Analyzing the Impact of Client Dropouts on Vertical Federated Learning Models

Sindhuja Madabushi, Joshua Carter, Sepideh Fatemi,
Rodrigo Figueroa Justiniano,
Teona Zurabashvili

Virginia Polytechnique Institute and State University

1 Abstract

Vertical Federated Learning (VFL) enables collaborative Neural Network training across multiple clients by sharing only intermediate activations and gradients. However, client dropout remains a critical challenge, as not all clients have access to same amount of resources for training complex models. Hence, we might lose out on valuable data some clients possess. This project investigates the impact of client dropout patterns on the convergence of VFL models. We first explore various dropout scenarios, including random dropouts and dropouts near convergence, and assess their relation to feature importance. To mitigate the adverse effects of dropout, we analyze the effect of different optimizers on drop out patterns. We evaluate the influence of data density functions for clients on convergence behavior under dropout conditions. Our findings contribute to understanding and enhancing the resilience of VFL models to client dropouts, paving the way for more reliable and efficient collaborative learning frameworks. The code implementation and experimental setup are publicly available in Github link .

2 Introduction

Ideally, we would want the data of all clients to be independently and identically distributed. However, nothing guarantees such a distribution. Some clients may contribute less after the training process, some may contribute more. The question becomes, how do we utilize the data the dropped out clients possess to train the model?

Assume that neural network has $\mathcal{K}$ clients: $\mathcal{K} := \{1, 2, ..., k\}$. Further, assume that each client k, in the client set $\mathcal{K}$, is holding a local dataset defined as [4]: $D_k := \{x_k^1, x_k^2, ..., x_k^N\}$, where N represents number of samples, while the subindex $j = 1, 2, ..., k$ represents the client's index. In other words, for $n = 1, 2, ..., N$ and $j = 1, 2, ..., k$, x_j^n represents the block of features of n^{th} sample that belongs to the client j[4]. There are two types of federated learning known as Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). The former refers to the scenario when different clients share same features but the data corresponding to those features is decentralized or private. In contrast with the HFL, in VFL clients share same data, however the features are distinct. One of the examples in which VFL would be applicable is university database and a local bank database. A student may have their student account attached to their bank account. In other words, the student may be considered as a shared user between the bank and the university. Even though the systems may have distinct features[1], they could share same users with limited access to the data of that user. The challenge becomes, how do we train VFL model given the lack of access to data? How does dropout affects the convergence rate of the predictions?

One of the common methods proposed to train VFL models is by using split neural networks[4].

To use split neural networks for training VFL model, first we will need to create a *representation model* for each client. Representation model refers to a technique of representing a client data. We can denote *representation model* by f_k and parametrize it by θ_{f_k}. Clients will extract representations, in other words, the clients will use the output of the function f_k to input into a *fusion model* g. In general, fusion model is a technique of combining multiple representations to make a robust prediction. Let g be parametrized by θ_g. Two common practices are to have a fusion level at a client or server level[4]. The main goal is to learn parameters of the predictor function that minimizes the loss function. More formally one can define a predictor [2][4]

$$h(\mathbf{x}^n; \theta_K) := g(\{f_k(\mathbf{x}_k^n; \theta_{f_k})\}_{k=1}^K; \theta_g).$$

Our objective is to minimize

$$\min_{\theta_k} \frac{1}{N} \sum_{n=1}^{N} l(h(x^n; \theta_K), y^n)$$

where l denotes loss function and y^n denotes the label of sample n. It is important to note that y^n and fusion model are both held by a client, or both are held by a server. Once the clients' data gets aggregated (in our case we are just concatenating client data), we pass this data to the server using $WX + B$ transformation, where $W : \mathbf{R}^m \longrightarrow \mathbf{R}^n$ is an $m \times n$ matrix with entries representing the weights. X is the input (concatenated label) and B is the bias ($m \times n$ matrix). Once the server receives this aggregated data, it will perform again $WX + B$ and apply softmax function to it. The softmax

[1] In general what is meant by "feature" is a type of information.

[2] Note that $\mathbf{x}^n$ is a vector of features that corresponds to n^{th} sample.

function takes the output of $WX + B$ and it outputs the vector where the entries are normalized entries, that is the probabilities that are proportional to the exponentials of the input entries. After this, We will compute the loss function defined above. In our implementation we will be using cross entropy loss function. Under reasonable conditions of smoothness, we would compute the gradient of this loss function. Our optimization problem will be to determine parameters that minimizes the loss function under a given constraint. What makes it difficult to solve the minimization problem is that some neurons may drop out from the training process. The question becomes, how do we train the common model successfully when individual client dropouts constitute to lack of access to the data that without the drop out would have been accessible to participate in training?

3 Background

In this section we will mainly talk about challenges and methods to resolve these challenges in VFL.

