Exploring Federated Learning Aggregation Algorithms for the Creation of Synthetic Datasets Using Diffusion Models

Alexander Senden
University of Manitoba
Winnipeg, Manitoba, Canada
sendena@myumanitoba.ca

Abstract

Data privacy has become an increasingly important issue in the era of large machine learning (ML) models. Synthetic dataset generation and federated learning have been proposed as measures to increase data privacy, since each method allows for ML models to be trained without directly transferring real data. While each method has its limitations, we note that using the two techniques together helps mitigate their respective drawbacks. As a result, we propose a method of generating synthetic image datasets via diffusion models and federated learning, from which synthetic datasets of arbitrary size can be generated and distributed without sharing any real data. We then analyze two federated learning paradigms and evaluate the impact of six different aggregation algorithms on diffusion models, based on the quality of the resulting synthetic datasets and their effectiveness in training downstream machine learning models. Notably, a significant increase in execution time was discovered in a blockchain-based federated learning implementation due to a large network overhead. Additionally, the FedYogi aggregation algorithm outperformed the others, particularly for infrequent data classes, in settings where the data is not independently and identically distributed.

CCS Concepts

• **Computer systems organization** → **Fault-tolerant network topologies**; • **Security and privacy**; • **Computing methodologies** → **Distributed artificial intelligence**; **Computer vision**; **Learning settings**;

Keywords

Federated Learning, Aggregation Algorithms, Diffusion Models, Image Generation, Synthetic Datasets, Dataset Generation, Blockchain, Privacy

1 Introduction and Motivation

State-of-the-art diffusion models for image synthesis are among the many deep learning models that require vast amounts of data for training and fine-tuning. As a result of this necessity to accumulate massive amounts of training data, many large model researchers and research groups are either bottlenecked by data acquisition, or have come under scrutiny for potential data privacy violations. Data privacy and availability are major challenges when training or fine-tuning large deep learning models. However, these issues arise from a requirement of the traditional machine learning paradigm: the entity training the model must have full access to all of the training data. Federated learning (FL) is one approach which breaks away from this traditional paradigm [Konečný et al. 2016]. In FL, a distributed network of devices each train a local machine learning model using their local, private data, and communicates their learned updates with the broader network to construct a global model. In particular, a specific member or members of the network known as the "aggregator" collects the local updates, and aggregates them into the next iteration of the global model. In FL, no single entity controls or accesses all of the data; each node only shares parameter updates.

There are two further categorizations of FL. The first, hereby referred to as centralized FL, features decentralized training and centralized aggregation [Konečný et al. 2016]. That is, each trainer will train their local model on their local, private data, and then shares their parameter updates with a centralized aggregation server. In contrast, the second category of FL, hereby referred to as decentralized FL, features both decentralized training and decentralized aggregation, and is often coordinated using blockchain technology [Wang et al. 2024]. In decentralized FL, each trainer trains their local model on their local, private data, and then shares their parameter updates with the blockchain. The aggregators in the network then retrieve the updates from the blockchain, perform the aggregation, and then share the next iteration of the global model with the blockchain.

The decentralization of data in FL creates a unique challenge when compared to centralized learning: there is no guarantee that data samples are identically and independently distributed (i.i.d.) throughout the network. This means that the distributions from which each client's dataset is sampled may be significantly different, leading to different "ideal" optimizations at each device [Qi et al. 2024]. An active area of research within FL is finding model aggregation algorithms that quickly converge to a global optimum despite the non-i.i.d. data. FL under non-i.i.d. data slows convergence and potentially degrades the quality of the resulting model, in addition to the inherent overhead from the communication throughout the network [Konečný et al. 2016].

Recent improvements in generative artificial intelligence (AI) have led to alternative measures to overcome some instances of data scarcity. In this approach, a generative model can be trained or fine-tuned to produce data similar to the training set. The resulting synthetic data can then be used alone or in conjunction with the real data to train the downstream machine learning model [Trabucco et al. 2023].

This synthetic data technique can also be used to retain some data privacy. One entity may have full control over a dataset, and may consent to derivative works being distributed, but object to exact

Code for this project is available at:
https://github.com/alexsenden/federated-diffusion

elements of the dataset being shared. In this case, a generative model can be trained on the original dataset, and the resulting synthetic data, which ideally closely matches the real data distribution, can then be freely shared.

Each of these privacy-centric strategies have their own respective issues. The FL approach can be very application-specific and requires significant coordination between involved parties. Each party must run a training (and potentially aggregation) node configured to train the specific model. Due to the necessary communication and additional overhead incurred without more sophisticated coordination and initialization techniques, this type of FL is inadequate for any application that may involve experimentation or requiring multiple models to be trained. Furthermore, it requires any necessary data preprocessing to be performed by all parties; an infeasible task for particularly involved preprocessing such as image segmentation annotation. As for the generative approach, many generative models require significant amounts of data to train or fine-tune in order to create usable synthetic datasets, which makes it's problem recursive.

Despite their differences, the issues with these strategies have the potential to offset when used in conjunction. By training a generative model using FL, only one model needs to be trained, and the resulting generative model can be used to generate datasets for wide ranges of applications. In the domain of computer vision, this strategy could be used to create synthetic data for various tasks, such as image classification, segmentation, and object detection, by allowing for full access to the synthetic data (and by extension, its preprocessing). Then, the group-sourcing of data alleviates the data scarcity problem normally faced when training a generative model.

This raises a question: what is the difference in performance between centralized and FL approaches when training deep generative models, specifically, state-of-the-art image diffusion models? To date, there is limited research regarding the use of FL for training diffusion models [Balan et al. 2024; Goede et al. 2024; Stanley Jothiraj and Mashhadi 2024] and the corresponding effects of varying FL paradigms and aggregation algorithms.

As a result, this work aims to experimentally demonstrate the effects of FL paradigms and aggregation algorithms for the training of diffusion models for the purpose of synthetic image dataset creation. The three investigated paradigms are centralized learning, centralized FL, and decentralized FL. Within these paradigms, six different aggregation algorithms are individually explored. Finally, both the quality of the of the resulting dataset and the performance of a downstream deep learning model are examined.

2 Related Work

2.1 Diffusion Models

Diffusion models are a class of generative models that synthesize image data from samples of random noise using principles from non-equilibrium statistical physics known as Langevin dynamics. They were first introduced by [Sohl-Dickstein et al. 2015] with the intention to scale well with regards to parameters and computational requirements while maintaining simplicity in the training and sampling process. When training a diffusion model, training samples are incrementally converted to Gaussian noise via the repeated addition of noise, known as the forward process. The model then learns to iteratively denoise the image, known as the reverse process. As a result, noise can be easily sampled and input into the model to generate synthetic images from a distribution similar to the original training distribution.

Further improvements in diffusion models demonstrated their superiority to generative adversarial networks [Dhariwal and A. Nichol 2021; Ho et al. 2020; A. Q. Nichol and Dhariwal 2021; Song and Ermon 2019], another class of image-based generative model. This led to improved conditioning techniques [Dhariwal and A. Nichol 2021], including the ability for diffusion models to receive full-sentence input and demonstrate a semantic understanding of the prompt [Saharia et al. 2022].

