the first problem through traditional and differential data augmentation methods. Federated learning(FL), an emerging concept [9], also showed promise in overcoming the inability of GANs to generate high-quality images when lacking data. This paper will address this problem by adapting GANs to a distributed setting through FL. In FL, each client uses their private datasets to jointly learn the global model. The clients trains the model locally with their training data and send the updated weights to the central server to aggregate and update the global model. Thus, FL benefits from rich distributed training data while also potentially preserving the client's privacy.

Similarly, many architectures were proposed to address the last two problems. For instance, Ishan et al. [10] proposed GMAN, a generic architecture with one generator and multiple discriminators. By introducing a one vs all game where the generator tries to fool many discriminators, the chance of mode collapse is significantly reduced. Likewise, Q. Hoang et al.[11] suggested a Multi-Generator GAN (MGAN) with an arbitrary grouping of generators and discriminators to mitigate instability and improve convergence.

FLGAN(or FedAvgGan)[12] was the first architecture to bridge the gap between FL and GANs by assigning a generator and discriminator to each client. For each iteration, FLGAN uses the aggregated weights[13] of local models to update the global model, i.e. global generator and discriminator. However, this baseline architecture was found to be vulnerable to mode collapse[14] and privacy leaks[15].

Although other variants[16] [17] were proposed to improve the performance and privacy of the FLGAN architecture, Hardy et al[18]. were the first to suggest decreasing the computation cost at worker nodes. They did this by introducing MDGAN, a novel architecture that extends GMAN to a distributed setting with a central generator at the server and a discriminator per client. To elaborate, MDGAN swaps each discriminator every k iteration and aggregates their weights using fedAvg. This way, a one vs all game is introduced where the generator at the server attempts to fool all discriminators in each client node. By decreasing the computation cost for each client node, MDGAN has achieved phenomenal[18] performance relative to standard FLGAN for independent and identically distributed(iid) datasets.

Similar to online streaming data with varying bit distribution, real-world data distributions are hardly ever iid. In a non-iid setting, each client's dataset may have a different distribution, significantly increasing training difficulty and mode collapse vulnerability.

Some works[19][20] have presented unique architectures to solve this problem. However, none of these works address the stability problem due to increasing clients. The above discussion raises the following open question: **As the number of clients increase, can an architecture maintain high and stable performance for non-iid datasets while avoiding mode collapse?**

We propose MULTI-FLGAN, an architecture inspired by both MDGAN and MGAN. MULTI-FLGAN extends MGAN to a distributed setting to produce an architecture with high and stable performance for non-iid distributions.

Contributions: This paper's contributions are:

- To propose MULTI-FLGAN, a novel architecture that can produce diverse high-quality images with faster convergence than the baseline FLGAN.

- To compare the performance of MULTI-FLGAN against baseline FLGAN and similar competitors over multiple clients using Inception(IS) and Frechet inception scores(FID).

- To compare the learning performance of MULTI-FLGAN against baseline FLGAN and similar competitors on fixed number of clients, i.e. how well MULTI-FLGAN performs over iterations compared to other competitors on IS score.

- To highlight privacy risks and relevant attacks on MULTI-FLGAN. Specifically, Inference attacks and Model poisoning attacks.

Paper Organisation: In section 2, we give a general background of GAN and FLGAN. Section 3 introduces MULTI-FLGAN. Section 4 details the experimental setup. Section 5 presents the experiments conducted on different architectures along with their results. Section 6 presents a variety of possible improvements to MULTI-FLGAN. Section 7 details responsible research practices. Finally, Section 8 concludes this paper with recommendations for future work.

2 Preliminaries

Classical GANs

A classical GAN, as proposed by Ian Goodfellow et al. [21] consists of two neural networks: a generator G and a discriminator D. Their objective can be described by a min-max game, where the discriminator tries to minimize the probability of classifying a fake sample as real while the generator tries to fool the discriminator by producing data similar to the training set.

Discriminator Learning Phase:

The generator takes in a random noise signal V and generates data d_f in the same format as the training data (e.g. 28x28 with 1 color channel as in MNIST dataset). The data d_f is then fed into the discriminator along with real data d_r. The discriminator acts as a classifier, classifying whether a sample is fake or real. The classification loss is used to train the weights of the discriminator through back-propagation.

Generator Learning Phase:

The generator learns from the loss of the discriminator. It does so by using the Kullback-Leibler Divergence(KL)[22] loss function. KL quantifies how much the generated distribution differs from the actual distribution. The generator can learn from the discriminator and update its weights by employing this loss function.

FLGAN

FLGAN[12] extends classical GAN to a distributed setting. Suppose we have the following setup with N clients: Each client i is equipped with a private dataset d_i, a generator, and a discriminator. The main server has a global model w, i.e a global generator and discriminator. A global iteration of FLGAN's learning algorithm is as follows:

- Each client trains their local models w_i and sends the updated weights to the server.
- The server aggregates these weights using fedAvg[13].
- In the next iteration, the server sends the updated global model w to the clients.
- After e epochs, the main server will have a trained global discriminator and generator.

3 Problem Formulation:

Suppose N clients wish to learn a global model W and agree on a common protocol. Each client i has a private dataset d_i distributed in a non-iid fashion[1]. This highlights that all clients will have an uneven sample distribution. For instance, client A may have 100 samples of label c and only 10 samples of label d, while client B may possess 10 samples of label c but 200 samples of label d .

The clients agree beforehand on the number of discriminators X and generators Y to use. The problem is to efficiently learn the global model W by minimizing the error on each local model w_i using dataset d_i.

We seek to develop an architecture to solve this problem, with the following characteristics:

- Robust: The architecture should maintain a high IS score even in the case of uneven labels for each client. Moreover, the algorithm should perform even when limited training samples are available, i.e. 2000 to 5000 samples.
- Stable: The architecture should perform in a consistent fashion. The IS score of the generated images should not fluctuate greatly when increasing or decreasing the number of clients.
- Performant: The architecture should be able to generate high quality and diverse images.

4 The MULTI-FLGAN Architecture

Rationale: Our architecture is mainly inspired by MGAN and MDGAN. By using multiple generators instead of one generator, MGAN successfully avoided the problem of mode collapse while achieving phenomenal performance. Similarly, MDGAN achieved a comparable performance by having a generator compete against multiple discriminators for a lower computation cost per client. As shown in figure 1, MULTI-FLGAN integrates these approaches by introducing an all vs all game with multiple discriminators and generators for each client.

Figure 1: High Level Architecture

---
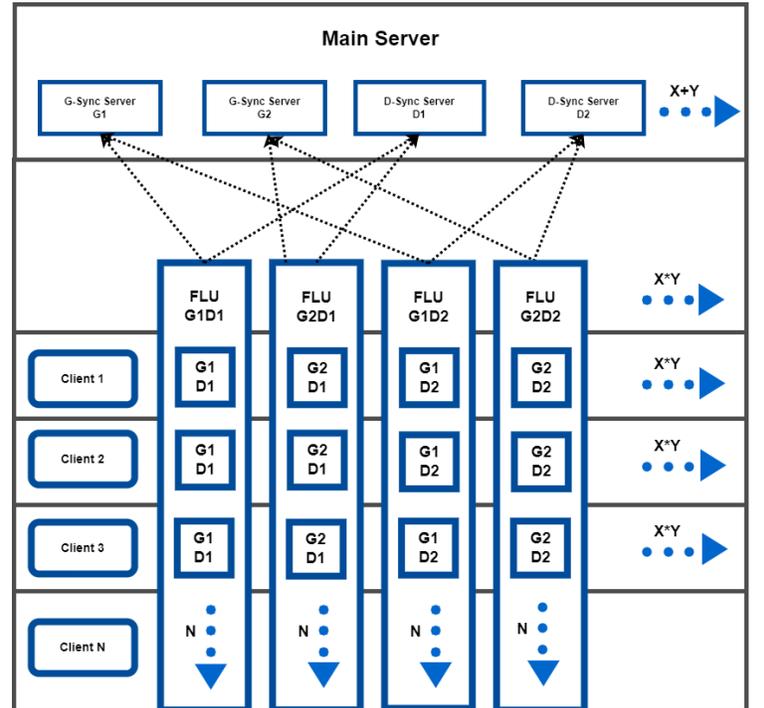
[1]This is further explained under Experiment Setup

Components of architecture

Federated Learning Unit(FLU) - An FLU contains a cluster of N identical GANs, keeping track of their average generator and discriminator weights. For instance, an FLU $G2D1$ contains a cluster of N identical GANs with generator id 2 and discriminator id 1.

Generator Sync server(G-sync) - A G-sync server aggregates the generator weights from its FLUs and stores the resulting weight. For instance, G-sync server G2 aggregates the generator weights of ids $G2D1$ and $G2D2$.

Discriminator Sync server(D-sync) - Similarly, a D-sync server aggregates the discriminators' weights from its FLUs and stores the resulting weight.

Sync server(s) - A general component referring to both D-sync and G-sync servers.

Main server(m) - This component is responsible for both allocating and initiating connections between FLUs and Sync servers.

Protocol: MULTI-FLGAN follows the following protocol:

- **Step 1: Protocol Initiation**
  The client initiates the protocol by sharing parameters X and Y with the server.

- **Step 2: Sync Server Allocation**.
  Main server m allocates X D-sync servers and Y G-sync servers. For example, in figure 1, m allocates 4 Sync servers in total.

- **Step 3: FLU Allocation**.
  The main server allocates $X*Y$ FLUs. Each Sync server then connects to its respective FLU. In figure 1, G-sync server $G1$ connects to FLU ids $G1D1$, $G1D2$, and G-sync server $G2$ connects to ids $G2D1$, $G2D2$. Respectively, D-sync server $D1$ connects to ids $G1D1$, $G2D1$ while $D2$ connects to ids $G1D2$ and $G2D2$.

- **Step 4: Client Distribution**.
  Each FLU will create a partition for each client by replicating identical GANs. For instance, FLU $G2D1$ creates identical replicas of GANs with generator $G2$ and discriminator $D1$ for each client.

Table 1: Summary of notations

Notation	Description
D	Global Dataset
W	Global Model
d_i	Training dataset of client i
e	epochs
X	Number of discriminators
Y	Number of generators
G	Generator
D	Discriminator
fl	Set of all FLUs
s	Set of all Sync servers
s_j	Sync Server j
w_i	Local model of client i
w_j	Local model of s_j
w_a	Local model of FLU a (fl_a)

Learning Algorithm: The goal of our algorithm[2] is to minimize the empirical loss F_w of the global model W on dataset D, with global learning rate α and e epochs. However, in a distributed setting, each client will solve the optimization problem of $min_w f(d_i, \{w_{i1}, ..., w_{ik}\})$, where f is the empirical loss of local models $\{w_{i1}, ..., w_{ik}\}$ under the client i and k is bounded by $X*Y$. Our algorithm takes the following steps to solve this objective:

- **Step 1: Synchronization of global model with FLU**
  Each Sync server s_j has a Sync model w_j. s_j sends w_j to its respective FLU fl_a, which updates its model w_a with the incoming model w_j.

- **Step 2: Training local models**
  Each client i trains local models $\{w_{i1}, ..., w_{ik}\}$ with learning rate $\alpha * N$ and a randomly selected and permuted batch of training dataset D_i. To force the models to converge, we increased the learning rate proportionally to the number of clients. Additionally, we allow for diversity by training each model with a randomly selected and permuted batch. This enables GANs to generalize over uneven distributions.

- **Step 3: Update FLU through aggregation**
  Upon training, each FLU aggregates the weights of its generators and discriminators using fedAvg[13]. Then the FLUs send the updated models to their respective Sync servers.

- **Step 4: Update Sync servers through aggregation**
  Each G-sync and D-sync server average the generator and discriminator weights of connected FLUs, respectively. Then, they update their Sync model w_j with these new weights.

- **Step 5: Termination**
  After e epochs, the main server chooses the best generator out of all G-sync server models using a generic metric such as inception score.

Client Model Learning Procedure: All client models are trained in batches of 64 samples. From a random noise vector V in each iteration, the generator generates a D^f batch with the same dimensions as a real sample. This is sent along with real batch D^r to the discriminator. The discriminator uses the loss of miss-classifying fake samples to update its weights. Similarly, the generator uses the KL [22] loss function to learn the true training sample distribution from the discriminator.

Main Characteristics of our algorithm:

- Our architecture introduces a new all vs all game where multiple generators play against multiple discriminators.

- By extending the idea of MGAN[3], we essentially minimize the risk of mode collapse

- Although the distributions for each client are different(non-iid), our algorithm is still able to learn their distinct features.

---

[2]The learning algorithm of the proposed architecture is presented on page 5

[3]Refer to the paper on MGAN

Algorithm 1: The Multi-FLGAN Algorithm

---

procedure SYNC(X, Y, fl, s)
  for $j = 1, 2 \ldots X+Y$ **do**
    for $a = 1, 2 \ldots X*Y$ **do**
      if ISCONNECTED(s_j, fl_a) **then**
        SEND(s_j, fl_a, w_j)
        Update (w_a, w_j)
      end
    end
  end

procedure TRAINFLU(N, X, Y)
  for $i = 1, 2 \ldots N$ **do**
    for $a = 1, 2 \ldots X*Y$ **do**
      $batch_i = $ GETRANDOM(D_i)
      TRAINLOCAL($w_{i_a}, batch_i, \alpha * N$, e)
    end
  end

procedure UPDATEFLU(N, X, Y, fl)
  for $a = 1, 2 \ldots X*Y$ **do**
    $sum_g = 0$
    $sum_d = 0$
    for $i = 1, 2 \ldots N$ **do**
      $sum_{gen} = sum_g + $ GETGWEIGHT(fl_{a_i})
      $sum_{disc} = sum_d + $ GETDWEIGHT(fl_{a_i})
    end
    fl_a.SETGWEIGHT($\frac{sum_g}{N}$)
    fl_a.SETDWEIGHT($\frac{sum_d}{N}$)
  end

procedure UPDATESYNCSERVER(N, X, Y, fl)
  for $j = 1, 2 \ldots X+Y$ **do**
    $s_g = 0$
    $s_d = 0$
    for $a = 1, 2 \ldots X*Y$ **do**
      if ISCONNECTED(s_j, fl_a) **then**
        $s_g = s_g + fl_a$.GETGWEIGHT()
        $s_d = s_d + fl_a$.GETDWEIGHT()
      end
    end
    s_j.SETGWEIGHT($\frac{sum_g}{X}$)
    s_j.SETDWEIGHT($\frac{sum_d}{Y}$)
  end

procedure TERMINATION(N, X, Y, S)
  $score = 0$
  $best_w = $ None
  for $j = 1, 2 \ldots X+Y$ **do**
    if $score > $ GETSCORE(s_j) **then**
      $score = $ GETSCORE(s_j)
      $best_w = s_j$
    end
  end
  return $best_w$

---

5 Experimental Setup

Computational Setup: Recently, there has been a trend in adapting serverless [18] frameworks(e.g. gossip protocol introduced by Dynamo back in 2007). However, the case for deep learning and machine learning systems is not yet sufficient. Distributed deep learning systems use massive amounts of data that use data-intensive operations such as back and forward propagation, making it necessary to operate in a parallel environment. We extended the implementation[4] of Distibuted GAN using Ray to emulate a parallel environment for distributed learning. Ray [23] was mainly used due to its superior performance benchmarks in distributed machine learning instead of frameworks like Dask[24].

Ray's distributed framework uses a central Redis server to simulate a head node responsible for communication and aggregation between different worker nodes. We simulated the main server as the head node and each Sync server as a worker node. We also assigned worker nodes for each FLU to simulate multiple FLUs under a sync server. In total, we used 8 worker nodes - 4 for Sync servers and 4 for FLUs.

Moreover, each GAN was given its own thread for each FLU worker node. This allowed for parallel training across all FLUs over multiple nodes. Of course, such an elaborate setup introduces latency and bandwidth issues. Nevertheless, such an environment was necessary to mimic real life use cases.

Datasets: We experimented with two classical datasets used for machine learning: MNIST[25] and Fashion MNIST(FMNIST) [26]. The MNIST and FMNIST datasets are comprised of 60,000 28 x 28-pixel grayscale image samples of handwritten digits, and clothing, respectively.

Sampling Process: To distribute k samples in a non-iid fashion from a global dataset D, we took a unique approach. We randomly selected 5000 samples from D as the training set t, to emulate the lack of data and decrease the training time. Next, we reserved a random fraction F_i of t, using Mersenne Twister[27], for each client. Each F_i was then divided into batches of 64 samples and assigned to their respective client.

Hardware: Our experiments were made using a TensorFlow backend with 4 NVIDA Tesla v100 GPUs and 100 CPU cores made available by DelftBlue[5][28]. The head node was given 44 cores and each worker node was assigned a partition of available GPUs.

Gan Architectures: In our experiments, we used a traditional type of GAN named DCGAN[29] [6]. The generator is comprised of six transposed convolution layers of 128, 64, 32, and 1 with kernels of size 5 x 5. On the other hand, the discriminator uses four transposed convolution layers of 32, 64, 128 and 256 with kernels of size 5 x 5 and a fully connected dense layer. Typically, a dropout layer is also included in both of these neural networks. Instead, we opted for batch normalization, as it helps quicken the learning process.