Let $\mathcal{X}$ be an input space partitioned into $\mathcal{K}$ feature spaces:

$$\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{X}_3 \times ... \times \mathcal{X}_K,$$

where $K \in \mathcal{K}$. Here $\mathcal{K}$ represents the set of clients, the notation "$\times$" represents cross product. Notice that $\mathcal{D}_k := (x_k^1, x_k^2, ..., x_k^N)$ (represents the local dataset of client k) will satisfy $\mathcal{D}_k \subseteq \mathcal{X}_k$. To train a general unconstrained predictor $\tilde{h} : \mathcal{X} \longrightarrow \mathcal{Y}$, one would need to solve [4]:

$$\min_{\tilde{\theta}} \frac{1}{N} \sum_{n=1}^{N} l(\tilde{h}(x^n; \tilde{\theta}, y^n)).$$

Assuming that some clients dropped out and data that these clients contributed for training is no longer used, leads to the new constraint in VFL. The goal remains to train the common model even though some parts of information are no longer available. If we try to approximate $\tilde{h}$, the unconstrained predictor, by h defined above, the issue will be that we will not be able to handle missing values during training or inference.

To avoid this, we can define and learn local predictors. Learning local predictors means that each client will learn local parameters θ_k to approximate $\tilde{h}$ by a local predictor $h : \mathcal{X}_k \longrightarrow \mathcal{Y}$. An abstract representation of h_k is given by:

$$h_k(x_k^n; \theta_k) := g_k(f_k(x_k^n; \theta_{f_k}); \theta_{g_k})$$

where $\theta_k := (\theta_{f_k}, \theta_{g_k})$[4]

Representation model is defined to be

$$f_k : \mathcal{X}_k \longrightarrow \mathcal{E}_k,$$

where $\mathcal{E}_k$ represents the space of a client k. Fusion model $g_k : \mathcal{E}_k \longrightarrow \mathbf{R}$ is also locally defined. This method of training the common model, allows individual clients to be trained and to perform predictions independently from other client; in other words, client k will not use features are held by client $k \neq j$, $k, j = 1, 2, 3, ...N$. To ensure that missing blocks do not prevent us from training the common model, we should aim to perform prediction on any possible subset of blocks $\mathbf{P}(\mathcal{K})$, where $\mathbf{P}(\mathcal{K})$ represents powerset of $\mathcal{K}$[4].

The advantage of defining local predictors, together with local fusion model is that it will allow training of singletons. However, the drawback of this method is that, similar to standatd VFL, the training of intermediate blocks. [3]

Our implementation accounts for missing blocks and trains blocks of intermediate sizes during training and inference.

[3] Blocks that are neither the entire block $\mathcal{K}$, nor singleton $\{i\} : i \in \mathcal{K}$.

4 Related Work

In HFL, prior works have explored the impact of client participation on training dynamics and robustness to dropouts. For instance, [5] introduced FedAvg, which demonstrated resilience to partial client participation but did not address vertical feature-specific dropouts. In contrast, VFL focuses on aligning feature-based contributions across multiple clients. [3] emphasized the role of feature importance in VFL by proposing methods to quantify client contributions based on their intermediate activations. Similarly, [7] investigated the effects of client selection and feature redundancy on convergence, highlighting the importance of identifying critical clients for robust learning.

Optimizer adaptability has also been studied as a means to counter dropout effects. [6] analyzed the role of adaptive optimizers like Adam and RMSProp in mitigating the performance degradation caused by uneven participation. Furthermore, studies such as [2] demonstrated that dropout-aware strategies could improve both fairness and robustness by balancing contributions across clients with varying importance. Despite these efforts, limited work addresses the interplay of feature importance-based dropouts and model convergence in VFL setups, motivating this study to bridge the gap.

Another related type of machine learning is federated transfer learning, or FTL. in FTL, datasets differ in both sample and feature spaces while having limited overlap. [4]. This method uses techniques from both HFL and VFL.

Regarding VFL in particular, Liu et. al. [4] have proposed a general training protocol for VFL. The first step is to align the data used for the training, a process known as entity alignment. This is done by finding common sample IDs using private set intersection techniques without revealing the IDs themselves. One possible future direction of research is to use a coupled design for fuzzy identifiers [4].

The next step after alignment is to begin training begin training the model using the aligned samples. The most commonly used training method is the gradient descent method, although many other methods are available [1]. This requires involved parties to transmit local model outputs along with corresponding gradients to the central server [4].

Because gradients and other data are being sent to the central server, special care must be taken to prevent privacy leakage. Examples of crypto based privacy preserving techniques are Homomorphic Encryption, Secure Multi-Party Computation and Trusted Execution Environment [4]. These encrypt the gradients being sent to the main server, protecting the information from both inside and outside attackers.

Utilizing these privacy preserving techniques is vital to ensuring confidentiality of each client's data, a crucial idea in any type of federated learning.