The ability of diffusion models to create high-fidelity imagery similar to the target domain with the degree of control granted by sentence-length natural language inputs has led to their use and adaptation for dataset augmentation and synthetic dataset generation [Nguyen et al. 2023; Trabucco et al. 2023; Wu et al. 2023], improving downstream model capabilities beyond what was previously achievable when only using the real data.

2.2 Federated Learning

Federated learning is a distributed machine learning training paradigm in which the training is divided across multiple trainers who may have their own private local datasets [McMahan et al. 2017]. The models then communicate with a centralized aggregation server who compiles local training results to construct a global model reflective of the entire data distribution. The trainers never need to share their local data, however their local data contributes to the learning of a global model, thus enhancing data privacy. Subsequent areas of research within FL include improved data privacy [Wei et al. 2020], better aggregation algorithms [Qi et al. 2024], and purpose-built FL schemes [Kanagavelu et al. 2020; Kumar et al. 2022; Zhou et al. 2021].

The traditional FL paradigm requires a centralized aggregation server. In an attempt to decentralize the aggregation process, blockchained FL was introduced [Kim et al. 2020]. In blockchained FL, a blockchain network is used to store global state and allow for communication and consensus across peers in the network, replacing many of the responsibilities of the central aggregation server. This increase in decentralization led to a variety of different blockchained FL strategies.

[Wang et al. 2024] describes three broad classes of blockchain-based FL strategies: fully-, flexibly-, and loosely-coupled blockchained FL. In the fully-coupled case, the clients involved in the FL process also act as nodes in the blockchain. This means that the same peers involved in training the model also participate in verifying and generating new blocks. In this case, the ledger itself generally contains the training data and all copies of the local and global updates, although some implementations still allow for local datasets [Hua et al. 2020]. A frequent problem with fully-coupled implementations is the additional network and storage overhead that is often associated with the reliance on the distributed ledger for the storage of large models.

Flexibly-coupled blockchained FL implementations significantly differ from fully-coupled architectures in network topology. In the flexibly-coupled case, clients of the FL process are not nodes within

the blockchain, and therefore do not participate in any blockchain lifecycle processes. Instead, the clients within the network simply perform training (and potentially aggregation, depending on the implemented strategy) and submit their results to the blockchain. That is, the blockchain's distributed ledger stores information regarding the the FL process, but the nodes in the blockchain network do not directly participate in the learning process. In some implementations, miners may participate in the aggregation process. The separation of responsibilities can help increase data privacy and decrease network congestion, however can lead to issues in constructing, configuring, and coordinating the network.

The final class, loosely-coupled blockchained FL, further removes FL responsibilities from the blockchain. Similar to the flexibly-coupled approach, clients in loosely-coupled implementations perform all of the training, and additionally perform all of the aggregation. The role of the blockchain is to generate reputation opinions for trainers in the network, distributing rewards and penalties as necessary. These reputations are communicated with the aggregators to influence the result of the aggregation algorithm. In this scenario, the blockchain and FL execution are almost entirely independent.

2.3 Federated Training of Diffusion Models

The recent advancement of diffusion models and their recognition as state-of-the-art methods for image synthesis [Dhariwal and A. Nichol 2021] has led to their training using FL. In [Stanley Jothiraj and Mashhadi 2024], the authors developed Phoenix, a novel FL strategy for the purpose of training diffusion models. Phoenix involves data sharing to help align the non-i.i.d. local data distributions. In their research, they trained and compared a denoising diffusion probabilistic model [A. Q. Nichol and Dhariwal 2021] trained with both i.i.d. and non-i.i.d. data, and found that their novel algorithm was able to outperform more traditional FL approaches.

A major issue with the federated training of large models is the network overhead caused by the exchange of the millions of model parameters. This was the motivation for the work in [Goede et al. 2024], where a denoising diffusion probabilistic model was trained using an adapted FedAvg algorithm to exchange fewer parameters. [Goede et al. 2024] demonstrated that this parameter-efficient training and communication strategy led to similar performance to baseline models, however the performance began to drop off upon the inclusion of non-i.i.d. data and increased trainers in the federation. Overall, [Goede et al. 2024] was able to reduce the number of communicated parameters by up to 74%.

Training data memorization can be a problem for generative models, where synthetic samples closely resemble training data. [Balan et al. 2024] introduced PDFed, a novel blockchain-based FL framework designed to limit data memorization in diffusion models. This was achieved using a new quality-novelty metric, which was used to modify the training process of local trainers in the network. This approach was shown to reduce data memorization in a class-conditional denoising diffusion probabilistic model.

The diffusion models used in these works are relatively small in size. The model used by [Stanley Jothiraj and Mashhadi 2024] and [Balan et al. 2024] is approximately 35 million parameters, whereas the model used by [Goede et al. 2024] is approximately 3 million parameters.

3 Methodology

This experiment is designed to better understand the effects of both centralized and decentralized FL compared to regular centralized machine learning, including the impacts of various federated aggregation algorithms. In particular, the effects had on a fine-tuned diffusion model for the purpose of synthetic image dataset creation. This includes both the quality of the generated images, and the performance of a downstream deep learning model trained with the synthetic dataset.

3.1 Learning Paradigms

Three different learning paradigms will be explored. They are: centralized learning, centralized FL, and decentralized FL. A summary of the workflows for each approach are displayed in figure 1.

3.1.1 Centralized Approach. The centralized approach will be used to create a baseline model for comparison with the FL models. In this approach, a diffusion model will be trained using traditional techniques such as oversampling to counteract class imbalances and AdamW to optimize the weights [Loshchilov and Hutter 2017].

3.1.2 Centralized Federated Learning Approach. In centralized FL, there are n decentralized trainers, each with their own local, private dataset. That is, there are n distinct partitions of the global dataset. These datasets may be of varying size and content, and may contain significant class imbalances both within the local partition and across the global dataset as a whole. For example, *Class A* may be significantly overrepresented in *Partition A* while being underrepresented in the global dataset.

The centralized FL workflow proceeds in discrete intervals referred to as "server rounds". At the beginning of each server round, a subset of trainers are selected to form the next iteration of the global model. For the remainder of the server round, the unselected trainers remain idle. Once selected, trainers download the latest version of the global model weights from the centralized aggregation server. This global model is either the set of weights aggregated from the previous server round, or if this is the first server round, a set of newly-initialized weights from the central aggregation server.

The trainers have now been selected and all contain the same model parameters. Each trainer then performs a predefined number of local training steps, or a predefined number of training epochs on their local dataset, depending on the configuration. This local training process is identical to a regular, centralized machine learning training process. Upon completion of the local training steps, the trainer will send its updated weights to the centralized aggregation server.

Once all selected trainers have sent their weights to the centralized aggregation server, the aggregator performs the aggregation. The aggregation process is the most frequently modified component of the FL process. The result of the aggregation algorithm is a new set of global weights derived from the n collected local weights. This completes one server round, and the process then repeats until some predetermined criteria is met, such as the completion of a predefined number of server rounds.