Hyperparameters: Since the dropout layer was removed, we introduced a decay factor to prevent over-fitting. In order to fine-tune the hyperparameters of a standalone DCGAN, we performed grid search, varying the decay factor and learning

---

[4]https://github.com/bbondd/DistributedGAN
[5]High Performance Computing Platform
[6]Refer to Appendix A.1 for architecture diagram

rate. We found that the optimal decay factor was $1.5e^{-8}$ with a fixed learning rate of $2e^{-4}$.

Competing Approaches: We tested our architecture against traditional FLGAN and AFLGAN, a variant of MDGAN that aggregates only the generator weights while leaving the discriminator weights unchanged across iterations. We simulated AFLGAN and FLGAN using the same Ray setup and hardware. The main server was assigned its own head node, while each client was assigned one worker node.

Metrics: Evaluating GANs is often difficult as an objective metric is needed to capture image diversity and quality. We must ask, "Do images look like a specific object" and "Is a wide range of objects generated?"

A popular metric, Inception Score(IS)[30] addresses quality by considering how strongly an image is classified as one class over others. Similarly, it considers diversity by examining the marginal probability distribution of the generated images. Typically, a low inception score indicates low quality and uniform image, while a high inception score implies diverse and high-quality images.

However, the inception score does not reveal how far off the generated images are from the real images. Therefore we also use Frechet Inception Distance[31] or FID for short, as proposed by Heusel et al. FID calculates the distance between feature vectors of the original set of images and the generated set of images. In contrast with the IS score, a low FID score is preferred, as it implies that the difference between the distributions is small.

Table 2: Experiment Parameters

Experiment	IV	DV	Samples	LearningRate
Type 1	Clients	IS/FID	5000	0.0002
Type 2	Iterations	IS	5000	0.0002

Experiments: We conducted two kinds of experiments. First, we tested how well each architecture performs by varying the number of clients $N \in [2, 3, 5, 10, 20]$ on both datasets(FMNIST and MNIST). Each experiment was performed for 100 epochs with noise vector V of dimension 100. IS and FID scores were computed every 10 epochs. Note that FID scores were computed using the test dataset dimensions: 10000 samples.

Second, we use IS score to compare how well different architectures learn through iterations $e \in [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]$ for fixed clients $N \in [2, 3, 5, 10, 20]$ on both datasets(FMNIST and MNIST).

The experiments used MULTI-FLGAN with only two discriminators and generators. We opted for 2 discriminators and generators to demonstrate that even with minimal parameters, our architecture is far more performant than other alternatives. While this may not be the ideal amount of discriminators and generators, it suffices for the purposes of this study. Testing precise head-to-head comparisons between different discriminators and generators is left to future work.

6 Experiment Evaluation

Tables 3 and 4 report the average, min and max FID and IS scores of over 20 clients for experiments testing clients' performance. Additionally, figure 2 depicts graphs[7] of the client and learning performance experiments.

Table 3: Client Performance of FMNIST(IS/FID)

Heuristic	AFLGAN	FLGAN	MULTI-FLGAN
Min	1.00 / 328.10	1.00 / 442.41	3.82 / 11.60
Max	2.70 / 847.00	1.89 / 646.39	6.48 / 129.35
Average	2.00 / 535.30	1.37 / 495.75	4.95 / 82.33

2) Competitor score for FMNIST: With an average IS score of 1.37 and an FID score of 495.75, we observe that FL-GAN performs the worst relative to its competitors. It is also apparent from the FMNIST dataset inception scores shown in figure 2 that FLGAN struggles to generate high-quality data, having only achieved a maximum IS score of 1.89 for two clients. The learning performance of the FMNIST dataset is also unimpressive. For 2 to 3 clients, it reaches an average IS score of 4.5. For 5 clients, performance drops further to 2, and for 10 to 20 clients, hardly any learning occurs. This poor performance is further reflected by the generated[8] images, which gradually become undecipherable with increasing clients.

On the other hand, AFLGAN performs relatively well compared to FLGAN for 2,3, and 5 clients but drops to a minimum inception score of 1 for 10 and 20 clients, resulting in an average inception score of 2.00 and a FID score of 535.00. This sudden drop implies that AFLGAN cannot maintain its performance over increasing clients.

Interestingly, the learning performance graphs of FMNIST for 10 and 20 clients show that AFLGAN peaks at 30 - 40 iterations, even matching MULTI-FLGAN performance, before suddenly decreasing in performance. This behavior is verified by the generated images in Appendix A.3, where an obvious decrease in quality between iterations 40 and 50 is apparent.

We believe that this behavior is a consequence of uneven training samples. For instance, if generator $G1$ trains on 200 samples of label A while generator $G2$ trains on only 10 samples of label A, their weights will differ considerably. This leads to not only training instability, but also failure to converge, as evident in figure 2. Of course, it is unrealistic to expect convergence from only 100 iterations. Having said that, the parabolic behavior is proof of AFLGAN's inability to converge.

In contrast, MULTI-FLGAN tackles uneven distributions by having multiple generators compete against multiple discriminators. This results in more robust scores, as depicted in Table 3. While the fluctuation between 2, 3 and 5 clients shows that uneven label distributions still influence our architecture, it still maintains an average IS score of 4.95 and a 82.33 FID score, outperforming the other competitors. Additionally, the IS and FID graphs for FMNIST shown in figure 2 make clear that our architecture is performant even with 10

---

[7]Note that learning performance graphs for 2, 3 and 5 clients are presented in figure 6 of Appendix A.2

[8]Refer to Appendix A.3

Figure 2: Experiment Scores
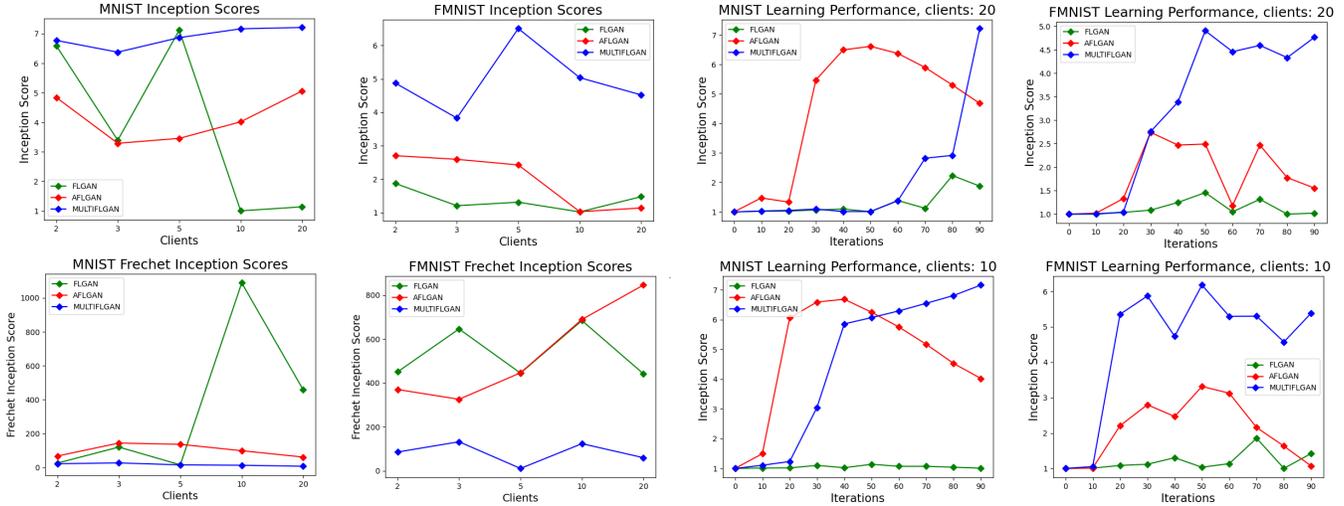
and 20 clients, achieving four times AFLGAN and FLGAN scores.

Table 4: MNIST(IS/FID)

Heuristic	AFLGAN	FLGAN	MULTI-FLGAN
Min	3.30/ 60.70	1.00/ 15.46	6.39/ 8.07
Max	5.10/ 141.90	7.10/ 1087.00	7.15/ 26.50
Average	4.10/ 101.20	3.81/ 341.34	6.84/ 17.10

2) Competitor score for MNIST: All architectures performed much better on MNIST than FMNIST, as seen on table 4, since it is much easier to learn the distribution of numbers than complex fashion designs with only 5000 samples and 100 epochs.

Interestingly, FLGAN experienced a drop in IS score from 7 to 1 between 5 and 10 clients. Inspecting the generated images [9], it indicates that FLGAN experienced mode collapse, resulting in an average IS score of 3.81 and FID score of 341.34. In this case, the mode collapse happened at iteration 50 where FLGAN exclusively generated the number 3. Subsequent iterations show that FLGAN was unable to generalize over other numbers, leading to a meaningless final outcome.

In contrast, AFLGAN performed exceedingly well. It sustained an average inception score of 4.10 and an FID score of 101.20. Although it exceeds the learning performance of MULTI-FLGAN in the first few iterations for clients 10 and 20, AFLGAN performance gradually decreases over subsequent iterations. This parabolic behavior reconfirms its inability to converge. Moreover, the images in figures 30 and 31 of Appendix A.4 show that AFLGAN mostly generates the number 1, hinting at a possible mode collapse.

MULTI-FLGAN, on the other hand, maintained the same performance using MNIST as FMNIST, resulting in an average IS score of 6.84 and an FID score of 17.10 without signs of mode collapse. Interestingly, the learning performance graphs for 10 and 20 clients show a sudden jump in improve-

[9]Refer to figure 24 of Appendix A.4

ment for both MNIST and FMNIST. For MNIST, the jump occurs at 50 iterations for 20 clients, while for FMNIST, the jump occurs at 20 iterations for 20 clients. This may be due to a lack of encountered samples in the first 20 or 50 iterations. However, once MULTI-FLGAN sees sufficient samples, its performance increases exponentially.

It is also interesting to compare the stability of MULTI-FLGAN with other competitors. We quantify *stability* as the difference between the max and min of FID scores. Intuitively, the distance between generated and actual images should not drastically fluctuate between clients.

Table 5: Stability(MNIST/FMNIST)

Heuristic	AFLGAN	FLGAN	MULTI-FLGAN
Stability	81.2/518.9	1071.9/203.9	18.43/117.8

Table 5 shows that MULTI-FLGAN is at least 30 times as stable as AFLGAN and 58 times times as stable as FLGAN. Trained on FMNIST, FLGAN's stability is deceptively close to that of MULTI-FLGAN. Note, however, that FLGAN's FID score is far higher relative to MULTI-FLGAN, indicating that the generated images vary significantly from the actual distribution.

2) Overall Performance: FLGAN performs poorly on both datasets, struggling to converge on higher number of clients. AFL-GAN produces decent results for the first 5 clients but then experiences a sudden drop when generalizing over 10 or more clients. On the other hand, MULTI-FLGAN has outperformed both architectures on both datasets. The graphs and the generated images show that MUTLI-FLGAN can both generalize over uneven samples and produce high-quality images with stable IS and FID scores over increasing clients.

3)Trends: It is hard to predict any architecture's learning performance trend for 50 or 100 clients. However, seeing that the learning performance of AFLGAN has already peaked within 100 iterations for 10 and 20 clients, it is safe to assume that it will only perform worse on a higher number of

clients. Since FLGAN reached a minimum IS score of 1 on both data sets for ten clients, it cannot be reasonably expected to learn for a higher number of clients. However, we cannot rule out the possibility of FLGAN realizing a jump similar to MULTI-FLGAN when given enough samples.

The learning performance graphs of FMNIST show that MULTI-FLGAN is stabilizing around iteration 70, while on MNIST, it is still increasing beyond 90 iterations. Regardless, the IS and FID scores lead us to believe that even for clients above 50, MULTI-FLGAN would be able to maintain the average inception score without significant fluctuations.

7   Improvements & Considerations

MULTI-FLGAN has shown promise in terms of performance, robustness, and stability. Nonetheless, we must still consider other aspects such as cost, scaling, and security prior to deployment.

Complexity: One of the main concerns is the architecture's suitability for the storage and computation limitations of small devices such as mobiles. To aid the discussion on computation and space complexity, we will introduce new notations in addition to Table 1. Let o be the object size(e.g image size in MB), b batch size and D_{i_o}, number of objects in local dataset D_i of client i.

The computation complexity and space complexity of FLGAN and MULTI-FLGAN have been summarized in table 6.

Table 6: Complexity

	FLGAN	**MULTI-FLGAN**
O(C)	$O(eb\sum_{i=1}^{N}\frac{|w_i|}{D_{i_o}})$	$O(eb(\sum_{i=1}^{N}\sum_{j=1}^{XY}\frac{|w_{i_j}|}{D_{i_o}})$
O(S)	$O(\sum_{i=1}^{N}|w_i|)$	$\sum_{i=1}^{N}\sum_{j=1}^{XY}|w_{i_j}|)$

Admittedly, the computation complexity of MULTI-FLGAN is much more expensive than traditional FL-GAN as it depends on both N and XY. Furthermore, in practice, there is the additional cost of communication between the main server, Sync servers and FLUs. Table 7 presents the computation time of different architectures used in experiment type 1.

Table 7: Time taken for Experiment Type 1

Architecture	Time Taken				
	Clients				
	2	**3**	**5**	**10**	**20**
FLGAN	0.1	0.2	0.6	2.5	4.5
AFLGAN	0.1	0.2	0.7	2.4	4.5
MUTLI-FLGAN	0.2	0.4	6.0	12.0	23.0

We notice that MULTI-FLGAN takes almost 6 times as long as FLGAN and AFLGAN. However, there are several adaptations we can make to improve this computational complexity. One possible way is to adapt a similar technique to MDGAN, where the number of FLUs are significantly reduced.

The strategy is to divide each combination of generators and discriminators among the clients avoiding the need for replication. Consider the case with two generators, three discriminators, and three clients. Client A will train $G1D1$ and $G1D2$, Client B - $G2D1$ and $G2D3$, and Client C - $G2D2$ and $G1D3$ reducing the number of FLUs needed to two. Eventhough, there is a significant improvement in the computation complexity, each client now trains a fraction of the models in the former architecture. To compensate for the lack of models, we will introduce a peer-to-peer mechanism where each discriminator and generator is swapped with a random client every two epochs .

The adapted strategy results in a drastic decrease in computation cost while preserving the same notion of an All vs ALL game. A possible drawback of using this adapted approach is connectivity. If a client disconnects during training, the algorithm loses unique pairs of generators and discriminators hindering the performance of the global model.

As seen from the space complexity of table 4, our algorithm uses much more memory than baseline FLGAN. However, we assume clients that adopt our protocol would meet the computational and spatial requirements. Besides most modern smart devices can easily handle 5 to 10 MB of data(Size of a model). Therefore we believe that our architecture is still feasible for a rational choice of X and Y.

Scalability: The second aspect to consider is whether our architecture is readily scalable. Currently, our architecture can accommodate any amount of clients with arbitrary parameters X and Y. Any client may participate in the training as long as they join at Step 1 of our learning procedure using the same X and Y parameters as other clients. If not, the client will be asked to wait until it reaches step 1 of our learning algorithm again for synchronization. In this sense, our architecture is scalable on demand.

However, our architecture does not currently support clients with different X and Y parameters. But by clustering FLUs based on X and Y, we can allow multiple clients with different parameters to train simultaneously.

Fault Tolerance: Our architecture is also resilient to node failures. If one FLU fails, then the respective Sync server will simply continue aggregating models from other connected FLUs. Likewise, if a Sync server fails the training would still continue with other available Sync servers. The same principle holds for a client who disconnects form the main server.

Security: Federated learning is vulnerable to different types of attacks. We will mainly consider Inference and Model poisoning attacks.

We consider a similar attack model to that of FL-TRUST[32]. More specifically, an attacker manipulates a minority of malicious clients, which can be fake clients injected by the attacker or genuine clients compromised by the attacker. The malicious clients can send arbitrary updates with the intent of destroying the model or inferring the private data of other clients. We assume that the attacker has full knowledge and access to the following information: learning rate, models and latent vector of compromised clients.

Inference Attacks: In an inference attack, the attacker is interested in inferring the private data of other clients. Our architecture is particularly vulnerable, as each client has access to a partition of all models in FLUs.

We assume the following setting: N clients train MULTI-FLGAN with 2 discriminators and generators on both MNIST and FMNIST dataset. Each malicious client j does not actively participate in training models $\{w_{j_1}, ..., w_{j_k}\}$, instead they return the models as it is. After e epochs, the attacker can simply use any of the compromised clients' generators to produce images from random noise. These generated images can then be easily reconstructed as described by hitaj et al. [33]. We have presented the result of such an attack with 15 genuine clients and 5 malicious clients in figure 3.

Figure 3: Inference attacks
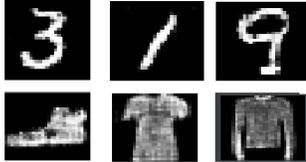
From figure 3, it is clear that the malicious clients could successfully infer private training data through this attack. Inference attacks can be mitigated using differential privacy. Differential privacy obfuscates the generator weights by adding Gaussian noise to protect the privacy of the training data set.

Typically, the noise level can have a perturbing effect on the performance of the global model. A lot of research was done to mitigate its affect on the IS score of the generated images. For example, Xie et al. [34] suggested using DP with carefully designed noise vectors with gradient clipping to minimise the impact on the global model. Similarly, Xinaglong et al. [35] proposed training generators, with a special loss function that obfuscates only the visual features. Both these methods cause little harm to the overall performance of the model. Our architecture could be adapted to use either of these methods by changing the Client learning procedure.