5 Experiments

5.1 Experimental Setup

Our VFL model is a neural network consisting of multiple client models and a central server model. Each client model has a single hidden layer with 128 units and an output layer of 64 dimensions, enabling it to process its respective input features independently. The outputs from all client models are passed to the server model where they are simply concatenated. These concatenated units are used by the server as input for final predictions. In our implementation, the server also has a hidden layer with 64 units and an output layer of size 10. We use the MNIST dataset for all our experimental evaluations. We were only able to use 10000 data points in MNIST due to time and resource constraints. For all scenarios, we ran our model for 60 epochs and 10 iterations and divided the features of MNIST dataset equally amongst all clients.

The training process for our model is configured with certain hyperparameters to ensure effective optimization and convergence. The server model is trained with a learning rate of 0.001, while the client models use a slightly lower learning rate of 0.0005 to allow more controlled updates from clients. We used a batch size of 512 and the model is trained for 60 epochs. The entire process is repeated for 10 iterations to ensure consistent performance evaluation.

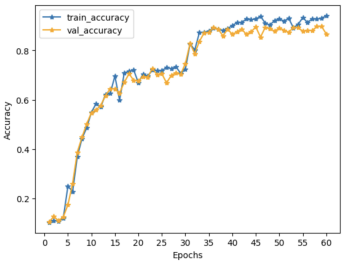
5.2 Feature importance of clients

We evaluate the feature importance of each client by measuring its impact on the model's performance. We compare the test accuracy when the client's inputs are included versus when they are not included. We take the difference in accuracy between these two scenarios as a measure of the client's feature importance.

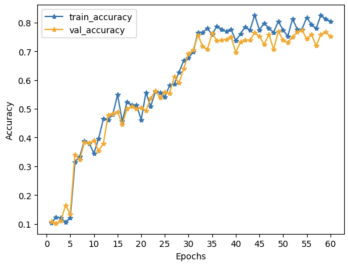
Table 2 shows feature importance results for VFL model with 4 clients and a central server. We see that disabling Client 2 resulted in the most significant drop in test accuracy, from 0.77805 to 0.6839, indicating that its features are the most critical for the model's performance. Similarly, removing Client 4 led to a notable decrease, followed by Client 3 and Client 1. Based on the magnitude of the accuracy drops, the importance of the clients can be ranked as follows: Client 2 > Client 4 > Client 3 > Client 1. From this ranking, it is clear that some clients contribute more substantially to the model's predictive capabilities than others. Note that central server is always active.

Table 1: Impact of Client Dropout on Test Accuracy in VFL model with 4 clients and 1 server

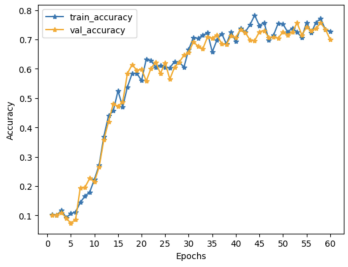
Configuration	Test Accuracy
All Clients Active	0.77805
Client 1 Dropped	0.7209
Client 2 Dropped	0.6839
Client 3 Dropped	0.7019
Client 4 Dropped	0.67965



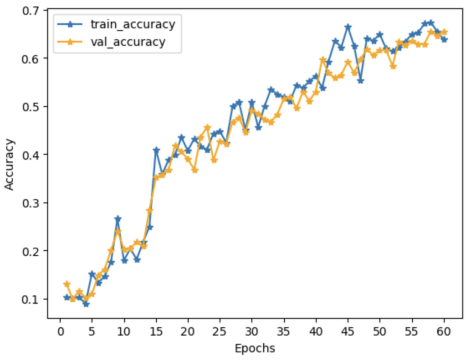
(a) All Clients Active



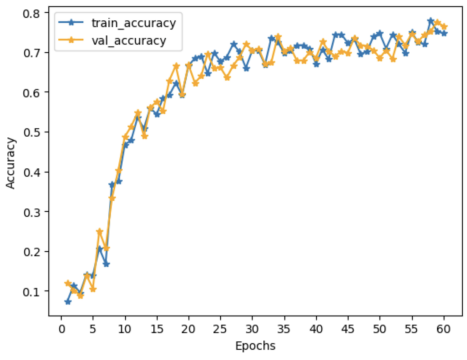
(b) Client 1 Dropped



(c) Client 2 Dropped



(d) Client 3 Dropped



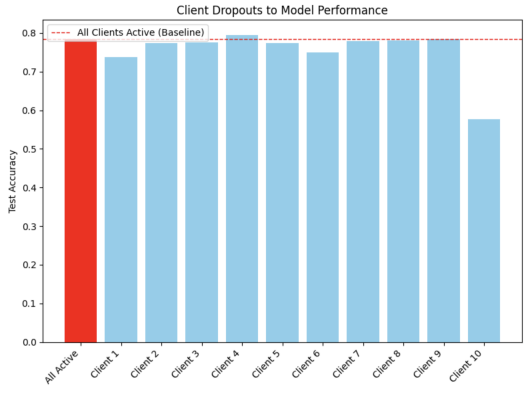
(e) Client 4 Dropped

Figure 1: Convergence plots for different client dropout scenarios in the VFL model with four clients and a central server