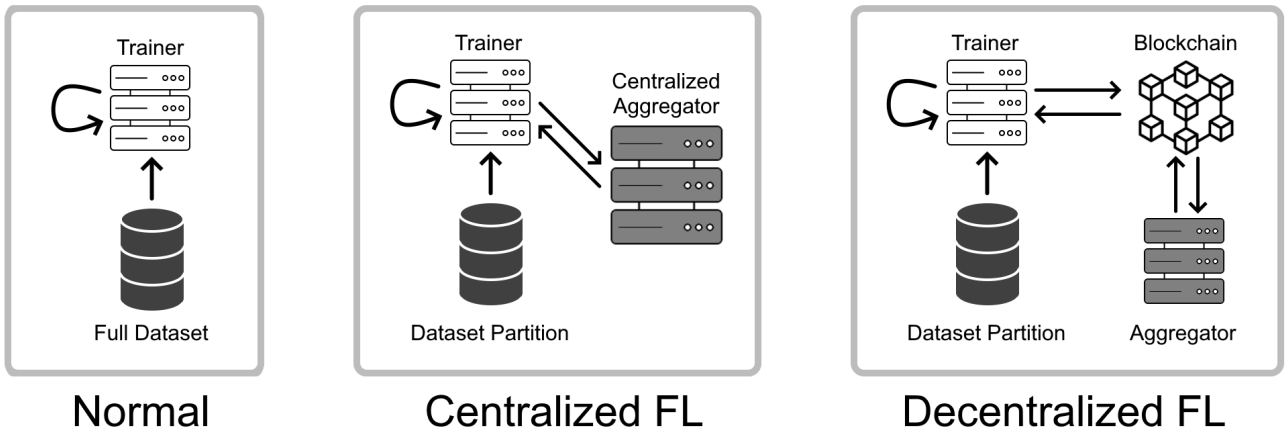

Figure 1: A visual representation of centralized training, centralized FL, and (blockchain-based) decentralized FL.

3.1.3 Decentralized Federated Learning Approach. Decentralized FL is based on centralized FL, but expands upon it to increase the degree of decentralization. The decentralized FL approach followed by this work is the loosely-coupled blockchain-based Blocklearning framework [H. Dias and Meratnia 2023; H. A. C. Dias 2022]. In this approach, there are n trainers and m aggregators in the FL process. Like centralized FL, each trainer contains their own local, private partition of the global data, and therefore there are n data partitions.

Also like centralized FL, the decentralized FL workflow proceeds in server rounds. However, unlike centralized FL, there is no centralized aggregator to perform duties such as global state management, including trainer selection. Instead, a model owner performs this duty. Since this approach relies on blockchain technology for communication and coordination between peers, the model owner is responsible for deploying the smart contracts that declare the network environment and facilitate the communication. This configuration allows for different nodes to act as owners for different models within the same network by simply sharing the address of their deployed smart contract.

Before server rounds can begin, the address of the smart contract must be circulated to any trainers or aggregators in the network. The trainers and aggregators must then call a function in the smart contract to register themselves as either a trainer or an aggregator; that is, the nodes must inform the network of their existence. Once enough nodes are registered, the training may commence.

Server rounds in decentralized FL begin with the model owner selecting the trainers that will participate in that round. These trainers are selected from the pool of trainers who have self-registered. The local training process then proceeds as in centralized FL. Upon completion of the local training steps, trainers store their updated weights in a content-addressable decentralized file system (such as IPFS [Benet 2014]). The storage of these weights in the content-addressable file system returns a content ID, an identifier unique to the content of the weights, which can be used to retrieve the file from anywhere within the network. The content ID of each trainer's update is then stored on the blockchain via the owner's smart contract.

After all trainers have submitted their updates, the model owner then starts the aggregation phase. This begins with the model owner selecting the aggregators to be used in the current round's aggregation, akin to the trainer selection process. Selected aggregators then retrieve the content IDs of the new updates and fetch them from the decentralized file system. Aggregators then perform their own individual aggregations, resulting in a new global model like the centralized aggregator in the centralized FL scenario. Once an aggregator has completed its local aggregation process, it stores its aggregated weights on the decentralized file system and submits the resulting content ID to the blockchain. In contrast with the training process, aggregation is deterministic for the same input weights, and therefore aggregators should each produce the same resulting aggregation, addressable by the same content ID. This completes one server round, and as in the centralized FL case, the process then continues until some predetermined criteria is met, such as the completion of a predefined number of server rounds.

3.2 Aggregation Algorithms

Since the inception of FL, a significant amount of research has been performed to develop improved and purpose-driven aggregation algorithms [Moshawrab et al. 2023; Qi et al. 2024]. In this work, we identify, analyze, and evaluate six aggregation algorithms from three distinct classes: statistical algorithms, score-based algorithms, and momentum-based algorithms. Two algorithms from each class are chosen. For statistical algorithms, we consider FedAvg [McMahan et al. 2017] and FedMedian [Yin et al. 2018]; for momentum-based algorithms, FedAvgM [Hsu et al. 2019] and FedYogi [Reddi et al. 2021]; and for score-based algorithms, Krum and MultiKrum [Blanchard et al. 2017].

Updates can be communicated to the aggregator in various different forms, the most common being the value of the new weights themselves or the change from the last global model. The following summary assumes the updates are passed as the value of the new weights, however conversion between the two forms is trivial.

The following notation is used in descriptions of the aggregation algorithms: t denotes the index of the current server round, S_t

denotes the set of trainers for server round t, n_k denotes the number of local steps performed by trainer k, and w_t denotes the model update for server round t and can be indexed by trainer for the local trainer update.

3.2.1 FedAvg [McMahan et al. 2017]. FedAvg is the simplest approach to aggregation, first proposed alongside the idea of FL itself. FedAvg simply returns the mean of the local trainer updates, weighted by the number of local training steps performed by the associated trainer. If all trainers perform equal training steps, then this calculation is simply the mean of the local trainer updates.

Algorithm 1 FedAvg [McMahan et al. 2017]

procedure FedAvg(t, n, S, w)
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
  return w_{t+1}
end procedure

3.2.2 FedMedian [Yin et al. 2018]. FedMedian is a similarly simple approach to aggregation, utilizing medians rather than means for an aggregation similar to FedAvg, however the use of means provides some key differences in the aggregation. Medians are less impacted by the presence of outliers than means, and as a result, FedMedian can lead to a better aggregation in the presence of outlier local datasets or under the influence of malicious trainers. FedMedian uses the coordinate-wise median rather than the median update itself. More formally, the coordinate-wise median is defined as $g := \texttt{med}\{x^i : i \in [m]\}$ for m vectors $x^i \in \mathbb{R}^d$, $i \in [m]$, such that g is a vector where the k-th coordinate is given by $g_k := \texttt{med}\{x_k^i : i \in [m]\}$ for each $k \in [d]$, with med being the one-dimensional median [Yin et al. 2018].

Algorithm 2 FedMedian [Yin et al. 2018]

procedure FedMedian(t, n, S, w)
  $w_{t+1} \leftarrow \texttt{elementwise\_median}(\{w_{t+1}^k : k \in S_t\})$
  return w_{t+1}
end procedure

3.2.3 FedAvgM [Hsu et al. 2019]. The FedAvgM aggregation algorithm is designed to improve the speed of convergence and to improve the performance of FL under non-i.i.d. data partitions. To do this, FedAvgM uses a server-side momentum term v, which is accumulated in successive server rounds. This means that either the centralized aggregator, or the blockchain which is coordinating the network of aggregators, must store this new state between server rounds. A new hyperparameter β is used to decay the momentum between consecutive server rounds. FedAvgM begins with the same calculation as FedAvg, where the mean of all local updates is taken. Then, the difference between this mean and the previous global weight is calculated, Δw_{t+1}. This can then be used to update the momentum v by computing $v_{t+1} \leftarrow \beta v_t + \Delta w_{t+1}$, before finally updating the global model weights with $w_{t+1} \leftarrow w_t + v_{t+1}$.