Model Poisoning Attacks: Another common family of attacks are model poisoning attacks. In this scenario, the attacker wants to destroy or force the global model only to generate images that the attacker is interested in. The only difference from the previous scenario is that the malicious clients update the models with random weights instead of returning the model. Admittedly, we have not conducted any experiments to deduce the effect of model poisoning on our architecture. However, we would like to remark that this is very similar to having uneven distributions for each client. Therefore, we believe that for a minority of clients, our architecture would still be quite resilient.

However, the averaging operation used by the FLUs is quite sensitive to differences in weight. This operation could be improved by replacing the it with criteria-based aggregation methods. For instance, one could use the inception score as a metric to choose the best model for the next iteration or even other statistical methods such as trim-mean[36], median and Krum[37].

8   Responsible Research

Any research aims to discover, enrich and reinforce the current domain of science through insights into existing work or discoveries that broaden the current domain. However, the researchers have to take careful measures to ensure the research's validity and integrity.

Research Integrity Research integrity primarily focuses on ensuring a fair and truthful subject analysis. To provide a truthful analysis, the researcher must ensure that the results are not manipulated or trimmed without a valid explanation. For instance, if the researcher notices certain outliers in a subject, it may be necessary to remove them to highlight the findings. However, the researcher must then explicitly mention and explain why it was necessary in that context. In contrast, if the researcher willingly manipulated the data - trimming, clipping or discarding without any explicit mention, the research loses its integrity. The second aspect of research integrity is focused on the authenticity of the research. Typically, researchers refer to extensive scientific literature when formalizing the literature study. However, they must take extra care to give credit to all the work that has been used, even if it has not been explicitly published. In our paper, we explicitly mention the use of an adapted implementation even though the author has wished to remain anonymous. The third aspect is to ensure a fair test of the subject. Highlighting the positive aspects of research is essential, but it is equally important to highlight its disadvantages. In other words, the researcher must ensure that his or her work can stand the scrutiny of other peers. Besides, without a proper analysis, the research might come to a biased conclusion that may not be objectively true. In our research, we took extra care to mention the improvements that can be made to our architecture.

Research Reproducability As previously mentioned, any research must withstand the scrutiny of fellow peers. However, to truly criticise a subject, the subject must be clearly reproducible. But there are occasions where the research cannot be reproduced to the exact decimal; in such cases, the researcher should be able to defend and explain why a discrepancy is plausible. This is only possible if the researcher clearly and transparently explains the methodology to arrive at the results as indicated by the research.

In other cases, the scientific community may not agree with the author's interpretation of the results. However, this does not have any implications for reproducibility. In our work, we extensively explain the methodology and share the pseudo code and our setup to ensure that others can reproduce our results.

9   Conclusion

We proposed General Adversarial Networks in the novel context of distributed learning to solve the instability and model collapse problem for non-iid datasets. We drew inspiration from MDGAN and MGAN to develop an all vs all game that allows GANs to easily generalize over uneven labels . Our extensive evaluations of 2 datasets show that our architecture can achieve phenomenal performance and stability with minimal discriminators and generators. In particular, MULTI-FLGAN was able to achieve to maintain the highest average IS score over the 20 clients we experimented with. However, in our research, we tested MULTI-FLGAN only on two grayscale datasets. It would be interesting to see how well MULTI-FLGAN performs on coloured datasets and with different parameters. We believe this work has introduced a vi-

able solution for the instability problem experienced by federated GANs and hope that raised perspectives will inspire future works.

References

[1] J. Brownlee, *18 impressive applications of generative adversarial networks (gans)*, Jul. 2019. [Online]. Available: https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/.

[2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[3] D. 7 and I. Salian, *Nvidia research achieves ai training breakthrough*, Jun. 2021. [Online]. Available: https://blogs.nvidia.com/blog/2020/12/07/neurips-research-limited-data-gan/.

[4] S. Zhao, Z. Liu, J. Lin, J.-Y. Zhu, and S. Han, "Differentiable augmentation for data-efficient gan training," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 7559–7570. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/55479c55ebd1efd3ff125f1337100388-Paper.pdf.

[5] X. Mao and Q. Li, "Generative adversarial networks (gans)," *Generative Adversarial Networks for Image Generation*, pp. 1–7, 2020. DOI: 10.1007/978-981-33-6048-8_1.

[6] S. A. Barnett, "Convergence problems with generative adversarial networks (gans)," *CoRR*, vol. abs/1806.11382, 2018. arXiv: 1806.11382. [Online]. Available: http://arxiv.org/abs/1806.11382.

[7] Y. Burad and K. Burad, "A comparative study of cycle gan and progressive growing gan for synthetic data generation," *International Journal of Engineering Applied Sciences and Technology*, vol. 5, no. 3, pp. 657–660, 2020. DOI: 10.33564/ijeast.2020.v05i03.114.

[8] T. Chen, Y. Cheng, Z. Gan, J. Liu, and Z. Wang, "Ultra-data-efficient GAN training: Drawing A lottery ticket first, then training it toughly," *CoRR*, vol. abs/2103.00397, 2021. arXiv: 2103.00397. [Online]. Available: https://arxiv.org/abs/2103.00397.

[9] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. [Online]. Available: https://arxiv.org/abs/1610.05492.

[10] I. Durugkar, I. Gemp, and S. Mahadevan, *Generative multi-adversarial networks*, 2016. DOI: 10.48550/ARXIV.1611.01673. [Online]. Available: https://arxiv.org/abs/1611.01673.

[11] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, *Multi-generator generative adversarial nets*, 2017. DOI: 10.48550/ARXIV.1708.02556. [Online]. Available: https://arxiv.org/abs/1708.02556.

[12] R. Ghonima, "Implementation of gans using federated learning," in *2021 Tenth International Conference on Intelligent Computing and Information Systems (ICICIS)*, 2021, pp. 142–148. DOI: 10.1109/ICICIS52592.2021.9694141.

[13] T. Sun, D. Li, and B. Wang, *Decentralized federated averaging*, 2021. DOI: 10.48550/ARXIV.2104.11375. [Online]. Available: https://arxiv.org/abs/2104.11375.

[14] G. Xie, J. Wang, Y. Huang, *et al.*, *Fedmed-gan: Federated multi-modal unsupervised brain image synthesis*, Jan. 2022.

[15] X. Zhang and X. Luo, *Exploiting defenses against gan-based feature inference attacks in federated learning*, 2020. DOI: 10.48550/ARXIV.2004.12571. [Online]. Available: https://arxiv.org/abs/2004.12571.

[16] D. Chen, T. Orekondy, and M. Fritz, *Gs-wgan: A gradient-sanitized approach for learning differentially private generators*, Jun. 2020.

[17] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, *Differentially private generative adversarial network*, 2018. DOI: 10.48550/ARXIV.1802.06739. [Online]. Available: https://arxiv.org/abs/1802.06739.

[18] C. Hardy, E. L. Merrer, and B. Sericola, "MD-GAN: Multi-discriminator generative adversarial networks for distributed datasets," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2019. DOI: 10.1109/ipdps.2019.00095. [Online]. Available: https://doi.org/10.1109%2Fipdps.2019.00095.

[19] X. Cao, G. Sun, H. Yu, and M. Guizani, *Perfed-gan: Personalized federated learning via generative adversarial networks*, 2022. DOI: 10.48550/ARXIV.2202.09155. [Online]. Available: https://arxiv.org/abs/2202.09155.

[20] W. Li, J. Chen, Z. Wang, Z. Shen, C. Ma, and X. Cui, "Ifl-gan: Improved federated learning generative adversarial network with maximum mean discrepancy model aggregation," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2022. DOI: 10.1109/tnnls.2022.3167482.

[21] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, *Generative adversarial networks*, 2014. DOI: 10.48550/ARXIV.1406.2661. [Online]. Available: https://arxiv.org/abs/1406.2661.

[22] J. M. Joyce, "Kullback-leibler divergence," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_327. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_327.

[23] P. Moritz, R. Nishihara, S. Wang, *et al.*, "Ray: A distributed framework for emerging AI applications," *CoRR*, vol. abs/1712.05889, 2017. arXiv: 1712.05889. [Online]. Available: http://arxiv.org/abs/1712.05889.

[24] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th Python in Science Conference*, K. Huff and J. Bergstra, Eds., 2015, pp. 130–136.

[25] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[26] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms*, cite arxiv:1708.07747Comment: Dataset is freely available at https://github.com/zalandoresearch/fashion-mnist Benchmark is available at http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747.

[27] M. Matsumoto and T. Nishimura, "Mersenne twister," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998. DOI: 10.1145/272991.272995.

[28] D. H. P. C. C. (DHPC), *DelftBlue Supercomputer (Phase 1)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[29] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015. DOI: 10.48550/ARXIV.1511.06434. [Online]. Available: https://arxiv.org/abs/1511.06434.

[30] T. Salimans, I. Goodfellow, W. Zaremba, *et al.*, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

[31] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," Dec. 2017.

[32] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *CoRR*, vol. abs/2012.13995, 2020. arXiv: 2012.13995. [Online]. Available: https://arxiv.org/abs/2012.13995.

[33] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: Information leakage from collaborative deep learning," Oct. 2017, pp. 603–618. DOI: 10.1145/3133956.3134012.

[34] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, *Differentially private generative adversarial network*, 2018. DOI: 10.48550/ARXIV.1802.06739. [Online]. Available: https://arxiv.org/abs/1802.06739.

[35] X. Luo and X. Zhu, "Exploiting defenses against gan-based feature inference attacks in federated learning," *CoRR*, vol. abs/2004.12571, 2020. arXiv: 2004.12571. [Online]. Available: https://arxiv.org/abs/2004.12571.

[36] R. Kowalchuk, H. Keselman, J. Algina, and R. Wilcox, "Multiple comparison procedures, trimmed means and transformed statistics," *Journal of Modern Applied Statistical Methods*, vol. 5, pp. 43–64, May 2006. DOI: 10.22237/jmasm/1146456300.

[37] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

A   Appendix

A.1   DCGAN Architecture

Figure 4: Generator

input_1	input:	[(None, 100)]
InputLayer	output:	[(None, 100)]

dense	input:	(None, 100)
Dense	output:	(None, 6272)

reshape	input:	(None, 6272)
Reshape	output:	(None, 7, 7, 128)

batch_normalization	input:	(None, 7, 7, 128)
BatchNormalization	output:	(None, 7, 7, 128)

activation	input:	(None, 7, 7, 128)
Activation	output:	(None, 7, 7, 128)

conv2d_transpose	input:	(None, 7, 7, 128)
Conv2DTranspose	output:	(None, 14, 14, 128)

batch_normalization_1	input:	(None, 14, 14, 128)
BatchNormalization	output:	(None, 14, 14, 128)

activation_1	input:	(None, 14, 14, 128)
Activation	output:	(None, 14, 14, 128)

conv2d_transpose_1	input:	(None, 14, 14, 128)
Conv2DTranspose	output:	(None, 28, 28, 64)

batch_normalization_2	input:	(None, 28, 28, 64)
BatchNormalization	output:	(None, 28, 28, 64)

activation_2	input:	(None, 28, 28, 64)
Activation	output:	(None, 28, 28, 64)

conv2d_transpose_2	input:	(None, 28, 28, 64)
Conv2DTranspose	output:	(None, 28, 28, 32)

batch_normalization_3	input:	(None, 28, 28, 32)
BatchNormalization	output:	(None, 28, 28, 32)

activation_3	input:	(None, 28, 28, 32)
Activation	output:	(None, 28, 28, 32)

conv2d_transpose_3	input:	(None, 28, 28, 32)
Conv2DTranspose	output:	(None, 28, 28, 1)

activation_4	input:	(None, 28, 28, 1)
Activation	output:	(None, 28, 28, 1)

Figure 5: Discriminator

input_2	input:	[(None, 28, 28, 1)]
InputLayer	output:	[(None, 28, 28, 1)]

leaky_re_lu	input:	(None, 28, 28, 1)
LeakyReLU	output:	(None, 28, 28, 1)

conv2d	input:	(None, 28, 28, 1)
Conv2D	output:	(None, 14, 14, 32)

leaky_re_lu_1	input:	(None, 14, 14, 32)
LeakyReLU	output:	(None, 14, 14, 32)

conv2d_1	input:	(None, 14, 14, 32)
Conv2D	output:	(None, 7, 7, 64)

leaky_re_lu_2	input:	(None, 7, 7, 64)
LeakyReLU	output:	(None, 7, 7, 64)

conv2d_2	input:	(None, 7, 7, 64)
Conv2D	output:	(None, 4, 4, 128)

leaky_re_lu_3	input:	(None, 4, 4, 128)
LeakyReLU	output:	(None, 4, 4, 128)

conv2d_3	input:	(None, 4, 4, 128)
Conv2D	output:	(None, 4, 4, 256)

flatten	input:	(None, 4, 4, 256)
Flatten	output:	(None, 4096)

dense_1	input:	(None, 4096)
Dense	output:	(None, 1)