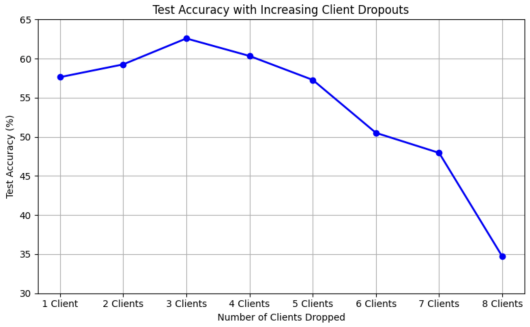
Figure 2a shows the impact of disabling individual clients on the test accuracy for a 10-client scenario. Disabling Client 10 leads to the largest drop in test accuracy suggesting that its features are the most critical for the model's performance. Clients 2, 3, and 8 have a minimal impact when disabled, indicating their features contribute relatively less to overall performance. However, disabling Client 4 slightly improves accuracy to 0.7940, which suggests its features may introduce noise or redundancy.

5.3 Client dropouts in the middle of training

In order to implement this scenario, we disabled most important client to the least in our 10-client scenario. We observed that the test accuracy and convergence remain stable even after dropping clients in decreasing order of contribution until 50% of the clients are removed. We think the possible reason for this is influenced by two factors:



(a) Test accuracy analysis for the VFL model with 10 clients. The red bar represents the baseline accuracy when all clients are active, while the sky-blue bars show accuracies when specific clients are disabled.



(b) Test accuracy analysis for the VFL model with client dropouts occurring from the beginning of the training process. Clients are dropped in the decreasing order of their importance.

Figure 2: Comparison of test accuracies for client dropout scenarios in the VFL model.

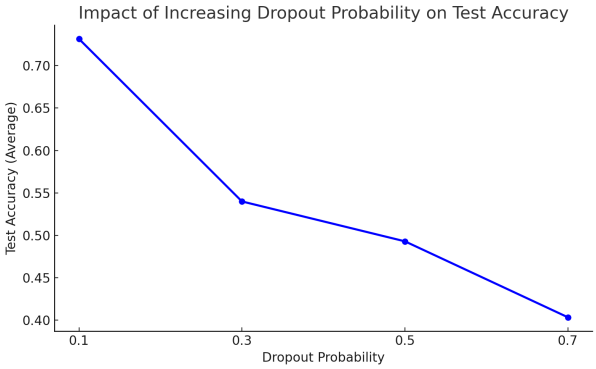


Figure 3: Test accuracy for the VFL model under random client dropouts. The results are averaged over multiple iterations, with clients being randomly deactivated based on different dropout probabilities.

- **The use of the Adam optimizer**: Adam dynamically adjusts of learning rates based on the first and second moments of gradients which may allow it to adapt effectively to changes in training. As clients are dropped, Adam compensates for the loss of features by adjusting the learning rates for the remaining parameters, preventing significant performance degradation.
- **The timing of the client removal**: dropping clients midway through convergence (after 30 epochs) likely minimizes the impact because the model has already learned meaningful patterns from all clients' features during the earlier epochs. This initial learning phase enables the model to generalize well, even as some features (clients) are removed later.

However, we performed the same experiments with client dropouts from the beginning of training. The results for this experiment are shown in Figure 2b. The test accuracy increases slightly as up to 3 clients are dropped, reaching 62.58%, which indicates that the removal of a few clients may help eliminate redundant or less informative features. However, beyond this point, accuracy begins to decline as and when clients are removed. This suggests that the model's performance is adversely affected once a critical amount of information is lost.

5.4 Effect of clients dropping out randomly

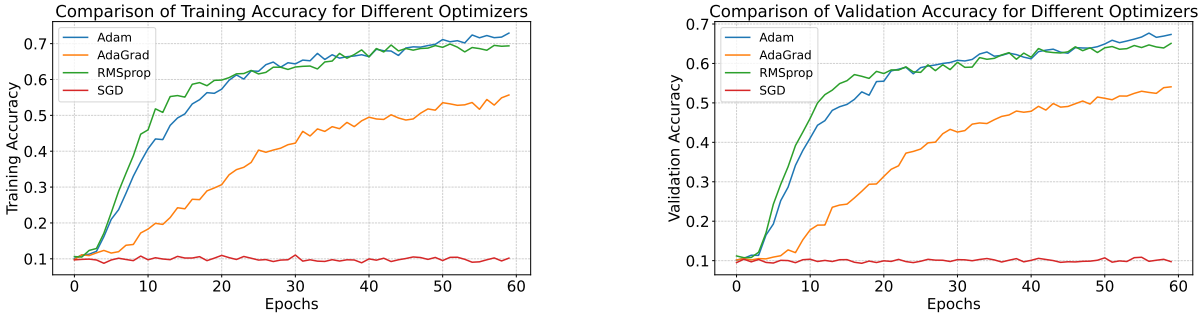
In order to evaluate the impact of random client dropouts, we determine active clients using a dropout probability, and the model is trained with only these clients. Figure 3 shows the impact of increasing dropout probability on the average test accuracy of our VFL model. At a 10% dropout probability, where 3 clients (Clients 1, 7, and 9) are dropped, the test accuracy remains relatively high at 0.7315,