Algorithm 3 FedAvgM [Hsu et al. 2019]

procedure FedAvgM(t, n, S, w)
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $mean \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
  $\Delta w_{t+1} \leftarrow mean - w_t$
  $v_{t+1} \leftarrow \beta v_t + \Delta w_{t+1}$
  $w_{t+1} \leftarrow w_t + v_{t+1}$
  return w_{t+1}
end procedure

3.2.4 FedYogi [Reddi et al. 2021]. FedYogi is the final aggregation algorithm to be investigated, and further builds on FedAvgM. Rather than simply using momentum to globally optimize weights, FedYogi instead employs the Yogi optimization algorithm [Zaheer et al. 2018], which uses both momentum and an adaptive learning rate. It it important to note that FedYogi also requires additional global state to be stored, similar to FedAvgM.

Algorithm 4 FedYogi [Reddi et al. 2021]

procedure FedYogi(t, n, S, w)
  $a_t \leftarrow \sum_{k \in S_t} n_k$
  $mean \leftarrow \sum_{k \in S_t} \frac{n_k}{a_t} w_{t+1}^k$
  $\Delta w_{t+1} \leftarrow mean - w_t$
  $m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\Delta w_{t+1}$
  $v_{t+1} \leftarrow v_t - (1 - \beta_2)\Delta_{t+1}^2 \texttt{sign}(v_t - \Delta_{t+1}^2)$
  $w_{t+1} \leftarrow w_t + \eta \frac{m_{t+1}}{\sqrt{v_{t+1}} + \tau}$
  return w_{t+1}
end procedure

3.2.5 Krum [Blanchard et al. 2017]. Krum, like FedMedian, is designed to perform a more robust aggregation in the presence of outlier local datasets or under the influence of malicious trainers. In particular, Krum is configurable via the hyperparameter f, which represents the number of malicious trainers that the algorithm can tolerate. In contrast to FedMedian, Krum uses a more sophisticated method of determining which parameter updates to incorporate compared to the straightforward median method seen in FedMedian.

The main idea of Krum is to select the local model update that is the most consistent with the majority of other local updates, and to use that as the next global update. The Krum algorithm proceeds update-by-update, and calculates the Euclidean distance between w_i and w_j, $j \in S_t, i \neq j$. Then, the closest $n - f - 2$ updates are summed. This score calculation intends to disregard any potentially malicious trainers. The trainer update with the smallest score is selected to become the next global update.

A notable consequence of the Krum aggregation algorithm is that it only incorporates one trainer's update into the global update per server round, which can lead to a much slower convergence in non-i.i.d. scenarios.

3.2.6 MultiKrum [Blanchard et al. 2017]. MultiKrum is a variation of Krum intended to improve Krum's slow convergence by incorporating multiple local updates per server round while still

Algorithm 5 Krum [Blanchard et al. 2017]

procedure KRUM(t, n, S, w)
  $smallest \leftarrow \infty$
  $smallest\_index \leftarrow 0$
  for $k \in S_t$ **do**
    $distances \leftarrow []$
    for $j \in S_t$, $j \neq k$ **do**
      $distances$.append($||w_{t+1}^k - w_{t+1}^j||$)
    end for
    $distances = distances$.sort()
    $score = 0$
    for $i = 0$; $i < n - f - 2$; $i++$ **do**
      $score \leftarrow score + distances[i]$
    end for
    if $score > smallest$ **then**
      $smallest \leftarrow score$
      $smalles\_index \leftarrow k$
    end if
  end for
  $w_{t+1} \leftarrow w_{t+1}^{smallest\_index}$
  return w_{t+1}
end procedure

Algorithm 6 MultiKrum [Blanchard et al. 2017]

procedure MULTIKRUM(t, n, S, w)
  $scores \leftarrow []$
  $update\_indices \leftarrow []$
  // Calculate scores
  for $k \in S_t$ **do**
    $distances \leftarrow []$
    for $j \in S_t$, $j \neq k$ **do**
      $distances$.append($||w_{t+1}^k - w_{t+1}^j||$)
    end for
    $distances = distances$.sort()
    $score = 0$
    for $i = 0$; $i < n - f - 2$; $i++$ **do**
      $score \leftarrow score + distances[i]$
    end for
    $scores$.sorted_insert($score$)
    $rank \leftarrow scores$.indexof($score$)
    $update\_indices$.insert_at($k, rank$)
  end for
  // Mean of lowest m scores
  $w_{t+1} \leftarrow (\sum_{i=0}^{m-1} w_{t+1}^{update\_indicies[i]})/m$
  return w_{t+1}
end procedure

maintaining fault tolerance. MultiKrum uses an additional hyperparameter m which denotes the number of local updates that should be used to form the next global update.

MultiKrum begins in the same manner as Krum, calculating Euclidean distances between each pair of updates, and calculating a score for each individual local update. Rather than selecting the single update with the lowest score, the m updates with the lowest scores are selected. The mean of these m scores are then taken to produce the next global update.

MultiKrum acts as a midpoint between FedAvg and Krum, incorporating elements of both algorithms. This leads to a faster aggregation than Krum while retaining fault tolerance, unlike FedAvg. However, MultiKrum still converges slower than FedAvg, especially for small values of m.

3.3 Evaluation Strategy

The goals of the trained diffusion model are two-fold: to generate image samples from a distribution that closely matches the real global distribution; and to be able to generate a synthetic image dataset that can be used to train a downstream model capable of good performance on real images. As a result of the dual goals, two different evaluation techniques must be employed. The first is direct evaluation on the generative model, and the second is evaluation on a downstream model trained from a synthetic dataset created by the generative model.

The results of each of the diffusion models trained with FL are compared to the results of a baseline diffusion model trained via normal centralized learning.

3.3.1 Generative Model. Two popular generative metrics are used to gauge the performance of the diffusion model. The first metric is the Inception Score (IS) [Salimans et al. 2016], which uses a pretrained Inception-v3 classifier model [Szegedy et al. 2015] to quantify both image quality and diversity. IS only uses generated samples, and does not use any real images in the score calculation. A greater IS is considered favourable. The second generative evaluation metric is the Fréchet Inception Distance (FID) [Heusel et al. 2017], which also uses a pretrained Inception-v3 classifier, but instead involves real images and compares the distribution of real and synthetic images in the Inception-v3 feature space. A lower FID is considered superior.

3.3.2 Downstream Model. In addition to the generative model metrics, a downstream ResNet50 classifier [He et al. 2016] pretrained on the ImageNet-1k dataset [Deng et al. 2009] is fine-tuned on the synthetic dataset generated by each diffusion model. The synthetic datasets contain balanced classes with 200 samples per class. The classifier is then tested on a set of real data that was uniformly sampled from the real global dataset and withheld from diffusion model training. Four classifier metrics are collected from this test: accuracy, precision, recall, and F1 score.

4 Experiment Configuration

4.1 Dataset

The dataset to be used in this analysis is the CottonWeedID15 dataset [Lu 2021]. This dataset contains 5,187 high resolution RGB images with varying dimensions of common weeds from cotton fields in the southern United States. The weed images are divided into 15 classes depending on the type of weed. The number of samples in each class range from 61 to 1,115. This division into classes will allow the data to be partitioned in a non-i.i.d. manner for the FL case; that is, each device in the FL network will contain a subset of classes with little-to-no overlap with the remainder of the network.