A.2 Learning Performance Graphs

Figure 6: Learning performance for **Clients: 2, 3 ,5**

A.3 FMNIST Batches Per 10 Epochs

Figure 7: FMNIST FLGAN **Clients:2**

FMNIST FLGAN Final Outcome Clients:2

Figure 8: FMNIST FLGAN **Clients:3**

FMNIST FLGAN Final Outcome Clients:3
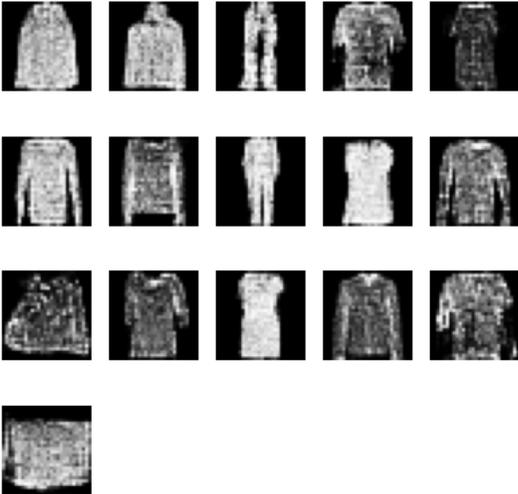
Figure 9: FMNIST FLGAN **Clients:5**
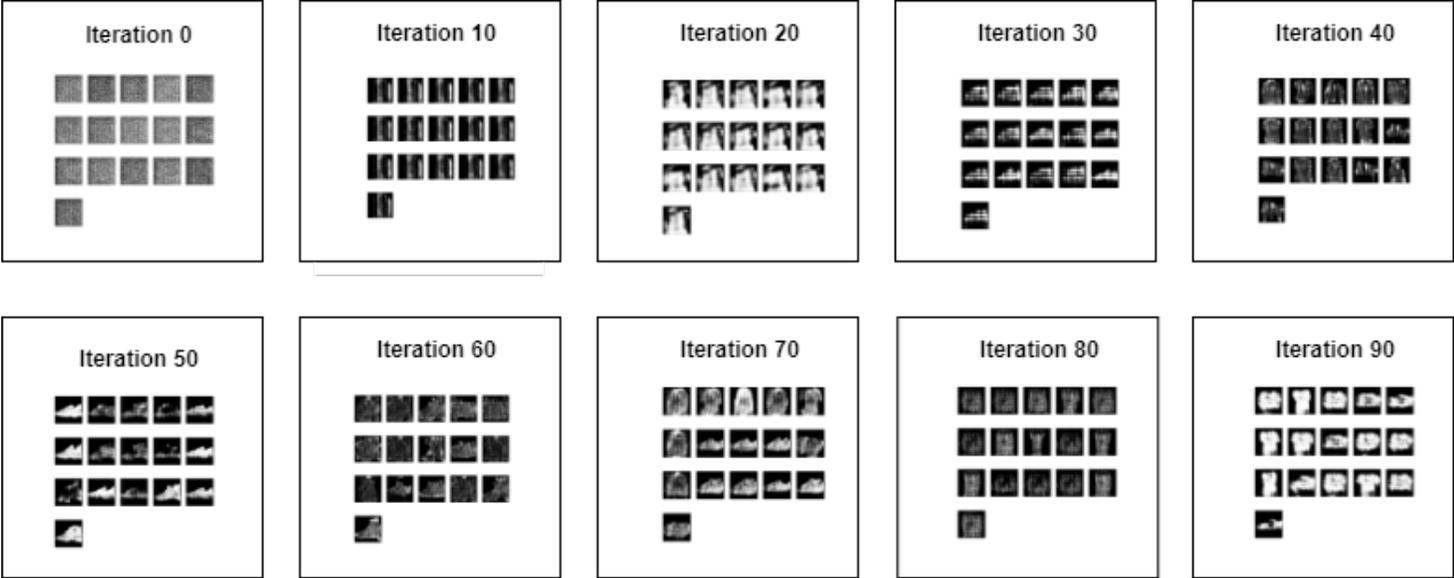
FMNIST FLGAN Final Outcome Clients:5
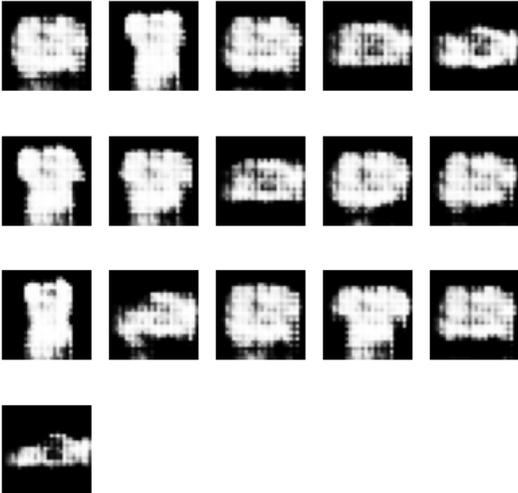
Figure 10: FMNIST FLGAN **Clients:10**
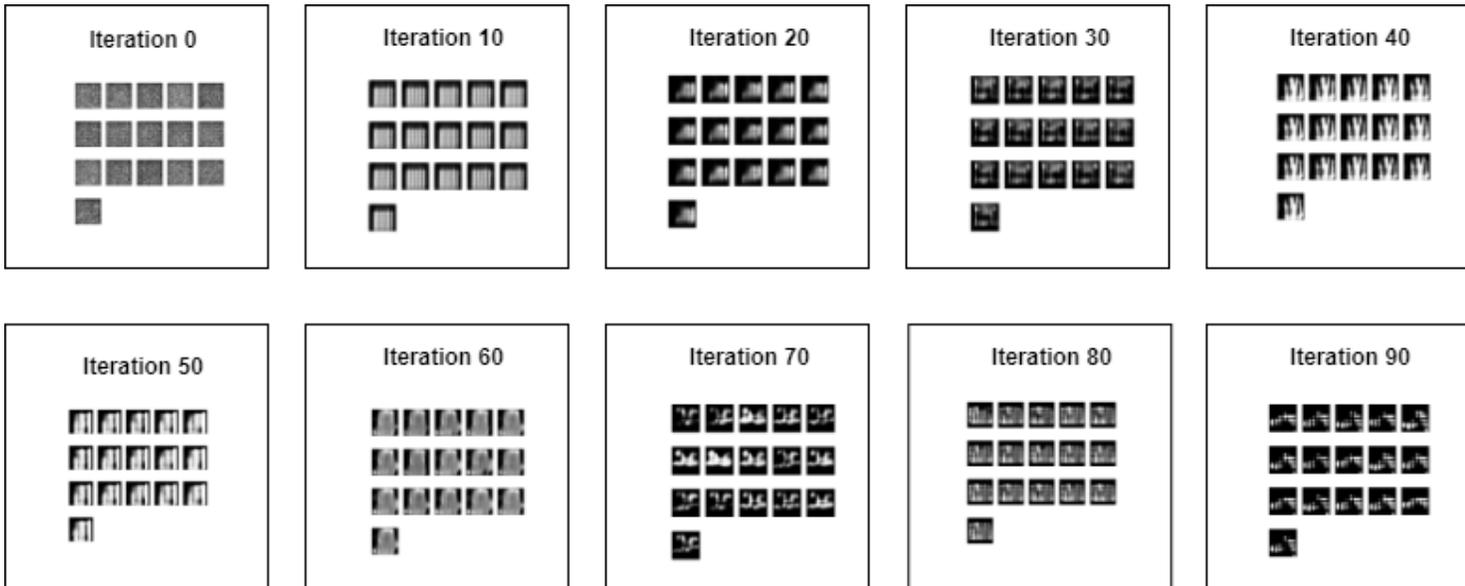
FMNIST FLGAN Final Outcome Clients:10
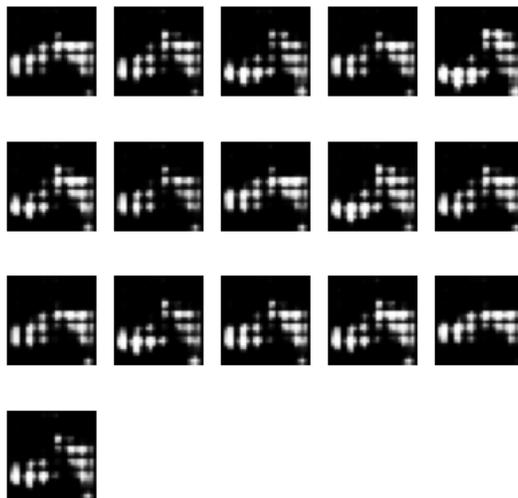
Figure 11: FMNIST FLGAN **Clients:20**
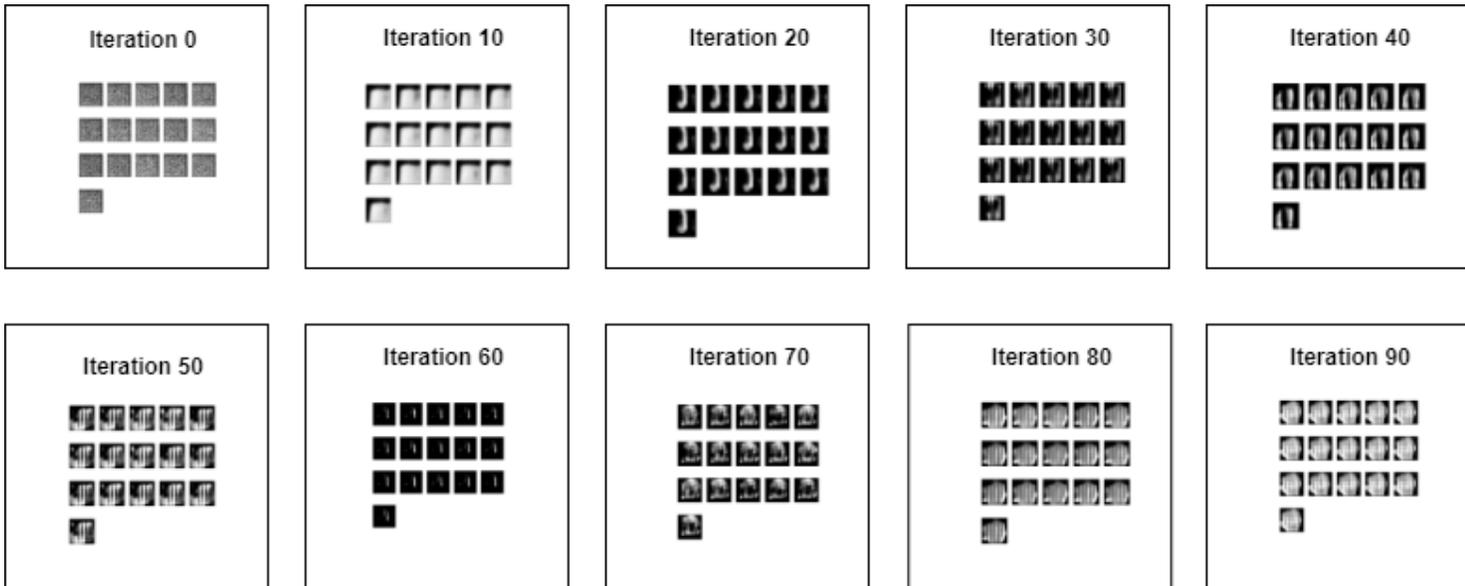
FMNIST FLGAN Final Outcome Clients:20
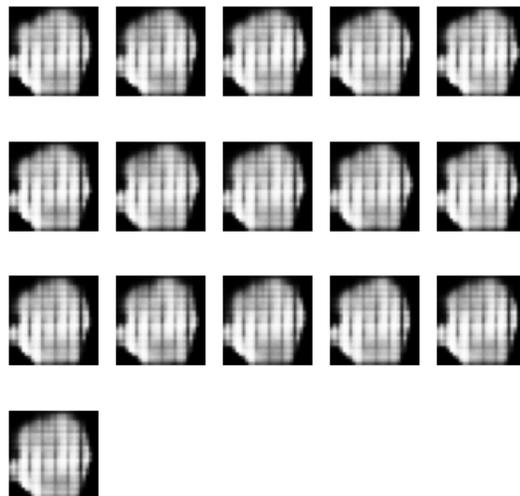
Figure 12: FMNIST AFLGAN **Clients:2**
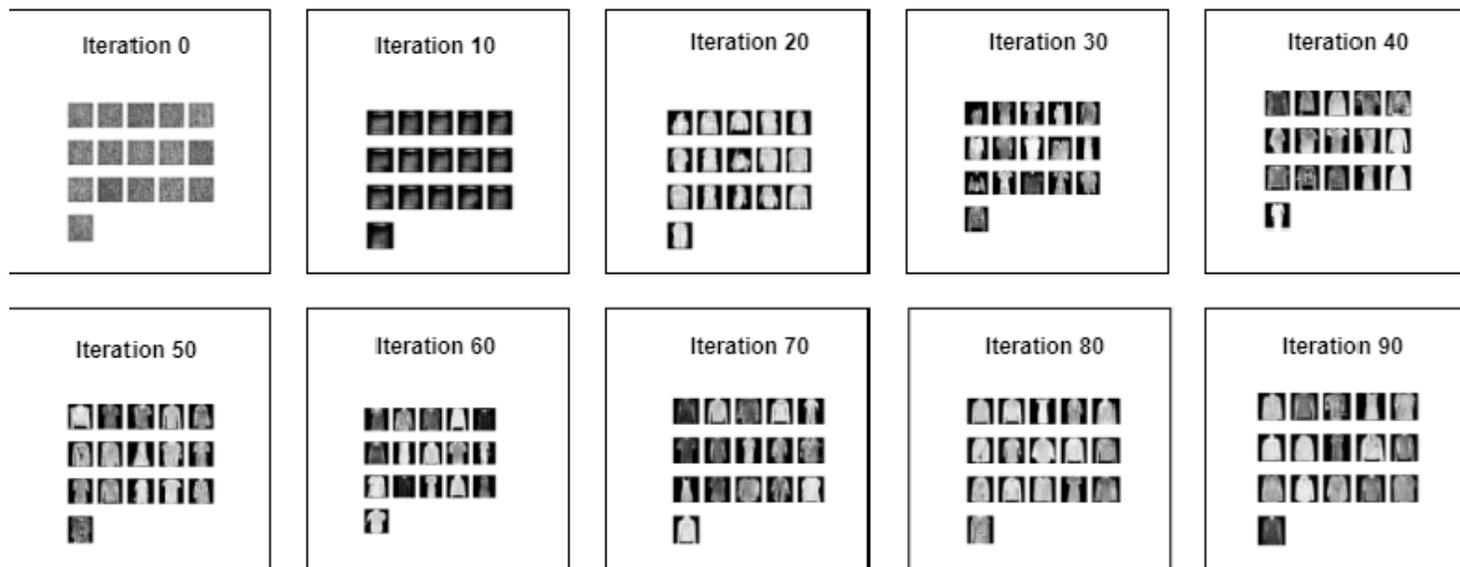
FMNIST AFLGAN Final Outcome Clients:2
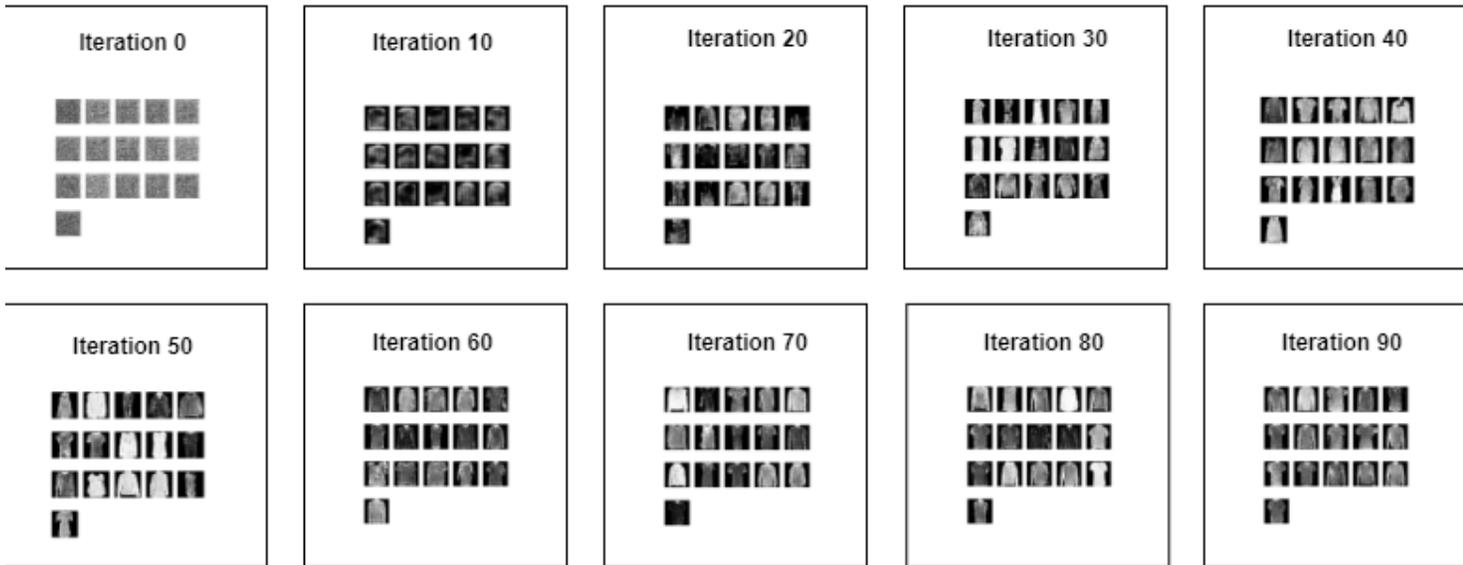
FMNIST AFLGAN Final Outcome Clients:3
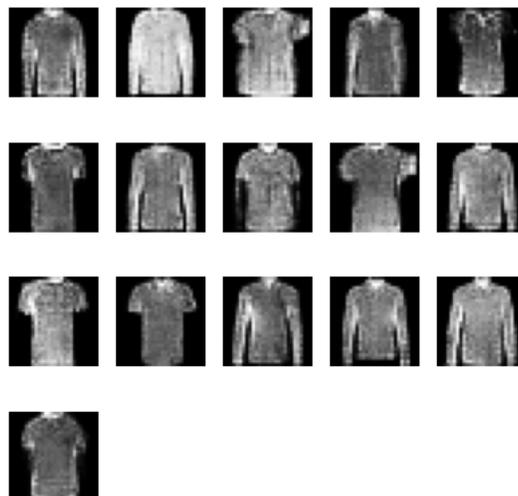
Figure 14: FMNIST AFLGAN **Clients:5**
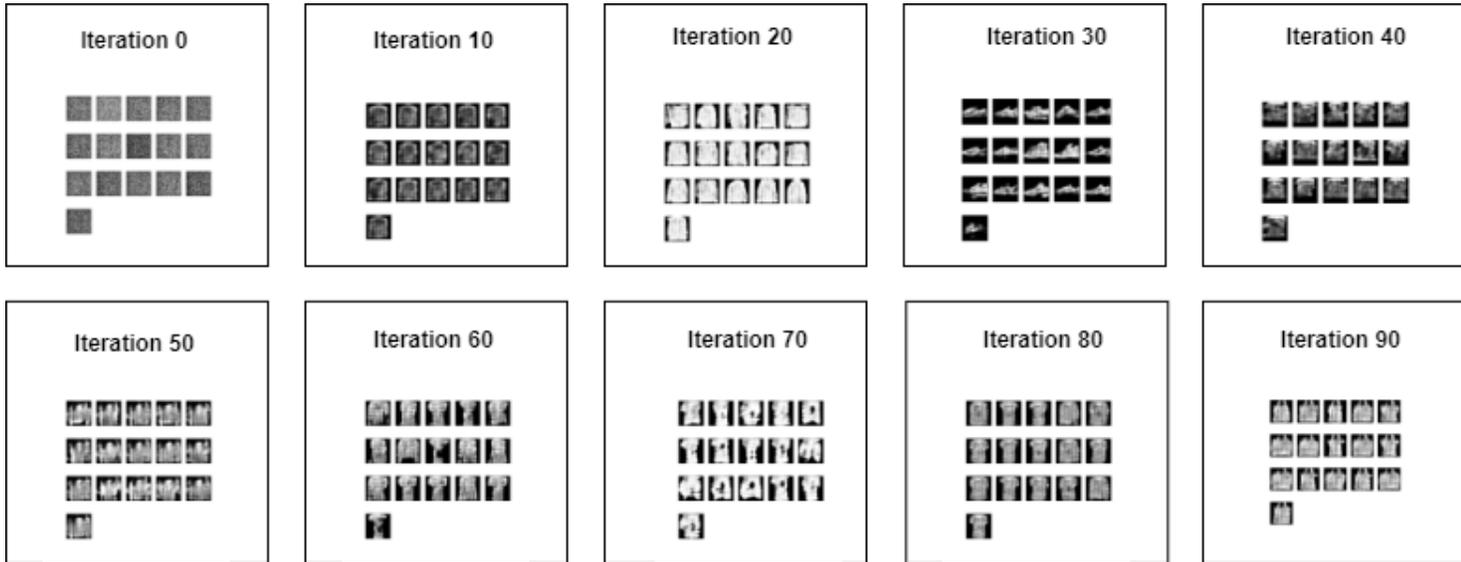
FMNIST AFLGAN Final Outcome Clients:5
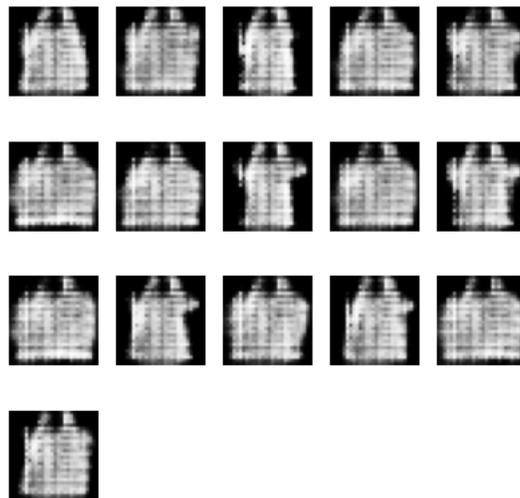
Figure 15: FMNIST AFLGAN **Clients:10**
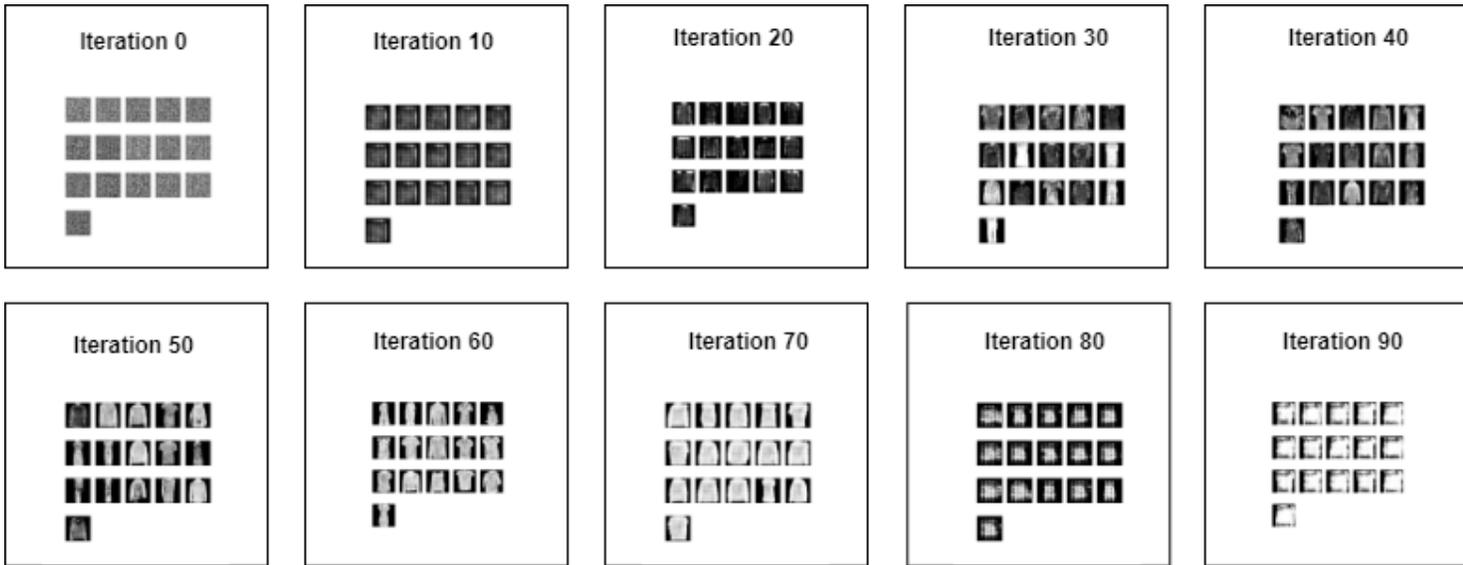
FMNIST AFLGAN Final Outcome Clients:10
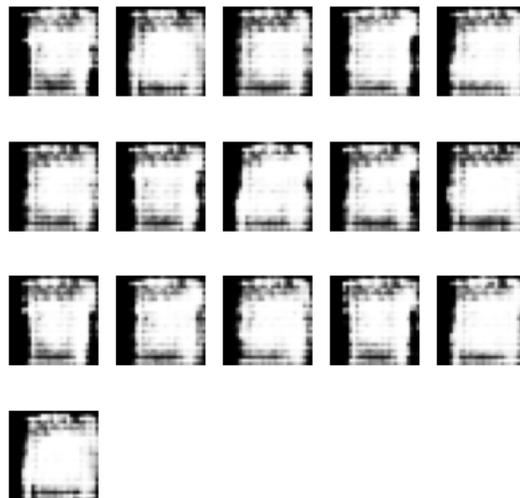
Figure 16: FMNIST AFLGAN **Clients:20**
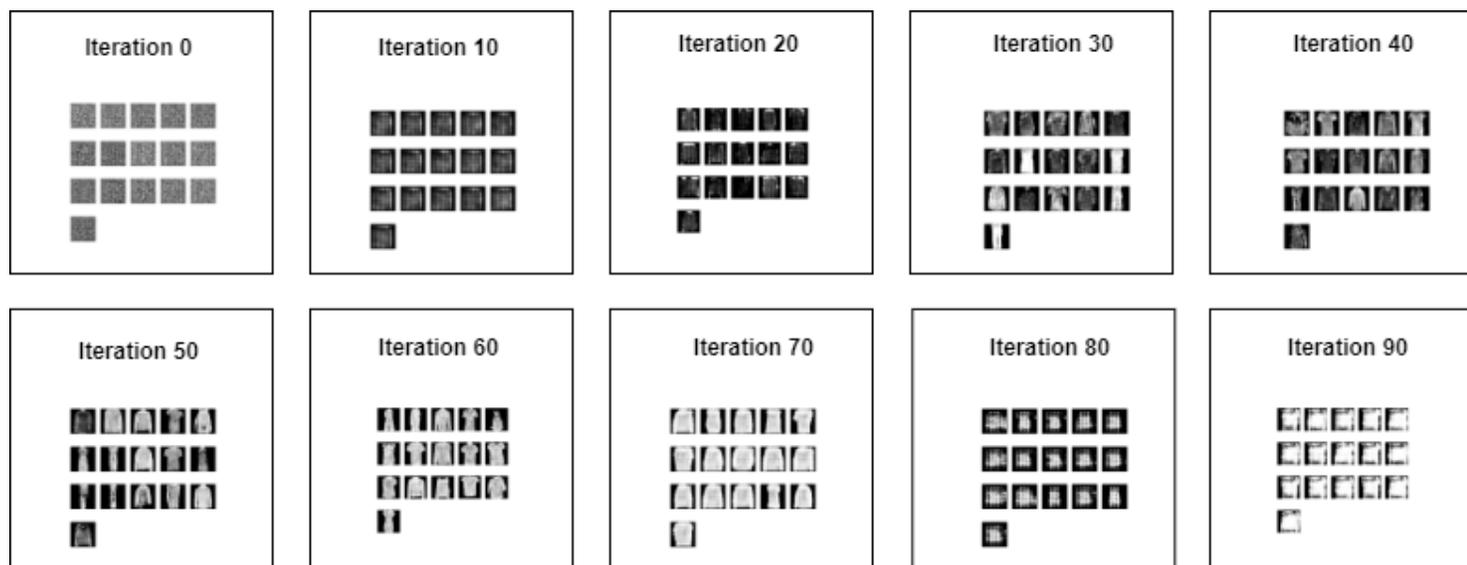
FMNIST AFLGAN Final Outcome Clients:20
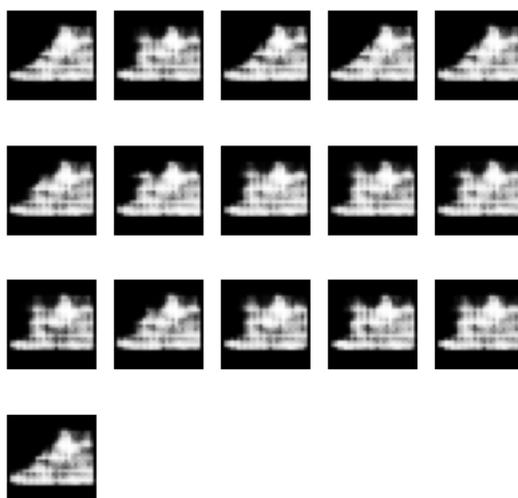
Figure 17: FMNIST MULTI-FLGAN **Clients:2**
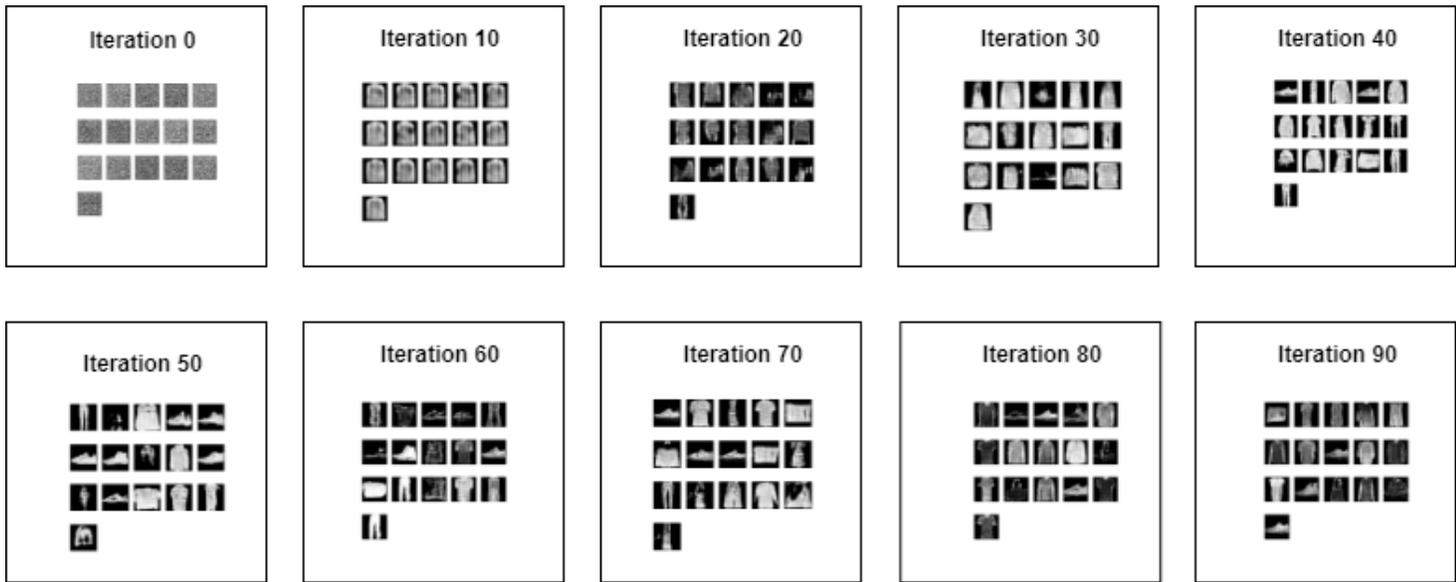
Multi-FLGAN Final Outcome Clients:2
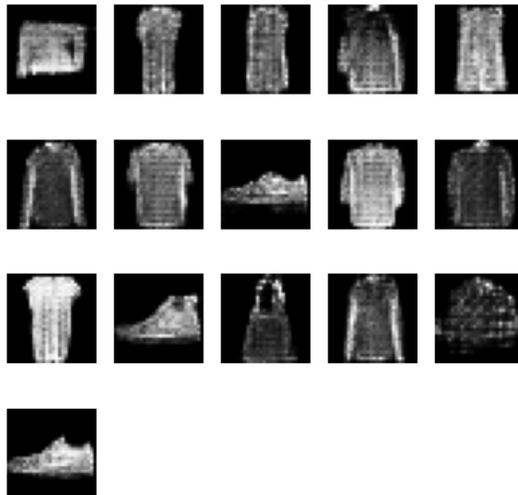
Figure 18: FMNIST MULTI-FLGAN **Clients:3**
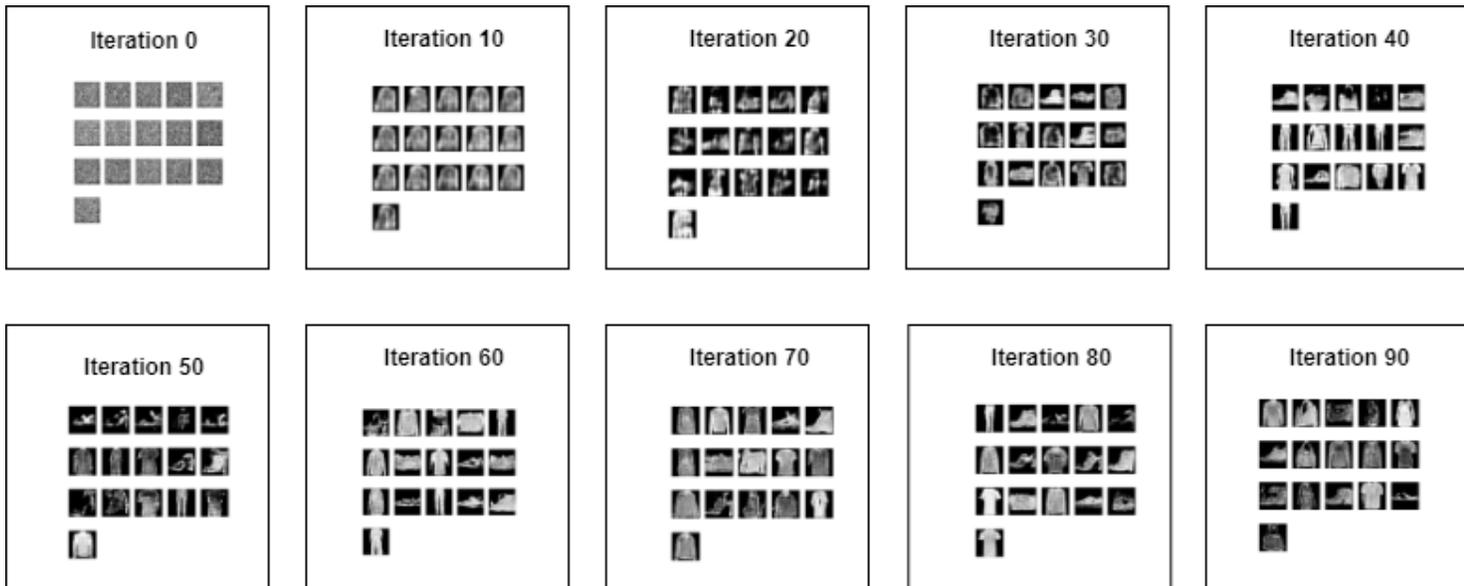
Multi-FLGAN Final Outcome Clients:3