suggesting that the model can compensate for the missing inputs, likely due to feature redundancy. As the dropout probability increases to 30%, with 5 clients dropped (Clients 1, 2, 3, 7, and 9), the accuracy drops to 0.54, with noticeable performance degradation. At 50% dropout probability, where 6 clients are dropped (Clients 1, 2, 3, 7, 8, and 9), the test accuracy declines further to 0.493, reflecting a significant loss of information. Finally, at 70% dropout probability, with 8 clients removed (Clients 0, 1, 2, 3, 5, 7, 8, and 9), the accuracy drops sharply to 0.4035, as the model struggles to generalize due to the extensive loss of client contributions. These results highlight the model's resilience to small dropout probabilities but its increasing sensitivity as more critical inputs are removed, particularly beyond the 50% threshold.

5.5 Effect of different optimizers

In this section, we evaluate the impact of different optimizers (Adam, AdaGrad, RMSprop, and SGD) on the performance and convergence of the Vertical Federated Learning (VFL) model. This experiment uses the same configuration as previous setups, with 10 iterations of 60 epochs each, distributed across 4 clients. Hyperparameters such as batch size and learning rates remained consistent across all optimizers. Based on the results of the previous experiment, we identified the last client as the most critical. This conclusion derives from the observation that when this client dropped out, the accuracy experienced the most significant decline.

To further investigate this phenomenon, we specifically analyzed the scenario where the most important client dropped out after 15 epochs. This setup allowed us to evaluate the impact of the critical client's dropout on the overall performance and better understand the system's resilience to such failures.



(a) Training Accuracy

(b) Validation Accuracy

Figure 4: Comparison of Accuracy for Different Optimizers

Adam: Among all the optimizers tested, Adam demonstrated the best overall performance, with a test accuracy of **67.3%**. Adam dynamically adjusts learning rates based on the first and second moments of gradients, enabling faster convergence and robustness to the changes introduced by client dropout. This optimizer was the most resilient, with strong performance even after the most important client dropped out. The training and validation loss showed steady declines, while accuracy increased consistently over epochs.

RMSprop: RMSprop achieved an average test accuracy of 64.4%. This optimizer works by adjusting the learning rate based on recent gradient values, which helps stabilize training and manage changes in the data. It performed well in maintaining accuracy despite client dropouts, showing stable training and validation trends. RMSprop is effective at handling non-stationary data, making it a reliable choice for this experiment.

AdaGrad: AdaGrad showed consistent improvement in both training and validation accuracy over epochs, achieving a test accuracy of **53.7%**. Its ability to perform well under the dropout scenario can be attributed to its adaptive learning rate adjustment, which handles sparse gradients effectively. However, the model converged relatively slowly, as seen in the training and validation accuracy trends.

SGD: SGD failed to converge effectively, with a test accuracy averaging only **10.0%**. The poor performance of SGD under this scenario can be explained by its lack of adaptive learning rate adjustment. After the most critical client was dropped, SGD struggled to adapt to the sudden reduction in available feature space. This led to oscillations in training loss and small improvements in accuracy, as reflected in both training and validation metrics. Adopting a learning rate scheduler could enhance the performance of SGD by dynamically adjusting the learning rate during training.

Table 2: Impact of different optimizers on Test Accuracy in VFL model with 4 clients and 1 server

Optimizer	Average Test Accuracy
Adam	0.673
RMSprop	0.644
AdaGrad	0.537
SGD	0.100

The analysis highlights the importance of selecting appropriate optimizers in VFL systems, especially under challenging scenarios such as client dropout. Adam emerged as the most robust optimizer, maintaining high accuracy and resilience due to its dynamic adjustment.RMSprop performed moderately well, while AdaGrad struggled with generalization under dropout conditions. In contrast, SGD lacked adaptability, resulting in poor overall performance. These findings underscore the need for adaptive optimizers in federated learning setups, particularly when resilience to client failures is critical.