From this dataset, 20% are uniformly randomly sampled and removed to form the test set for use on a downstream classification model. The remaining 80% is divided into 8 disjoint partitions such that no two partitions contain any common class, with the exception of the most frequent class, Morningglory, which is contained in 2 partitions.

Plant Class	Quantity	Percent
Carpetweed	763	14.71%
Crabgrass	111	2.14%
Eclipta	254	4.90%
Goosegrass	216	4.16%
Morningglory	1115	21.50%
Nutsedge	273	5.26%
Palmer Amaranth	689	13.28%
Prickly Sida	129	2.49%
Purslane	450	8.68%
Ragweed	129	2.49%
Sicklepod	240	4.63%
Spotted Spurge	234	4.51%
Spurred Anoda	61	1.18%
Swinecress	72	1.39%
Waterhemp	451	8.69%

Table 1: Weed label quantities and percentages

4.2 Software and Hardware

4.2.1 Tools. The model trained in this experiment is the stable-diffusion-v1-5 variant of Stable Diffusion [Rombach et al. 2022] using the weights and implementation given by the HuggingFace `Diffusers` library [HuggingFace 2025]. The centralized FL implementation uses the `Flower` FL framework [Flower Labs 2025]. For consistency, the aggregation algorithm implementations from `Flower` were also used in the decentralized FL implementation. Only the U-Net of the model is trained; the weights of the text encoder and variational autoencoder are frozen. The U-Net contains approximately 859 million parameters.

The downstream ResNet50 classifier was implemented using `PyTorch` [Paszke et al. 2019], and the generative metrics evaluations used the implementation given by the `torch-fidelity` library [Obukhov et al. 2020].

4.2.2 Configuration. The centralized model was trained using a standard implementation derived from the example diffusion model training script given by the HuggingFace `Diffusers` library [HuggingFace 2025]. This model uses oversampling to combat class imbalance.

The centralized FL models were trained using eight trainers, with four trainers selected for training per server round. All six aggregation algorithms were used to train six different centralized FL models.

The decentralized FL models were trained using eight trainers and eight aggregators, with four trainers and all eight aggregators being selected for training per server round. The nodes were coordinated using a private proof-of-work Ethereum network [Wood et al. 2014], and file sharing was enabled through IPFS [Benet 2014]. FedAvg and FedMedian were used to train two decentralized FL models.

In all scenarios, the diffusion models were trained for 20000 training steps with a local AdamW optimizer. In the FL cases, training occurred for 100 local steps per server round for a total of 50 server rounds. Each trainer in this experiment has full exclusive access to 1 NVIDIA V100-32GB GPU. As a result, the training processes for the FL scenarios required 8 NVIDIA V100-32GB GPUs.

5 Results and Discussion

5.1 Execution Time

Due to the significant differences in network topology between the baseline approach and the two FL approaches, we expect to see significant differences in execution time. There is no network overhead in the centralized baseline which is expected to increase execution speed, however, the FL approaches both allow for increased parallelism, which also increases execution speed. As a result, it is not immediately obvious which training paradigm can be expected to execute faster.

Table 3 displays the training time for 400 training steps. In both FL paradigms, 400 training steps is equivalent to one full server round (due to the configuration parameters). The centralized baseline executed the fastest, completing 400 local steps in an average of 74.1 seconds, demonstrating how the lack of network overhead significantly increases the execution speed. The centralized FL paradigm average server round execution time was 110.9 seconds, a 1.50x increase, but not remarkably dissimilar from the centralized baseline.

Compared to the centralized baseline and centralized FL implementations, the decentralized FL training process is extremely slow. Decentralized FL took an average of 621.8 seconds per server round, a 5.61x increase from the centralized FL experiment, and an 8.39x increase from the centralized baseline. This is an expected outcome, especially compared to the centralized FL implementation, due to the significant network overhead associated with the proof-of-work private Ethereum blockchain network and the frequent transmission of the 3.4GB model parameters. In the aggregation phase, each of the eight aggregators must download each of the four local updates, leading to the total transfer of 32 weight files, for a total transmission of approximately 108.8GB. More generally, for n trainers and m aggregators, the weight transfer per round in both explored FL paradigms is $O(m \cdot n)$, however since there is only a single aggregator in centralized FL, the expression simplifies to $O(n)$, making the difference in execution time more evident.

5.2 Generative Capability

A class-balanced 3000 image synthetic dataset was created for each aggregation algorithm. Figure 2 displays a sample of synthetic images generated by each algorithm, along with comparisons to real sample images. The most notable pattern within figure 2 is how the similarity between real and synthetic images decreases for less frequent classes, particularly for models trained using FL. Furthermore, the samples generated by models trained with FL generally look less realistic overall compared to the centralized baseline.