Figure 19: FMNIST MULTI-FLGAN **Clients:5**
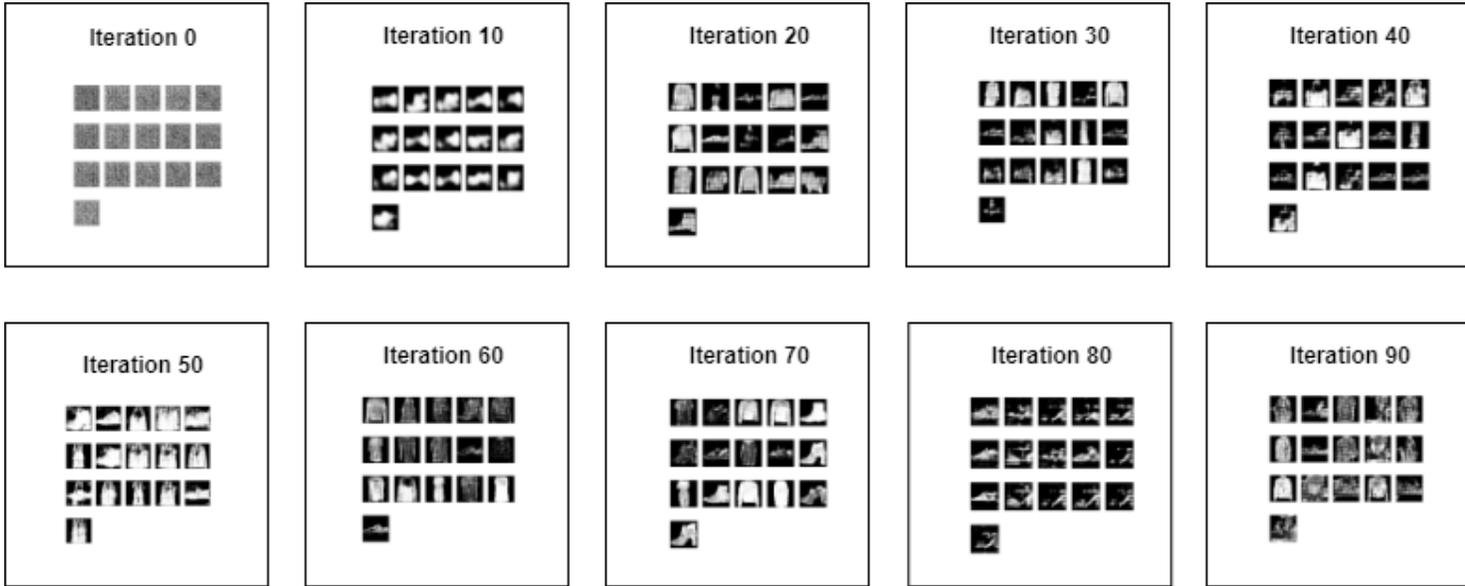
Multi-FLGAN Final Outcome Clients:5
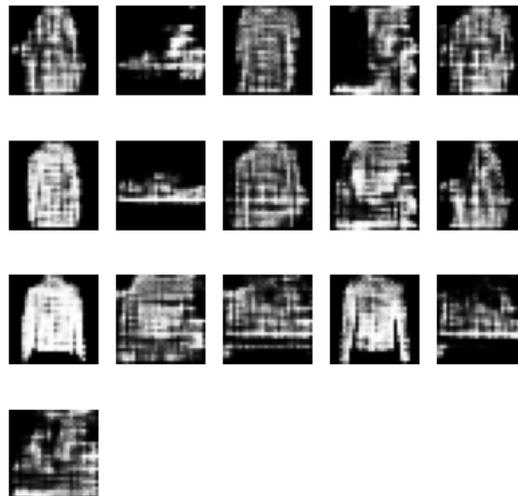
Figure 20: FMNIST MULTI-FLGAN **Clients:10**
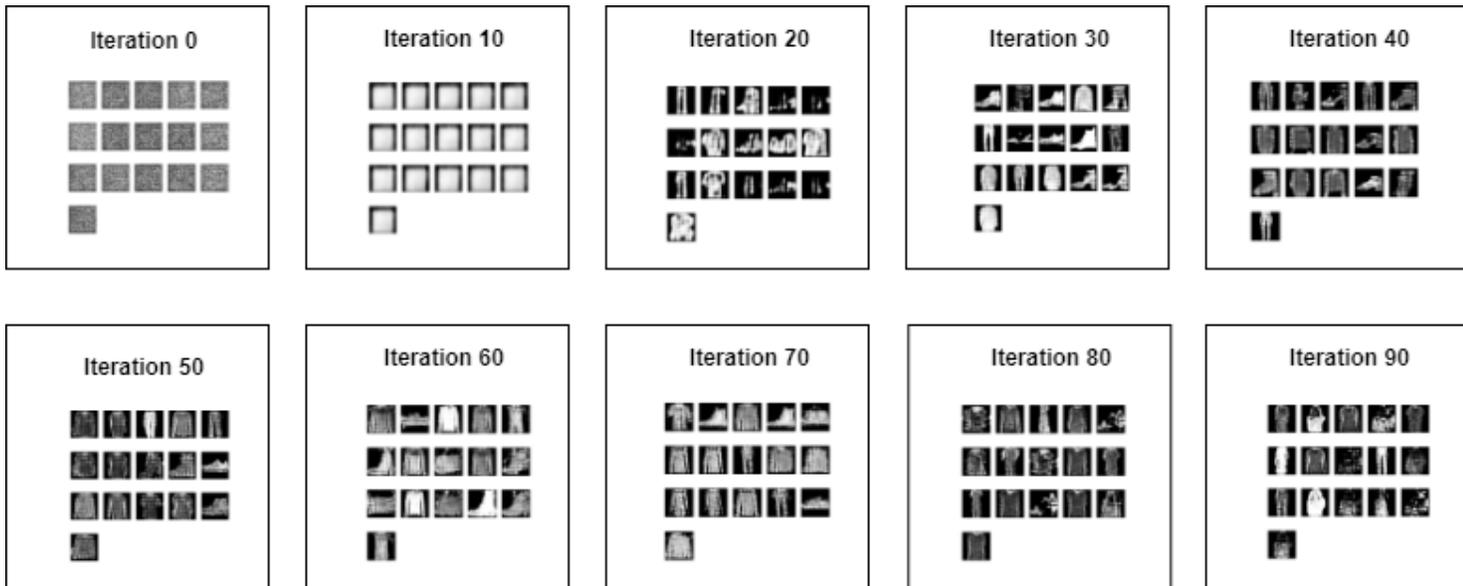
Multi-FLGAN Final Outcome Clients:10
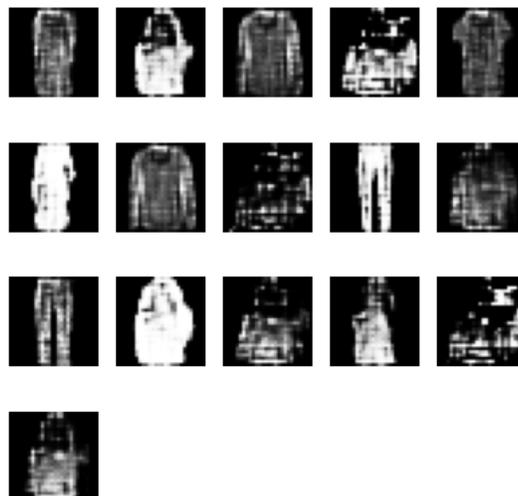
Figure 21: FMNIST MULTI-FLGAN **Clients:20**
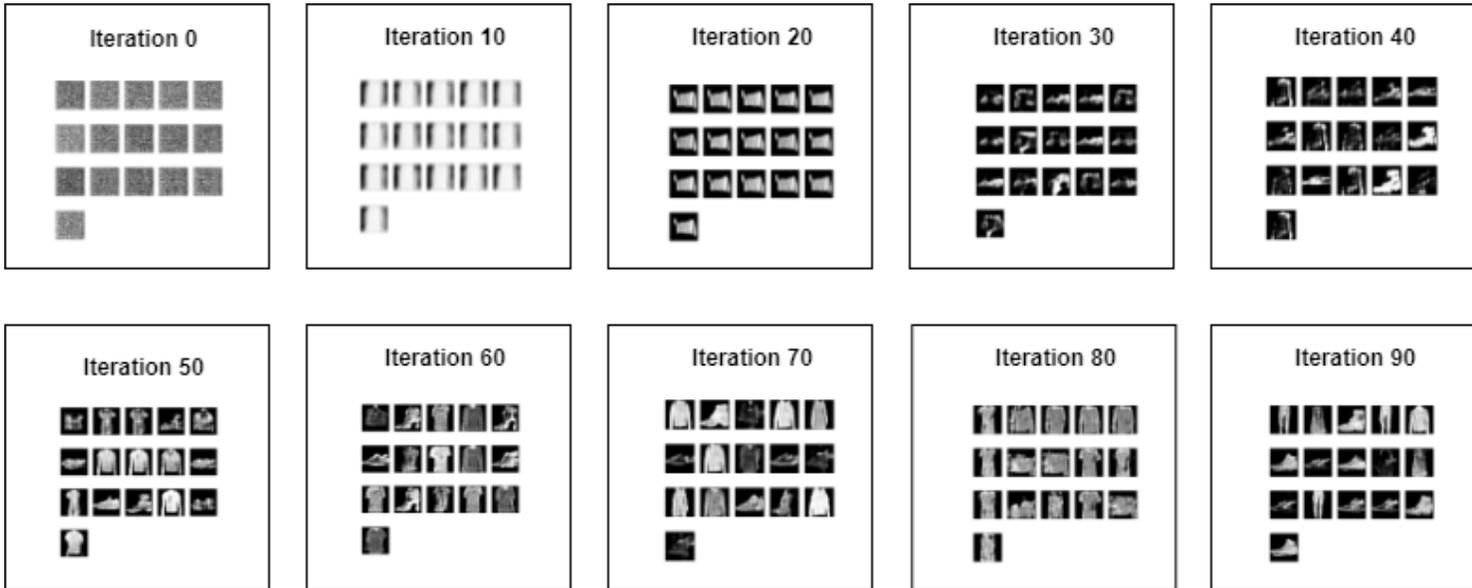
Multi-FLGAN Final Outcome Clients:20
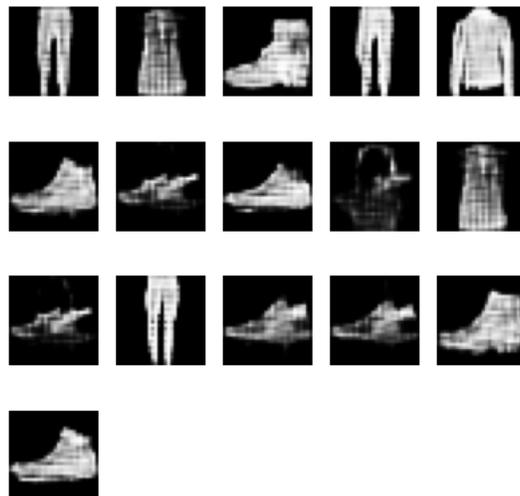
A.4 MNIST Batches Per 10 Epochs