5.6 Random Feature Sampling

5.6.1 The Need for Sampling Columns During Training

In Vertical Federated Learning (VFL), data is vertically partitioned, meaning that each participating organization (client) holds a subset of features (columns) corresponding to the same set of samples. Training models on the full set of features is computationally expensive and can lead to redundancy if certain features contribute minimally to the task. To address this, column sampling is introduced as a strategy to select a subset of features dynamically during training. This allows the model to focus on the most relevant features while reducing computational costs.

5.6.2 Motivation for Sampling Columns

The MNIST dataset consists of $28 \times 28 = 784$ pixel features for grayscale images of handwritten digits. However, not all pixels are equally important for classification:

- Pixels near the center of the image often contain the majority of the digit's structure and are more relevant.
- Peripheral pixels (corners and edges) tend to be less informative, as they are frequently blank or contain noise.
- Importance of features may also evolve during training, as the model learns to extract useful patterns.

Sampling columns dynamically allows the training process to prioritize informative features while ignoring less relevant ones.

5.6.3 Distributions for Column Sampling

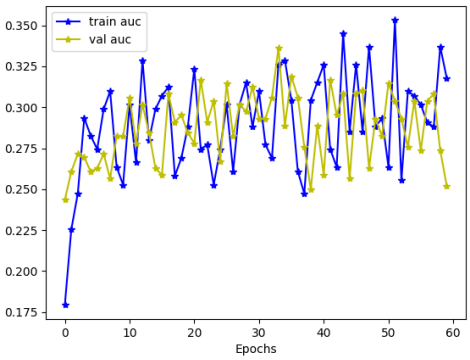
To sample columns effectively, we explore three distributions: uniform, normal, and exponential. These distributions define the likelihood of selecting each feature based on its assumed or observed relevance.

5.6.4 Uniform Distribution

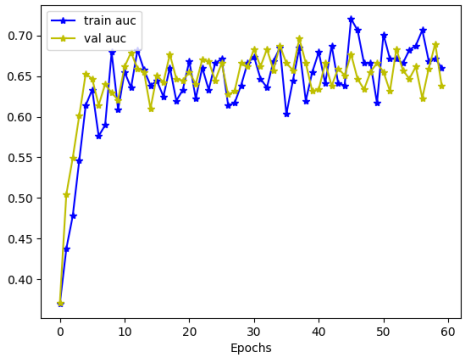
The uniform distribution assumes that all features are equally important. Each feature is assigned the same probability of being selected:

$$P(i) = \frac{1}{d}, \quad \forall i \in \{1, 2, \ldots, d\},$$

where d is the total number of features (e.g., 784 for MNIST). While simple and computationally efficient, this approach does not leverage any prior knowledge or adapt to feature relevance.

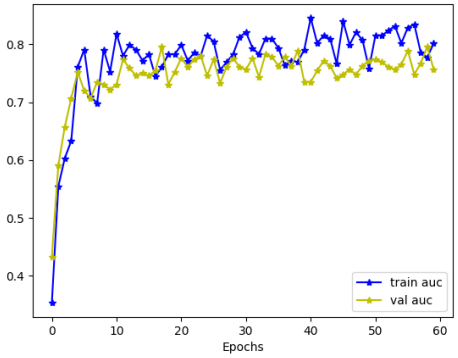


(a) Uniform Distribution with 1% Percent of Data Sampled



(b) Distribution with 5% Percent of Data Sampled

Figure 5: Uniform Distribution Sampling



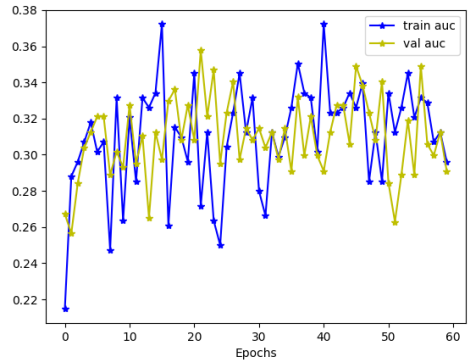
(a) Uniform Distribution with 10% Percent of Data Sampled

5.6.5 Normal Distribution

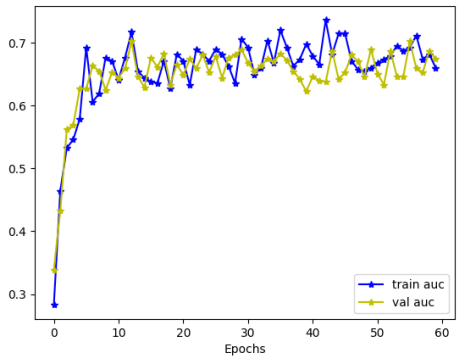
The normal (Gaussian) distribution assigns higher probabilities to features near a specified mean and decreases probabilities symmetrically as we move away. For MNIST, this can be centered on the central pixels (e.g., the middle of the 28×28 grid), as these pixels are more likely to contain parts of the digit. The probability of selecting feature i is given by:

$$P(i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(i-\mu)^2}{2\sigma^2}\right),$$

where μ is the center of the feature space, and σ controls the spread of the distribution. This approach works well if we assume that central pixels are generally more important.