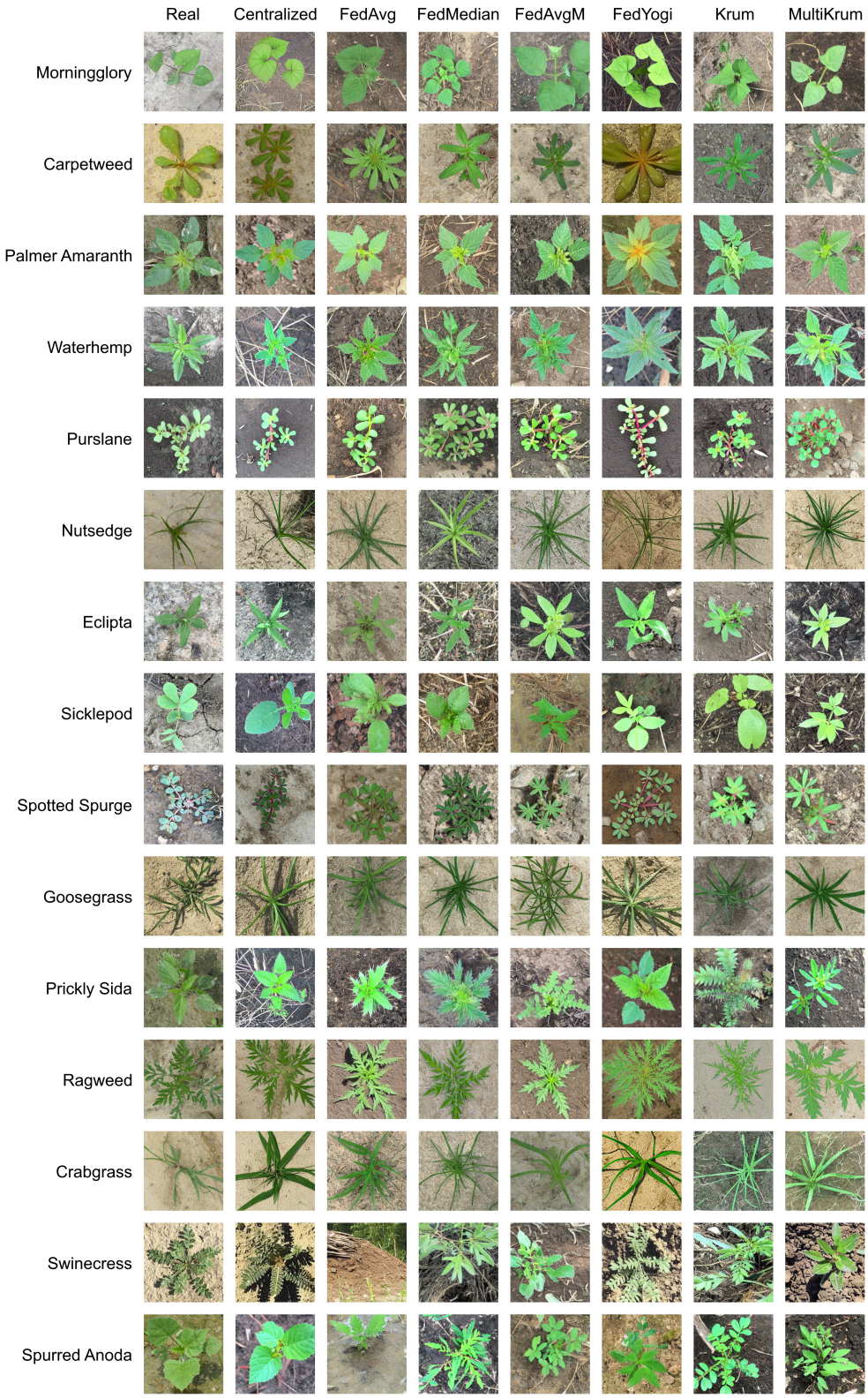

Figure 2: Array of images generated by model trained using the various aggregation algorithms. Real images from each class are displayed in the leftmost column. Classes are ordered from most frequent (top) to least frequent (bottom).

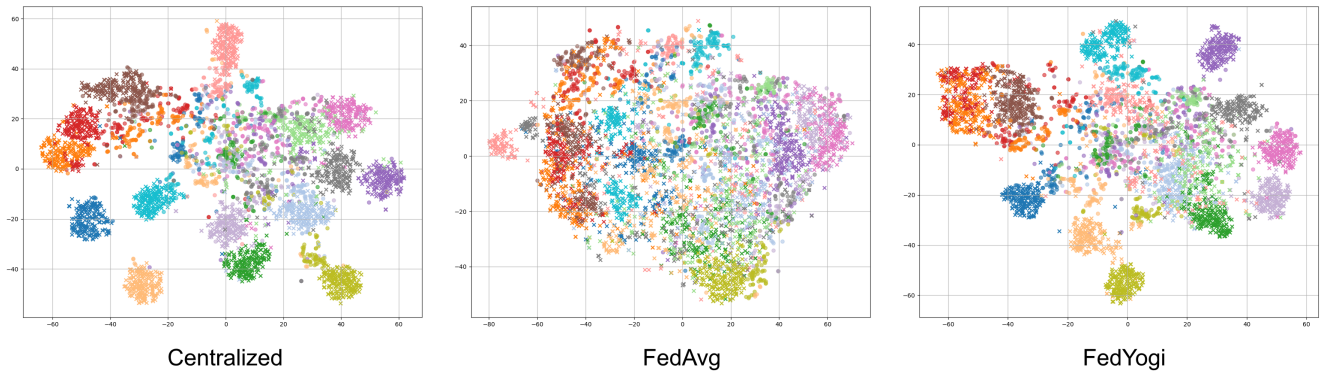

Figure 3: Projection of latent vectors from an Inception-v3 classifier using t-SNE. Each point represents a unique image. Points marked with a "x" represent synthetic images, whereas points marked with a circle represent real images. Classes are denoted by colour.

Paradigm	Execution Time Per Round
Centralized	1.23 minutes (74.1s)
Centralized FL	1.84 minutes (110.9s)
Decentralized FL	10.36 minutes (621.8s)

Table 3: Execution times per round for different architectures.

Figure 2 demonstrates a qualitative superiority in generative capability for the model trained using the FedYogi algorithm. For many of the low- and mid-frequency classes, the model trained using FedYogi was able to produce much more realistic samples compared to all other methods. Despite this, FedYogi demonstrated signs of feature amplification for high-frequency classes, which is especially noticeable for the Morningglory and Carpetweed classes. This is an interesting result, since many other aggregation algorithms failed to accurately capture the features of Carpetweed despite its high frequency in the dataset, likely due to Carpetweed's thick rounded leaves being unlike any other class in the dataset. This is particularly the case for the Krum and MultiKrum algorithms, which specifically discriminate against such departures.

Another notable result in figure 2 is the the struggle of many of the algorithms to accurately learn features of very low-frequency classes, particularly Swinecress and Spurred Anoda. In the case of Spurred Anoda, there is a significant difference between both the angle from which the image was taken as well as the structure of the plant when comparing the real and synthetic images. This pattern is notable within the Swinecress class as well, however, a more notable pattern emerges: each model trained using FL (with the exception of FedYogi) generated images for the Swinecress class that did not contain plants. Many of the images contained hills and industrial imagery from rural areas. These non-plant images were intentionally excluded from figure 2 to demonstrate any semblance of plant features that were captured in the training process, however, this result is very significant. The increased learning speed enabled by the momentum mechanism in FedYogi likely helped it's model avoid such issues by allowing it to extract relevant image features in fewer training rounds.

Table 2 displays the capabilities of each model through the common IS [Salimans et al. 2016] and FID [Heusel et al. 2017] metrics. FedYogi achieved the greatest IS, generally meaning the model trained using FedYogi demonstrates the best image quality without sacrificing diversity, although the IS is not significantly higher than other models, degrading the strength of this metric in showing differences between the models. In fact, the difference in IS between the model trained with FedYogi and most other models is within the standard deviation of FedYogi's IS.

The results for FID are equally unreflective of the differences between trained models. The model trained using FedAvgM yields

Algorithm	Inception Score (↑)	FID (↓)	Classifier Accuracy (↑)	Precision (↑)	Recall (↑)	F1 Score (↑)
Centralized	2.92 ± 0.10	81.2	53.47%	80.92%	53.47%	59.57%
FedAvg	2.68 ± 0.10	60.0	20.81%	67.14%	20.81%	26.87%
FedMedian	2.59 ± 0.08	64.4	22.16%	65.93%	22.16%	28.28%
FedAvgM	2.71 ± 0.12	**58.4**	17.34%	66.28%	17.34%	23.45%
FedYogi	**2.75 ± 0.11**	69.8	**31.60%**	**74.47%**	**31.60%**	**35.93%**
Krum	2.65 ± 0.10	72.1	17.82%	55.41%	17.82%	19.00%
MultiKrum	2.46 ± 0.08	67.5	15.13%	55.23%	15.13%	18.73%

Table 2: Comparison of aggregation algorithms across generative and downstream classifier-based metrics. Best values are emboldened.

a superior FID score to all other models, indicating that the model generates images that are more similar to the real image distribution than any other model. However, this is not reflected via the samples in figure 2. Furthermore, the FID values of all models trained using FL outperform the centralized baseline, which is also not reflected in figure 2.

The goal for the training of these diffusion models is to generate synthetic datasets that can be used for training downstream machine learning models. As a result, the performance metrics of the ResNet50 classifier [He et al. 2016] trained on the synthetic datasets are the main focus of this evaluation.