Figure 22: MNIST FLGAN **Clients:2**
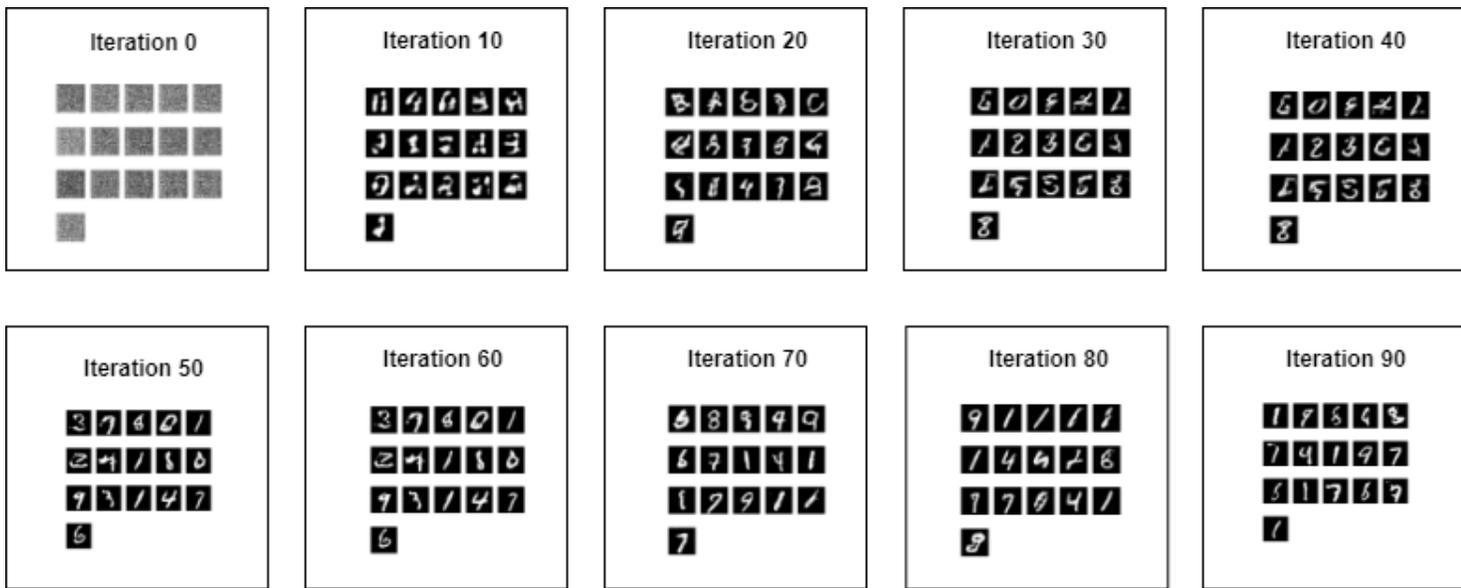
MNIST FLGAN Final Outcome Clients:2
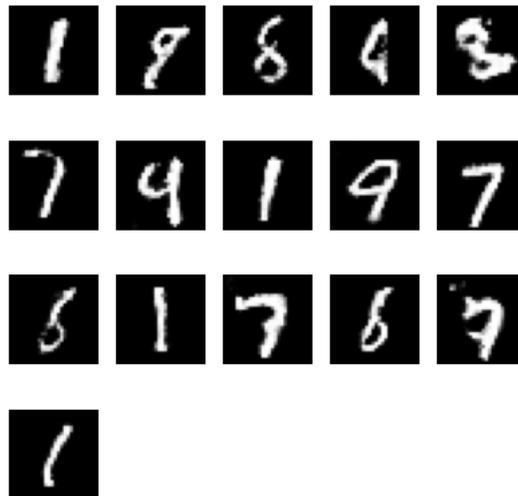
Figure 23: MNIST FLGAN **Clients:3**
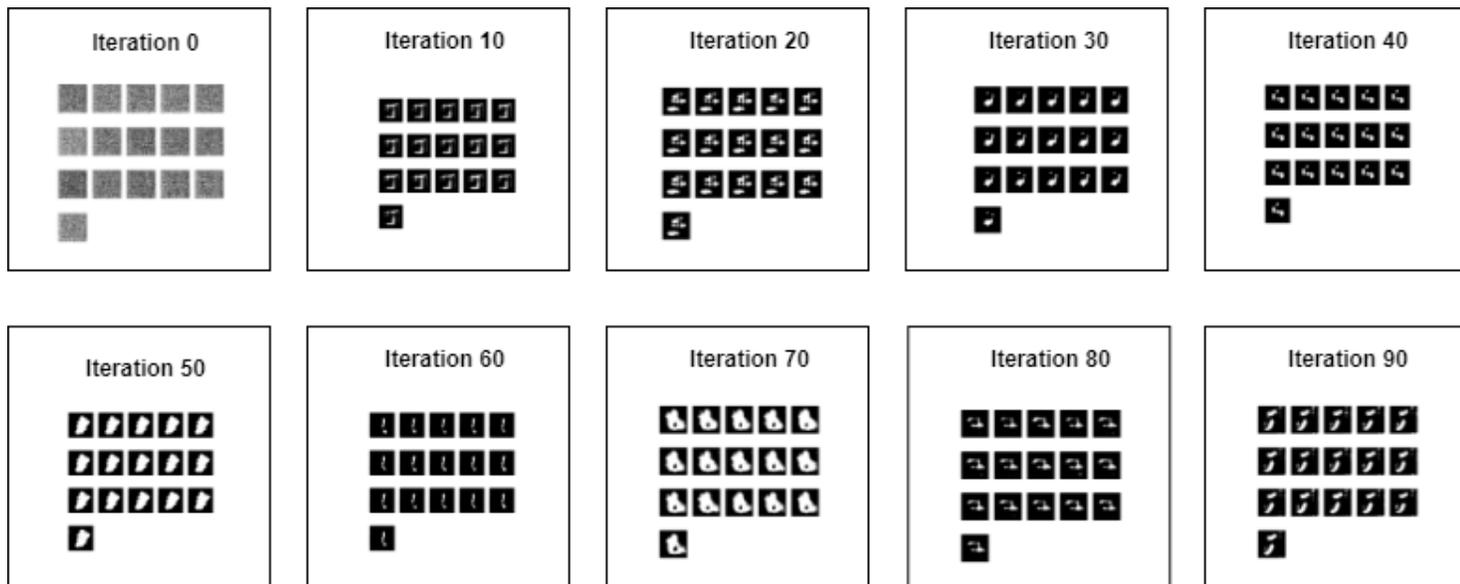
MNIST FLGAN Final Outcome Clients:3