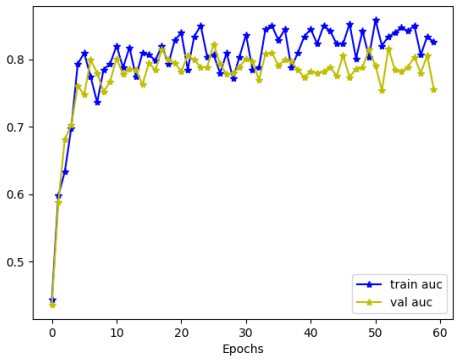


(a) Normal Distribution with 1% Percent of Data Sampled



(b) Normal Distribution with 5% Percent of Data Sampled

Figure 7: Normal Distribution Sampling



(a) Normal Distribution with 10% Percent of Data Sampled

5.6.6 Exponential Distribution

The exponential distribution assigns the highest probability to a small subset of features and decreases probabilities rapidly for the rest. This is particularly useful when the dataset contains a few highly relevant features while the majority contribute minimally. The probability of selecting feature i is given by:

$$P(i) = \lambda \exp(-\lambda i),$$

where $\lambda > 0$ controls the rate of decay. For MNIST, this aligns with the observation that a small subset of pixels (e.g., those defining the edges and curves of a digit) carries the majority of the discriminative information.

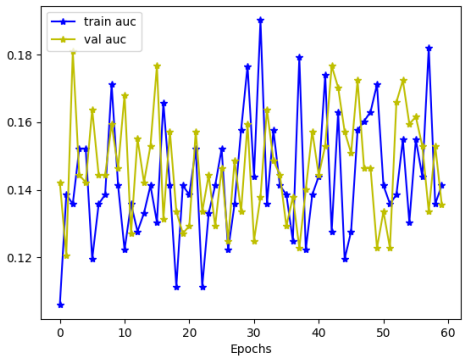
5.7 Experimental Results

5.7.1 No dropouts

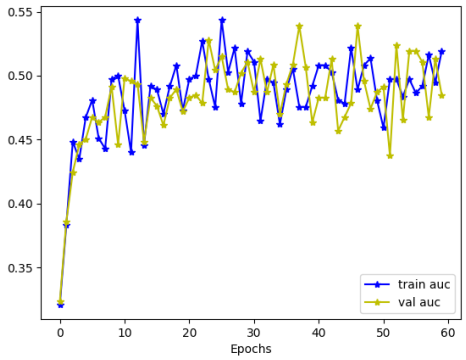
To evaluate the effectiveness of these distributions, we trained a vertical federated learning model on the MNIST dataset, dynamically sampling columns at each epoch. The sampling probabilities for the normal and exponential distributions were updated based on the feature importance, derived from gradients of the loss function with respect to input features.

5.7.2 Conclusion

The experimental results indicate that the exponential distribution outperforms both the uniform and normal distributions for column sampling in VFL on the MNIST dataset. By prioritizing a small subset

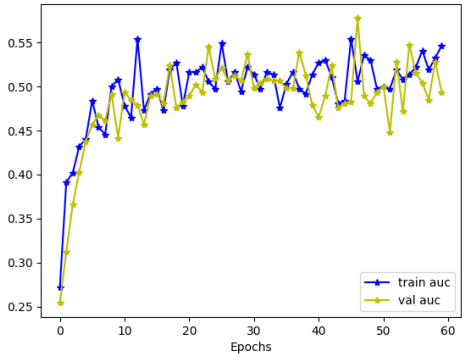


(a) Exponential Distribution with 7% Percent of Data Sampled

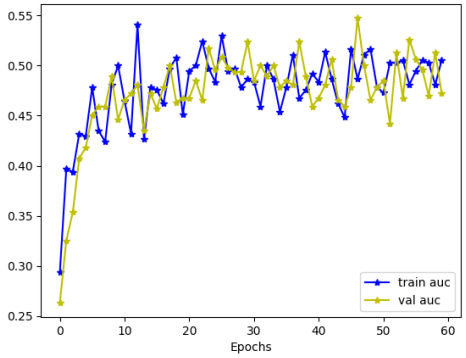


(b) Exponential Distribution with 1% Percent of Data Sampled

Figure 9: Exponential Distribution Sampling



(a) Exponential Distribution with 30% Percent of Data Sampled



(b) Exponential Distribution with 70% Percent of Data Sampled

Figure 10: Exponential Distribution Sampling

Table 3: Comparison of Sampling Distributions for MNIST Classification

Distribution	Test Accuracy (%)	Training Time (s)	Notes
Uniform	[Insert Value]	[Insert Value]	Baseline
Normal	[Insert Value]	[Insert Value]	Focus on central pixels
Exponential	[Insert Value]	[Insert Value]	Emphasized key features

of highly relevant features, the exponential distribution achieves superior classification accuracy while reducing computational overhead. This suggests that the majority of discriminative information in MNIST is concentrated in a small fraction of the features, validating the suitability of exponential sampling for this task.