In line with the IS results, FedYogi demonstrates the best performance among FL algorithms with an accuracy of 31.60% and F1 score of 35.93%. These results are significantly lower than that of the centralized baseline, which achieved an accuracy of 53.47% and an F1 score of 59.57%, however, they are significantly higher than the next nearest FL algorithms, FedMedian and FedAvg, with accuracies between 20-23% and F1 scores between 26-29%. Krum and MultiKrum attained poor generative model metrics and poor downstream classifier metrics. This result is expected, since these algorithms disregard some subset of local updates in each server round, thereby disregarding subsets of the data distribution due to the dataset partitioning, slowing convergence. This is especially noticeable in this experiment due to the fixed number of local training steps.

In addition to quantitative evaluations of the classifier-produced latent space representations of the synthetic datasets, qualitative evaluations can also be performed. As in the IS and FID metrics, latent vectors for both real and synthetic images can be extracted from an Inception-v3 classifier and visualized. Since the latent vectors are of high dimensionality, they must first be reduced to dimensions that are understandable by humans. One such method is through t-SNE, which can be used to reduce high-dimensional vectors to a 2-dimensional vector space, allowing for them to be graphed. This is displayed in figure 3.

Figure 3 illustrates connections between many data points, and therefore displays a variety of connections within both the real and synthetic datasets. Small distances between the real and synthetic projections of samples within the same class suggest significant overlap between their respective distributions. In contrast, tightly clustered samples within a class indicate that the class is more easily separable, facilitating more accurate classification. It is then evident that the centralized baseline best fulfills both patterns, followed closely by the model trained with FedYogi. The models trained using FedAvg and Krum lag significantly behind both in terms of both patterns.

Overall, the FedYogi algorithm performed the best when compared to alternative aggregation algorithms. This is because of FedYogi's inclusion of momentum and an adaptive learning rate within the global state, which allows for a faster convergence compared to simpler algorithms. In contrast, the Krum family of models displayed the poorest performance due to their disregarding of some subset of local updates, leading to the ignoring of a large amount of the dataset in each server round and a slower convergence. Other algorithms such as FedAvg, FedMedian, and FedAvgM are much more simplistic than FedYogi, and as a result, fail to converge as quickly.

6 Conclusion

Overall, the experiments performed in this work demonstrate that there is significant variation within FL implementations, especially for the training of large diffusion models. These results become exacerbated when used to train downstream machine learning models, such as image classification models. The use of decentralized FL techniques introduces substantial network overhead, leading to significantly increased execution times, particularly for large models. However, this approach enables greater decentralization in the training process by eliminating the need for a central aggregator. Furthermore, the use of different model aggregation algorithms leads to substantial differences in convergence time and global model quality. Within the context of fine-tuning a large diffusion model with non-i.i.d. data in the presence of no malicious nodes, FedYogi was found to demonstrate the fastest convergence and best resulting synthetic dataset for downstream model training. The most significant variance between paradigms or algorithms was the increase in training time for decentralized FL compared to centralized FL, and therefore the reduction of transmitted parameters and the associated effects on large diffusion models is a likely area for future research. In all, using federated learning for the training and fine-tuning of diffusion models reduces both training efficiency and model performance, however, it offers enhanced data privacy, particularly when generating synthetic datasets for training downstream machine learning models

References

Kar Balan, Andrew Gilbert, and John Collomosse. Nov. 18, 2024. "PDFed: Privacy-Preserving and Decentralized Asynchronous Federated Learning for Diffusion Models." In: *Proceedings of 21st ACM SIGGRAPH Conference on Visual Media Production*. (Nov. 18, 2024), 1–9. arXiv: 2409.18245[eess]. DOI: 10.1145/3697294.3697306.

Juan Benet. July 14, 2014. *IPFS - Content Addressed, Versioned, P2P File System*. (July 14, 2014). arXiv: 1407.3561[cs]. DOI: 10.48550/arXiv.1407.3561.

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent." In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. Retrieved Apr. 14, 2025 from https://proceedings.neurips.cc/paper_files/paper/2017/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. June 2009. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009 IEEE Conference on Computer Vision and Pattern Recognition. ISSN: 1063-6919. (June 2009), 248–255. DOI: 10.1109/CVPR.2009.5206848.

Prafulla Dhariwal and Alexander Nichol. 2021. "Diffusion Models Beat GANs on Image Synthesis." In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 8780–8794. Retrieved Nov. 29, 2024 from https://proceedings.neurips.cc/paper/2021/hash/49ad23d1ec9fa4bd8d77d02681df5cfa-Abstract.html.

Henrique Dias and Nirvana Meratnia. 2023. "BlockLearning: A Modular Framework for Blockchain-Based Vertical Federated Learning." In: *Ubiquitous Security*. Ed. by Guojun Wang, Kim-Kwang Raymond Choo, Jie Wu, and Ernesto Damiani. Springer Nature, Singapore, 319–333. ISBN: 978-981-99-0272-9. DOI: 10.1007/978-981-99-0272-9_22.

Henrique Afonso Coelho Dias. Sept. 7, 2022. "Impact Analysis of Different Consensus, Participant Selection and Scoring Algorithms in Blockchain-based Federated Learning Systems Using a Modular Framework." M.Sc. Thesis. Eindhoven University of Technology, (Sept. 7, 2022).

Flower Labs. 2025. *Flower: A Friendly Federated AI Framework*. Retrieved Apr. 16, 2025 from https://flower.ai/.

Matthijs de Goede, Bart Cox, and Jérémie Decouchant. June 18, 2024. *Training Diffusion Models with Federated Learning*. (June 18, 2024). arXiv: 2406.12575[cs]. DOI: 10.48550/arXiv.2406.12575.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. June 2016. "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. (June 2016), 770–778. DOI: 10.1109/CVPR.2016.90.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. Retrieved Jan. 31, 2025 from https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Dec. 16, 2020. *Denoising Diffusion Probabilistic Models*. (Dec. 16, 2020). arXiv: 2006.11239. DOI: 10.48550/arXiv.2006.11239.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Sept. 13, 2019. *Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification*. (Sept. 13, 2019). arXiv: 1909.06335[cs]. DOI: 10.48550/arXiv.1909.06335.

Gaofeng Hua, Li Zhu, Jinsong Wu, Chunzi Shen, Linyan Zhou, and Qingqing Lin. 2020. "Blockchain-Based Federated Learning for Intelligent Control in Heavy Haul Railway." *IEEE Access*, 8, 176830–176839. DOI: 10.1109/ACCESS.2020.3021253.

HuggingFace. 2025. *Diffusers*. Retrieved Apr. 16, 2025 from https://huggingface.co/docs/diffusers/en/index.

Renuga Kanagavelu et al.. May 2020. "Two-Phase Multi-Party Computation Enabled Privacy-Preserving Federated Learning." In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). (May 2020), 410–419. DOI: 10.1109/CCGrid49817.2020.00-52.

Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. June 2020. "Blockchained On-Device Federated Learning." *IEEE Communications Letters*, 24, 6, (June 2020), 1279–1283. Conference Name: IEEE Communications Letters. DOI: 10.1109/LCOMM.2019.2921755.

Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Oct. 8, 2016. *Federated Optimization: Distributed Machine Learning for On-Device Intelligence*. (Oct. 8, 2016). arXiv: 1610.02527[cs]. DOI: 10.48550/arXiv.1610.02527.