Figure 24: MNIST FLGAN **Clients:5**
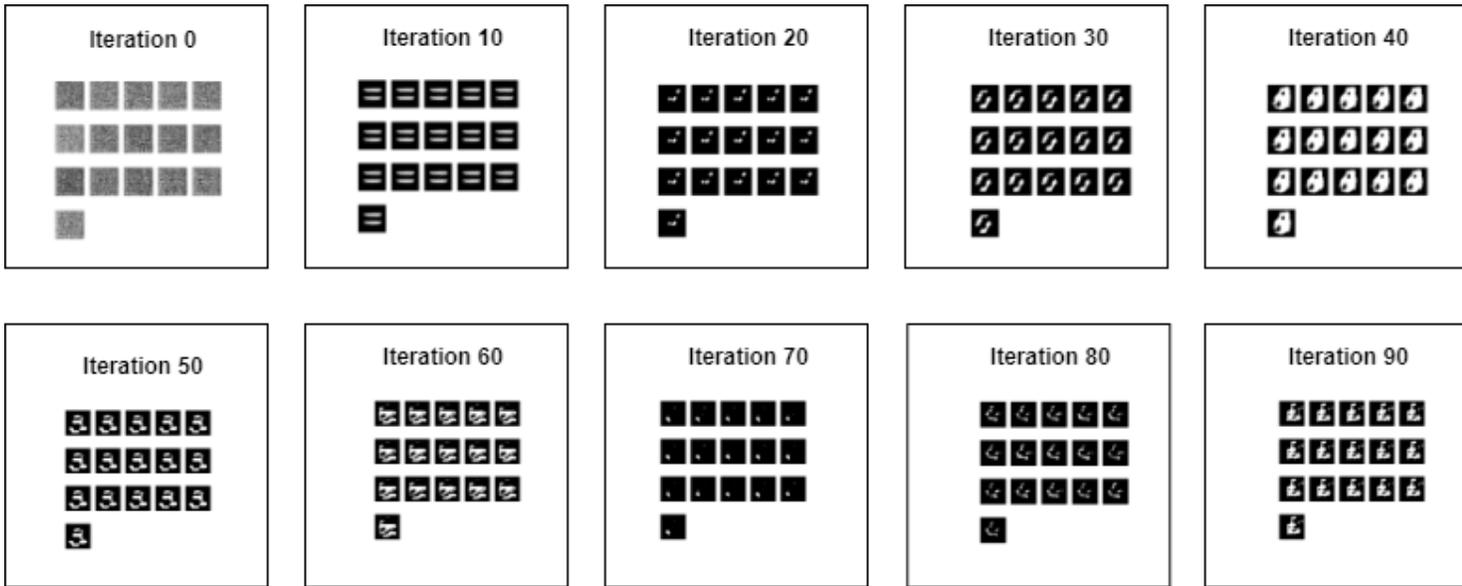
MNIST FLGAN Final Outcome Clients:5
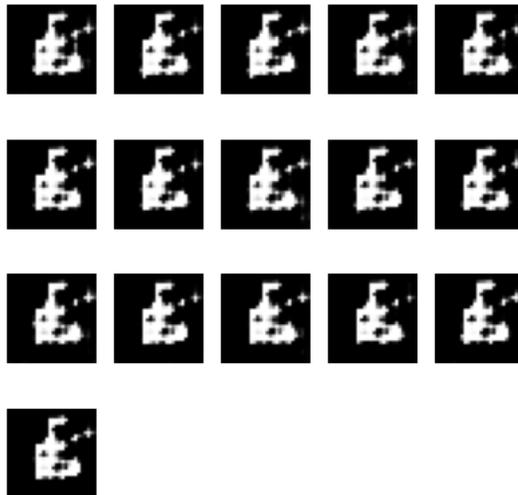
Figure 25: MNIST FLGAN **Clients:10**
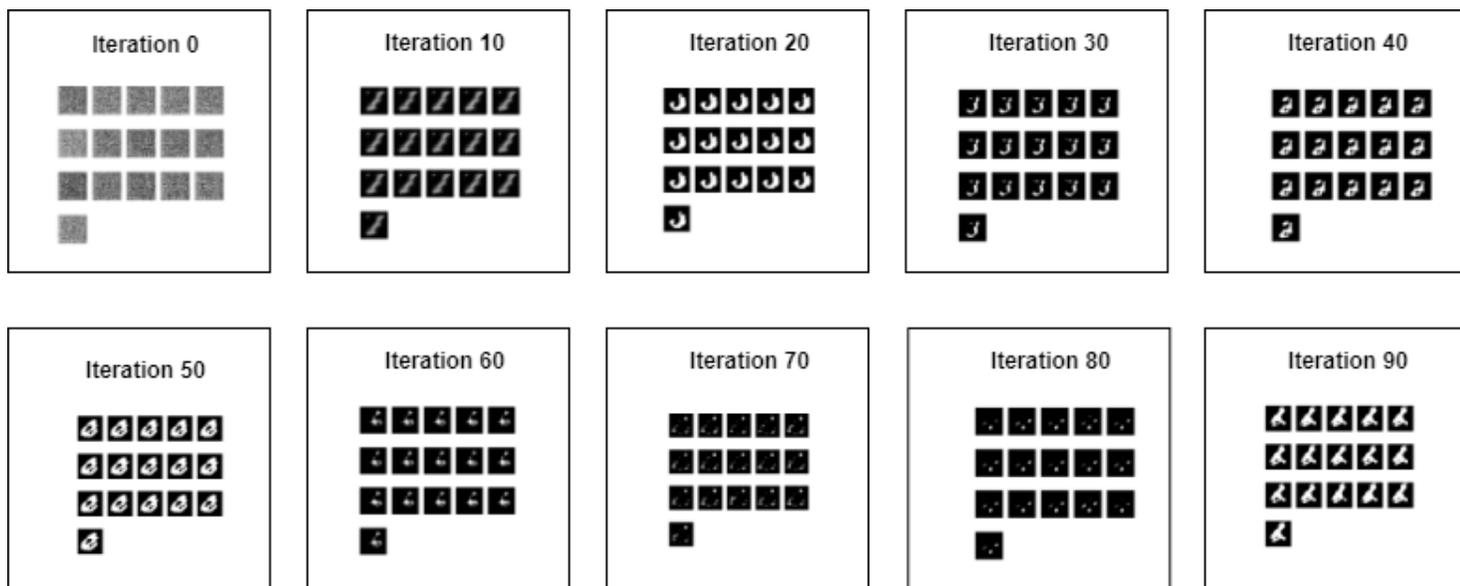
MNIST FLGAN Final Outcome Clients:10

Figure 26: MNIST FLGAN **Clients:20**
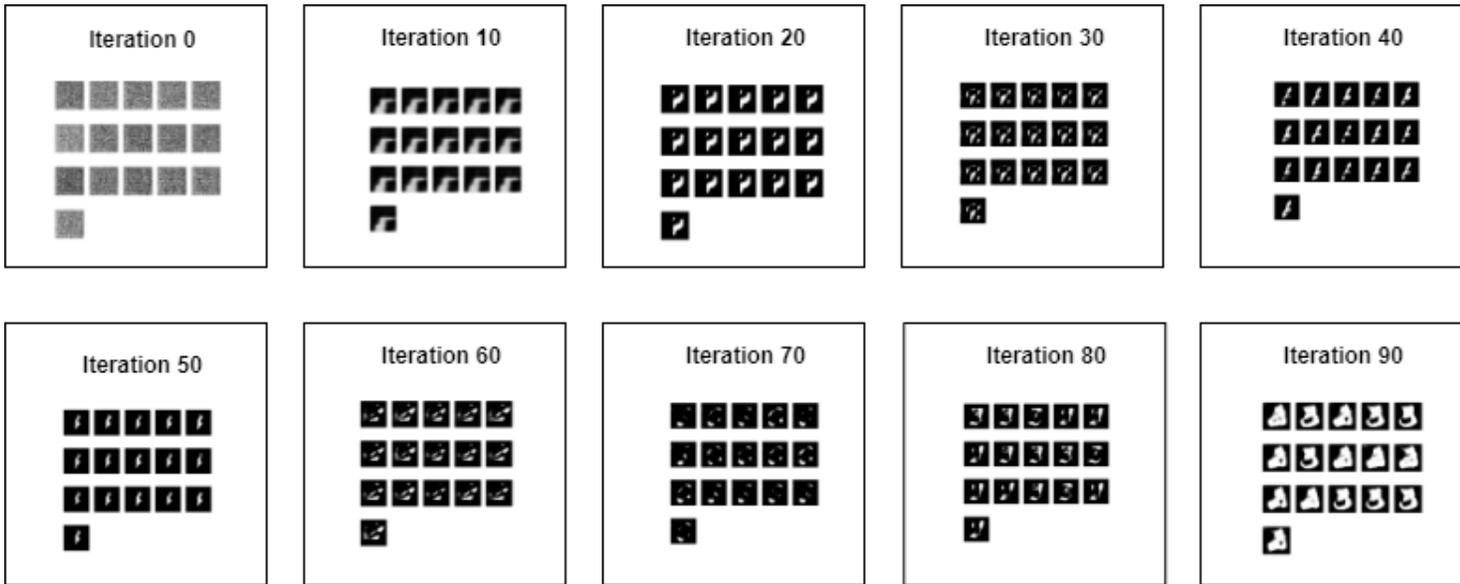
MNIST FLGAN Final Outcome Clients:20

Figure 27: MNIST AFLGAN **Clients:2**
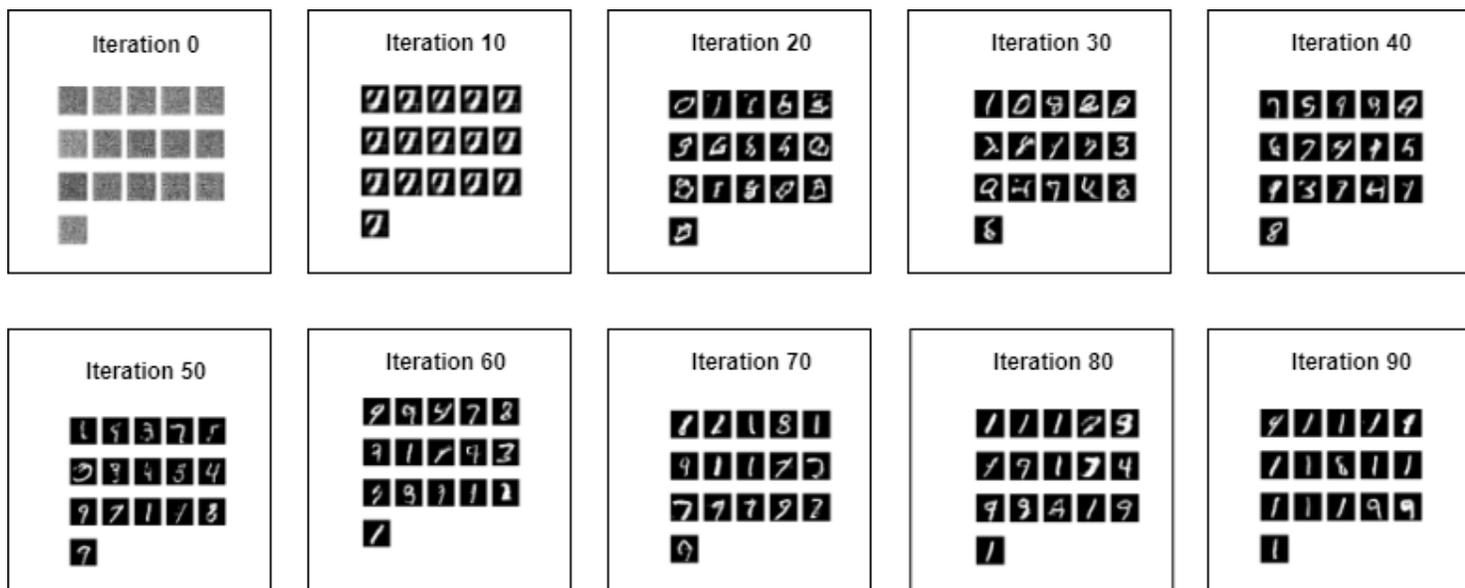
MNIST AFLGAN Final Outcome Clients:2

Figure 28: MNIST AFLGAN **Clients:3**
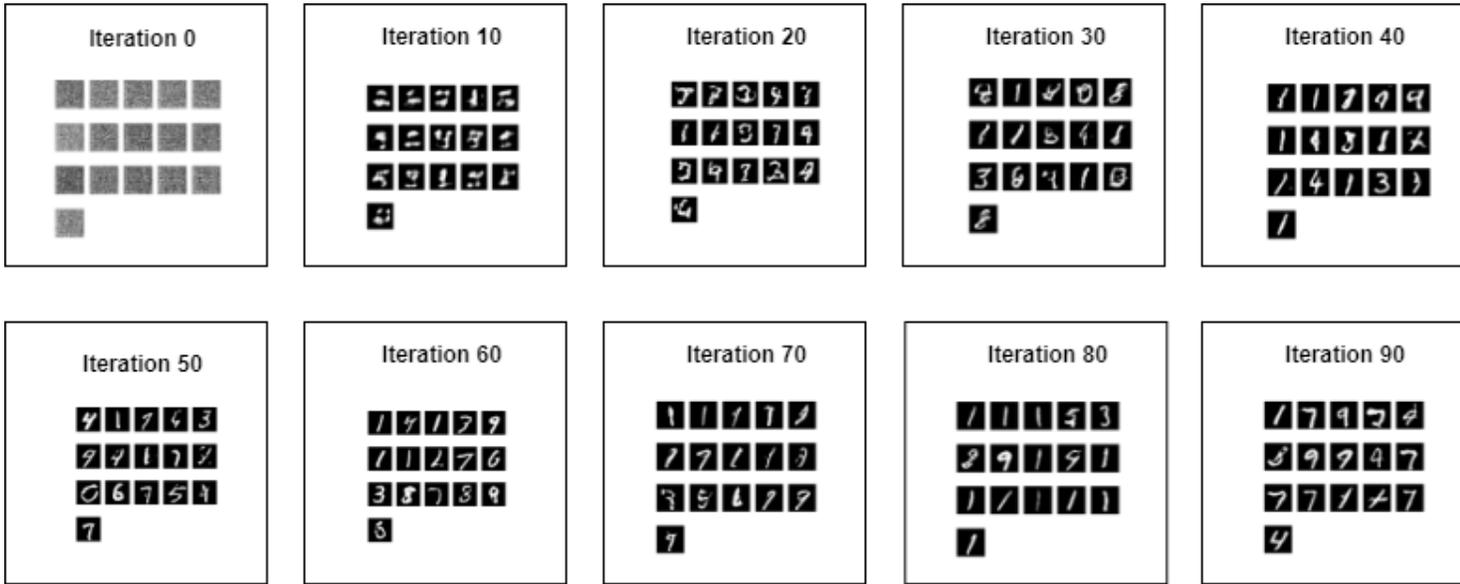
MNIST AFLGAN Final Outcome Clients:3
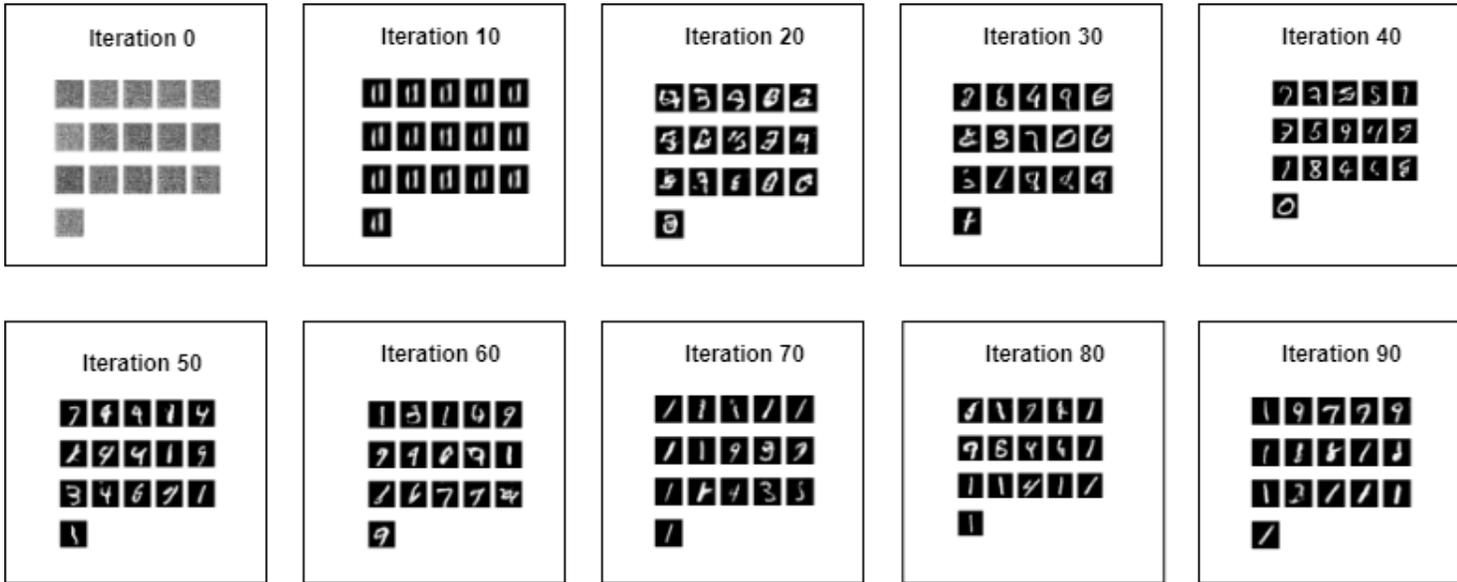
MNIST AFLGAN Final Outcome Clients:5
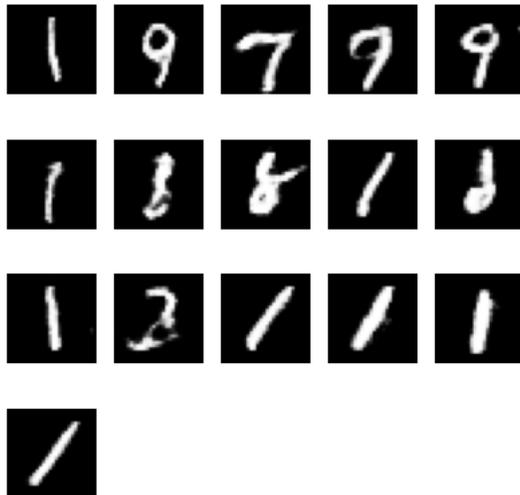
Figure 30: MNIST AFLGAN **Clients:10**
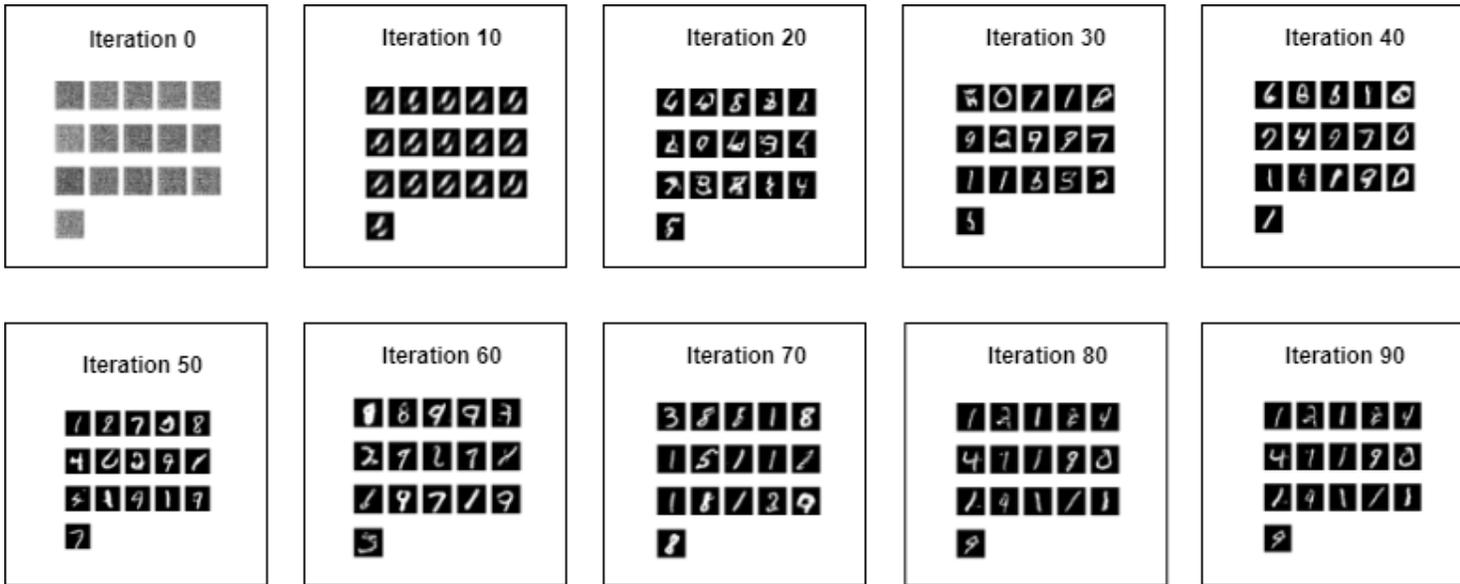
MNIST AFLGAN Final Outcome Clients:10