5.7.3 Random dropouts

In real-world vertical federated learning (VFL) scenarios, participating clients may occasionally become unavailable, resulting in random client dropouts during training. These dropouts reduce the total number of features available for training, which can significantly affect the learning process. To evaluate the impact of such dropouts, we combined exponential feature sampling with random client dropout probabilities of 70% and 35%, simulating extreme and moderate dropout conditions, respectively.

For this experiment, we used the exponential sampling distribution, which prioritizes highly relevant

features while assigning exponentially decreasing probabilities to less important features:

$$P(i) = \lambda \exp(-\lambda i),$$

where $\lambda > 0$ determines the decay rate. At each training step:

- A subset of clients was randomly dropped, based on a dropout probability $p \in \{0.7, 0.35\}$.
- Among the remaining active clients, we sampled either 5% or 10% of the total features using the exponential distribution.

The results of this experiment are summarized in Table 4. For a 70% dropout probability, there was no significant difference in test accuracy when sampling 5% versus 10% of the features. In contrast, for a 35% dropout probability, sampling 10% of the features significantly improved performance compared to sampling 5%, indicating that a lower dropout rate enables the model to better utilize additional features.

Table 4: Test Accuracy (%) Under Different Dropout Probabilities and Feature Sampling Rates

Dropout Probability	Sampling Rate	Test Accuracy (%)	Notes
70%	5%	[Insert Value]	Performance plateaus
70%	10%	[Insert Value]	No significant improvement
35%	5%	[Insert Value]	Limited feature usage
35%	10%	[Insert Value]	Improved performance

For the 70% dropout scenario, the plateauing performance suggests that a threshold is reached in the model's ability to learn. With only 30% of the clients contributing features, increasing the sampled feature rate from 5% to 10% does not provide sufficient additional information to enhance model performance. This indicates that the information loss caused by the high dropout rate cannot be compensated by increasing the feature sampling rate.

By contrast, in the 35% dropout case, the model benefits from the additional features sampled at the 10% rate, as the lower dropout probability ensures that more clients contribute to the training process. This highlights the importance of maintaining a sufficient number of active clients to make effective use of sampled features.

5.7.4 Conclusion

These results demonstrate the existence of a learning threshold under high dropout probabilities. Specifically, for a 70% dropout probability, the model's ability to learn saturates when sampling 5% of the features, and increasing the sampling rate to 10% yields no significant improvement. In contrast, under a 35% dropout probability, increasing the sampling rate improves performance, indicating that the model can effectively utilize additional features when more clients remain active. These findings emphasize the importance of balancing dropout probability and feature sampling rates to optimize learning in VFL.

6 Conclusion

This project investigates the impact of client dropouts—both random dropouts and dropouts based on feature importance—on model performance and convergence in Vertical Federated Learning (VFL). Through experiments with varying dropout probabilities, we observe that the model demonstrates resilience to small dropout rates. However, as dropout rates increase or clients with higher feature importance are removed, the model's performance begins to degrade significantly, particularly beyond 50% dropout probability. This highlights the critical contributions of certain clients and their features to overall model generalization and convergence. Experiments also revealed that redundant or less informative client features could slightly enhance model performance when excluded, highlighting opportunities for feature selection and optimization in VFL systems.

The analysis of optimizer effects demonstrated that Adam outperformed other optimizers, maintaining robust performance even under challenging dropout conditions. AdaGrad showed moderate success, while RMSprop and SGD struggled with generalization and adaptability, especially when critical clients dropped out. These results emphasize the importance of selecting appropriate optimization methods to ensure robustness in federated learning environments.

This study contributes to a better understanding of the dynamics of client dropouts in VFL and offers insights into strategies for enhancing model robustness. Future work could explore advanced strategies to mitigate dropout effects, such as designing adaptive architectures, or leveraging reinforcement learning to dynamically manage client contributions. By addressing these challenges, VFL can become a more reliable framework for collaborative learning, enabling practical applications in real-world scenarios where client availability is uncertain.

References

[1] S. Han L. Wan, W. K. Ng and V. C. S. Lee. "privacy-preservation for gradient descent methods. *13th ACM SIGKDD*, 2007.

[2] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.

[3] Ji Liu, Jizhou Huang, Yang Zhou, Xuhong Li, Shilei Ji, Haoyi Xiong, and Dejing Dou. From distributed machine learning to federated learning: A survey. *Knowledge and Information Systems*, 64(4):885–917, 2022.

[4] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning: Concepts, advances, and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[5] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[6] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[7] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE conference on computer communications*, pages 1698–1707. IEEE, 2020.