Abhinav Kumar, Vishal Purohit, Vandana Bharti, Rishav Singh, and Sanjay Kumar Singh. Aug. 2022. "MediSecFed: Private and Secure Medical Image Classification in the Presence of Malicious Clients." *IEEE Transactions on Industrial Informatics*, 18, 8, (Aug. 2022), 5648–5657. DOI: 10.1109/TII.2021.3138919.

Ilya Loshchilov and Frank Hutter. 2017. *Decoupled Weight Decay Regularization*. (2017). arXiv: 1711.05101[cs]. DOI: 10.48550/arXiv.1711.05101.

Yuzhen Lu. 2021. *CottonWeedID15*. (2021). DOI: 10.34740/KAGGLE/DSV/2685927.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Apr. 10, 2017. "Communication-Efficient Learning of Deep Networks from Decentralized Data." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. ISSN: 2640-3498. PMLR, (Apr. 10, 2017), 1273–1282. Retrieved Jan. 5, 2025 from https://proceedings.mlr.press/v54/mcmahan17a.html.

Mohammad Moshawrab, Mehdi Adda, Abdenour Bouzouane, Hussein Ibrahim, and Ali Raad. Jan. 2023. "Reviewing Federated Learning Aggregation Algorithms; Strategies, Contributions, Limitations and Future Perspectives." *Electronics*, 12, 10, (Jan. 2023), 2287. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/electronics12102287.

Quang Nguyen, Truong Vu, Anh Tran, and Khoi Nguyen. Dec. 15, 2023. "Dataset Diffusion: Diffusion-based Synthetic Data Generation for Pixel-Level Semantic Segmentation." *Advances in Neural Information Processing Systems*, 36, (Dec. 15, 2023), 76872–76892. Retrieved Apr. 15, 2025 from https://proceedings.neurips.cc/paper_files/paper/2023/hash/f2957e48240c1d90e62b303574871b47-Abstract-Conference.html.

Alexander Quinn Nichol and Prafulla Dhariwal. July 1, 2021. "Improved Denoising Diffusion Probabilistic Models." In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, (July 1, 2021), 8162–8171. Retrieved Nov. 16, 2024 from https://proceedings.mlr.press/v139/nichol21a.html.

Anton Obukhov, Maximilian Seitzer, Po-Wei Wu, Semen Zhydenko, Jonathan Kyl, and Elvis Yu-Jing Lin. 2020. *High-fidelity performance metrics for generative models in PyTorch*. Version v0.3.0. Version: 0.3.0, DOI: 10.5281/zenodo.4957738. (2020). DOI: 10.5281/zenodo.4957738.

Adam Paszke et al.. Dec. 3, 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. (Dec. 3, 2019). arXiv: 1912.01703[cs]. DOI: 10.48550/arXiv.1912.01703.

Pian Qi, Diletta Chiaro, Antonella Guzzo, Michele Ianni, Giancarlo Fortino, and Francesco Piccialli. Jan. 1, 2024. "Model aggregation techniques in federated learning: A comprehensive survey." *Future Generation Computer Systems*, 150, (Jan. 1, 2024), 272–293. DOI: 10.1016/j.future.2023.09.008.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Sept. 8, 2021. *Adaptive Federated Optimization*. (Sept. 8, 2021). arXiv: 2003.00295[cs]. DOI: 10.48550/arXiv.2003.00295.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. June 2022. "High-Resolution Image Synthesis with Latent Diffusion Models." In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, New Orleans, LA, USA, (June 2022), 10674–10685. ISBN: 978-1-6654-6946-3. DOI: 10.1109/CVPR52688.2022.01042.

Chitwan Saharia et al.. Dec. 6, 2022. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding." *Advances in Neural Information Processing Systems*, 35, (Dec. 6, 2022), 36479–36494. Retrieved Nov. 29, 2024 from https://proceedings.neurips.cc/paper_files/paper/2022/hash/ec795aeadae0b7d230fa35cbaf04c041-Abstract-Conference.html.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. 2016. "Improved Techniques for Training GANs." In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc. Retrieved Jan. 31, 2025 from https://proceedings.neurips.cc/paper_files/paper/2016/hash/8a3363abe792db2d8761d6403605aeb7-Abstract.html.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. June 1, 2015. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics." In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, (June 1, 2015), 2256–2265. Retrieved Nov. 15, 2024 from https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Yang Song and Stefano Ermon. Dec. 8, 2019. "Generative modeling by estimating gradients of the data distribution." In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, (Dec. 8, 2019), 11918–11930. Retrieved Nov. 15, 2024 from.

Fiona Victoria Stanley Jothiraj and Afra Mashhadi. May 13, 2024. "Phoenix: A Federated Generative Diffusion Model." In: *Companion Proceedings of the ACM Web Conference 2024* (WWW '24). Association for Computing Machinery, New York, NY, USA, (May 13, 2024), 1568–1577. ISBN: 979-8-4007-0172-6. DOI: 10.1145/3589335.3651935.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Dec. 11, 2015. *Rethinking the Inception Architecture for Computer Vision*. (Dec. 11, 2015). arXiv: 1512.00567[cs]. DOI: 10.48550/arXiv.1512.00567.

Brandon Trabucco, Kyle Doherty, Max Gurinas, and Ruslan Salakhutdinov. May 25, 2023. *Effective Data Augmentation With Diffusion Models*. (May 25, 2023). Retrieved Nov. 17, 2024 from http://arxiv.org/abs/2302.07944 arXiv: 2302.07944.

Zhilin Wang, Qin Hu, Minghui Xu, Yan Zhuang, Yawei Wang, and Xiuzhen Cheng. June 1, 2024. *A Systematic Survey of Blockchained Federated Learning*. (June 1, 2024). arXiv: 2110.02182[cs]. DOI: 10.48550/arXiv.2110.02182.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. 2020. "Federated Learning With Differential Privacy: Algorithms and Performance Analysis." *IEEE Transactions on Information Forensics and Security*, 15, 3454–3469. DOI: 10.1109/TIFS.2020.2988575.

Gavin Wood et al.. 2014. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper*, 151, 2014, 1–32.

Weijia Wu, Yuzhong Zhao, Mike Zheng Shou, Hong Zhou, and Chunhua Shen. 2023. "DiffuMask: Synthesizing Images with Pixel-level Annotations for Semantic Segmentation Using Diffusion Models." In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 1206–1217. Retrieved Apr. 15, 2025 from https://openaccess.thecvf.com/content/ICCV2023/html/Wu_DiffuMask_Synthesizing_Images_with_Pixel-level_Annotations_for_Semantic_Segmentation_Using_ICCV_2023_paper.html.

Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. July 3, 2018. "Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates." In: *Proceedings of the 35th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, (July 3, 2018), 5650–5659. Retrieved Apr. 14, 2025 from https://proceedings.mlr.press/v80/yin18a.html.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. 2018. "Adaptive Methods for Nonconvex Optimization." In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc. Retrieved Apr. 15, 2025 from https://proceedings.neurips.cc/paper_files/paper/2018/hash/90365351ccc7437a1309dc64e4db32a3-Abstract.html.

Xiaokang Zhou, Wei Liang, Jinhua She, Zheng Yan, and Kevin I-Kai Wang. June 2021. "Two-Layer Federated Learning With Heterogeneous Model Aggregation for 6G Supported Internet of Vehicles." *IEEE Transactions on Vehicular Technology*, 70, 6, (June 2021), 5308–5317. DOI: 10.1109/TVT.2021.3077893.