Figure 31: MNIST AFLGAN **Clients:20**
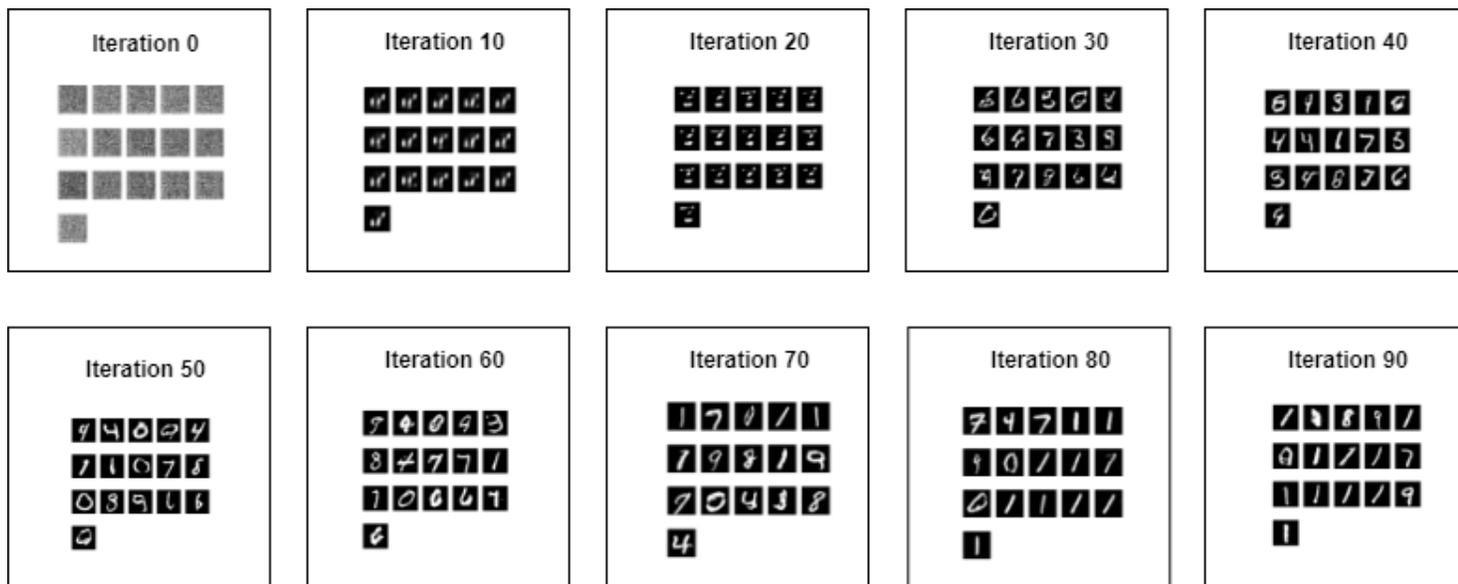
MNIST AFLGAN Final Outcome Clients:20
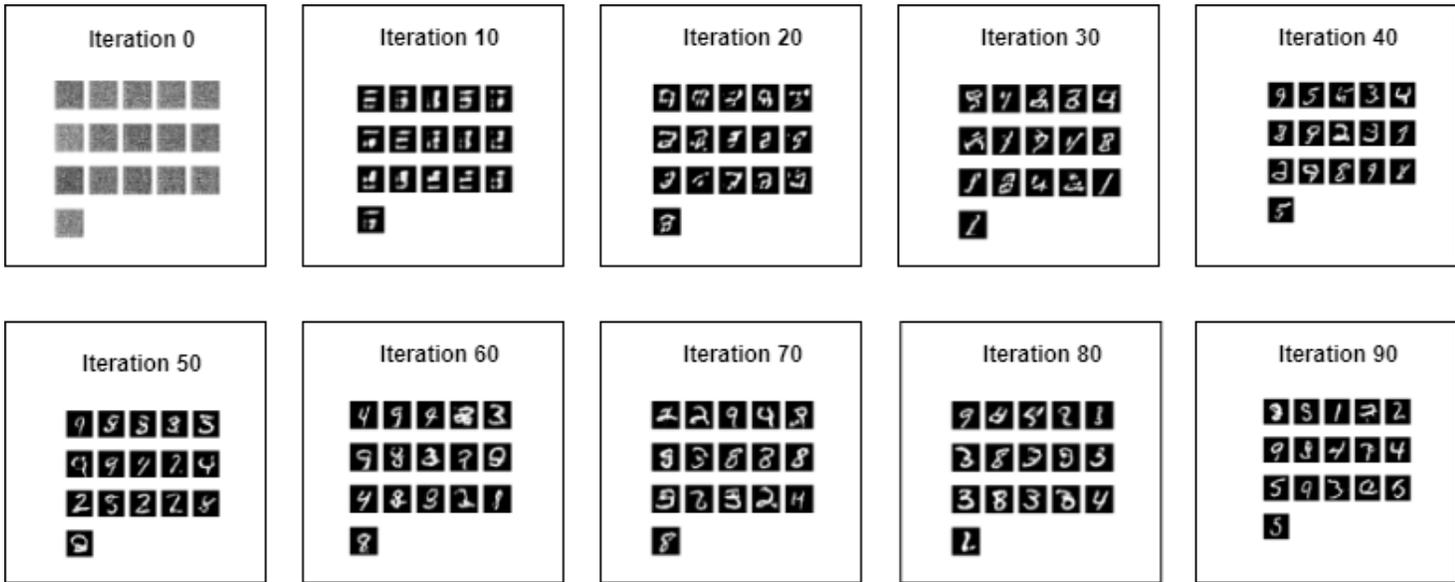
MNIST MULTI-FLGAN Final Outcome Clients:2
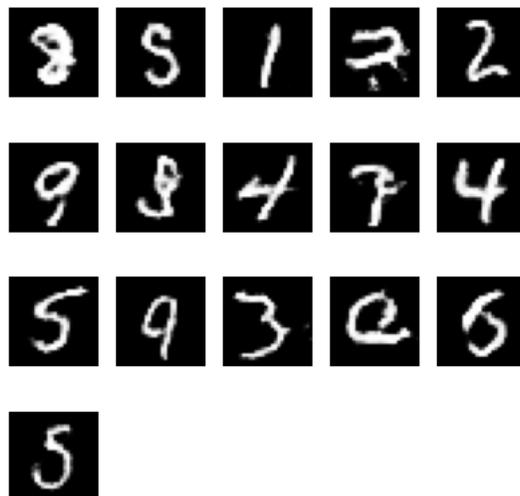
Figure 33: MNIST MULTI-FLGAN **Clients:3**
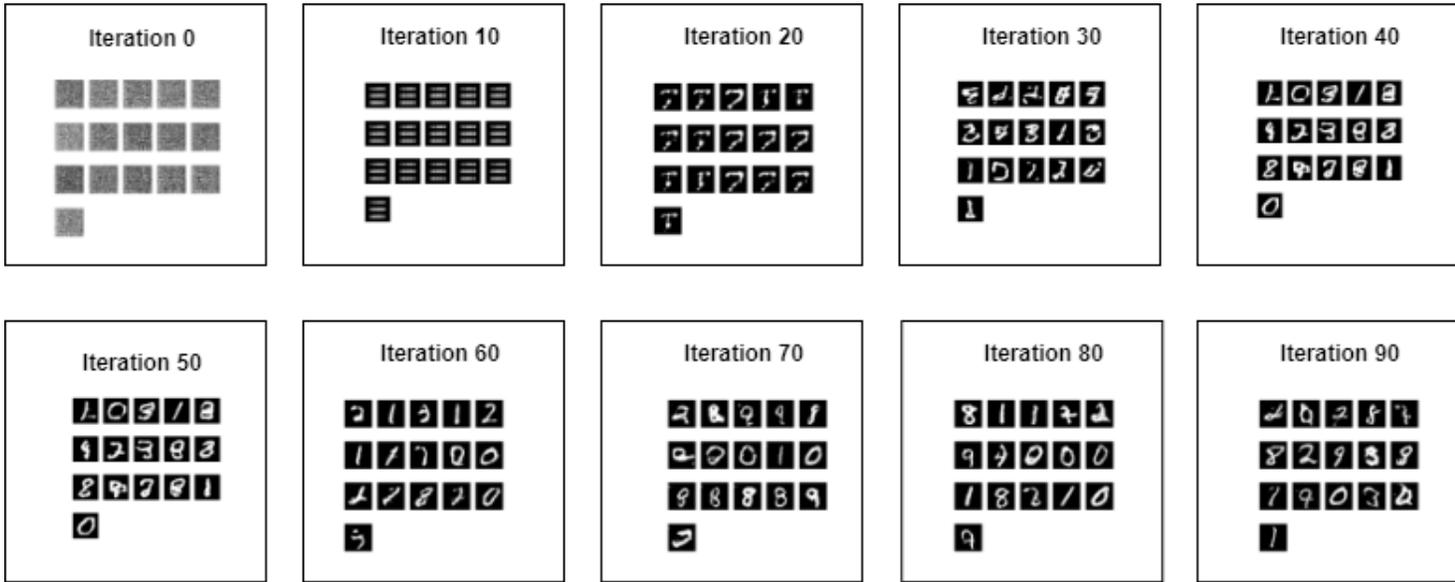
MNIST MULTI-FLGAN Final Outcome Clients:3

Figure 34: MNIST MULTI-FLGAN **Clients:5**
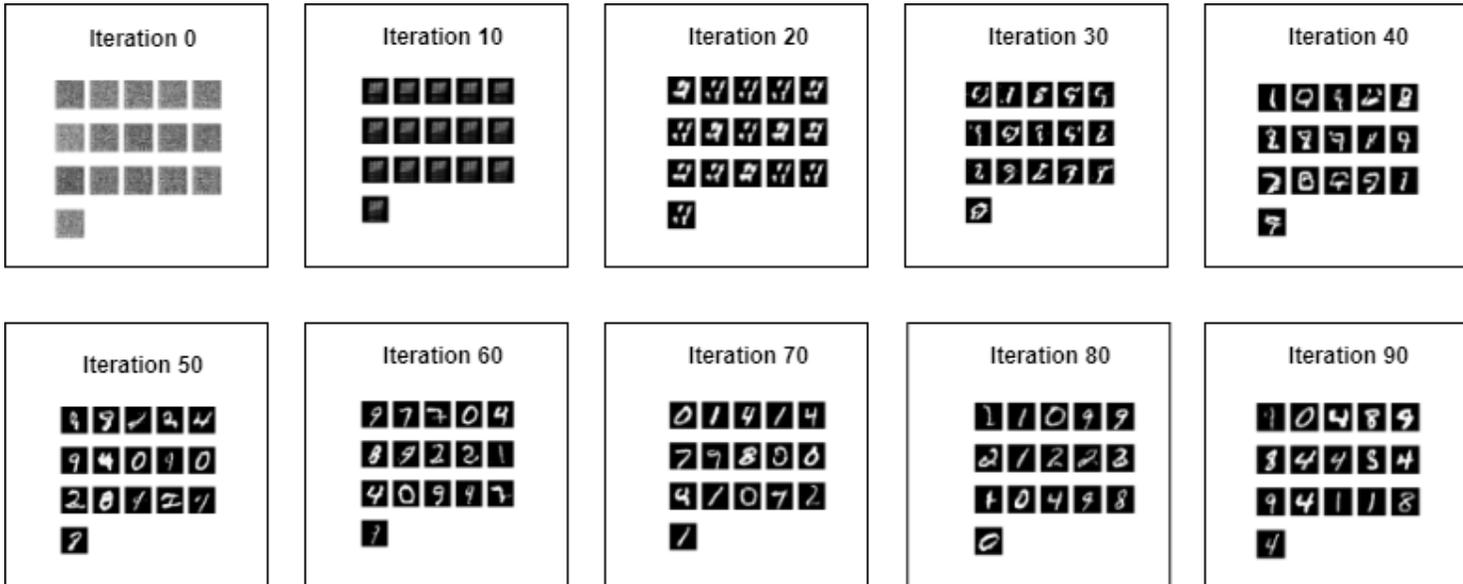
MNIST MULTI-FLGAN Final Outcome Clients:5
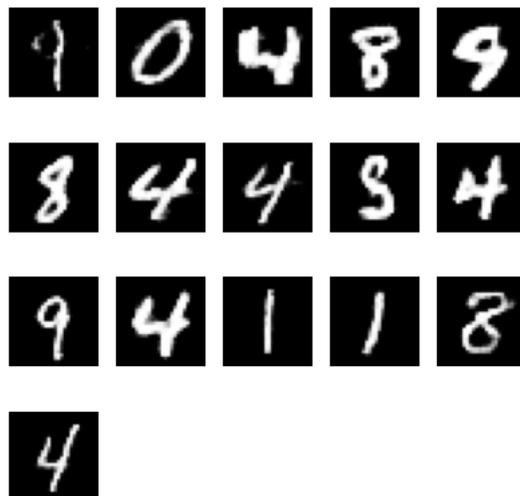
Figure 35: MNIST MULTI-FLGAN **Clients:10**
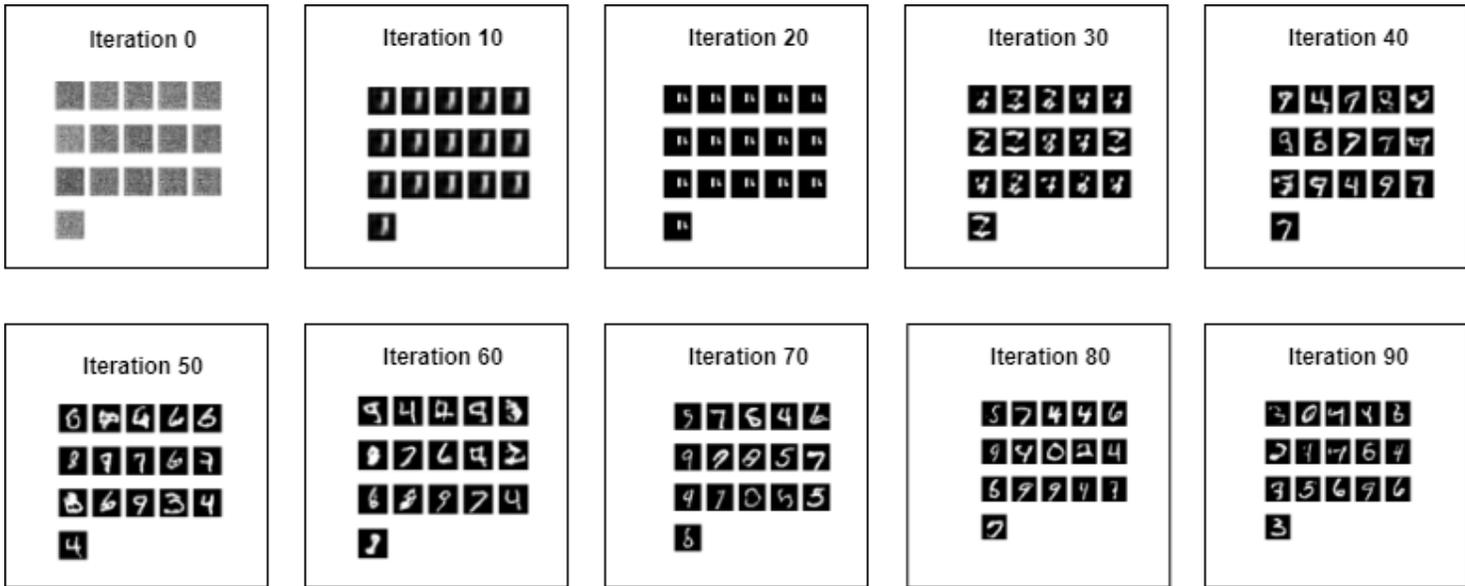
MNIST MULTI-FLGAN Final Outcome Clients:10

Figure 36: MNIST MULTI-FLGAN **Clients:20**

MNIST MULTI-FLGAN Final Outcome Clients:20