GTV: Generating Tabular Data via Vertical Federated Learning

Zilong Zhao[‡¶] , Han Wu[§*], Aad van Moorsel[†], Lydia Y. Chen[‖]
[‡]National University of Singapore, Singapore
[¶]Betterdata, Singapore
[§]University of Southampton, UK
[†]University of Birmingham, UK
[‖]University of Neuchâtel, Switzerland

***Abstract*—Synthetic data has emerged as a promising avenue for privacy-preserving data sharing. However, constructing synthetic data generators necessitates access to the real dataset, posing challenges, particularly when data features are disparately distributed across different organizations. Vertical Federated Learning (VFL) is a collaborative approach to training machine learning models among distinct tabular data holders, such as financial institutions, who possess disjoint features for the same group of customers. In this paper, we introduce the GTV framework for Generating Tabular Data via Vertical Federated Learning and demonstrate that VFL can be successfully used to implement GANs for distributed tabular data in a privacy-preserving manner, with performance close to centralized GANs which assume shared data. We make design choices with respect to the distribution of GAN generator and discriminator models, and we introduce a training-with-shuffling technique so that no party can reconstruct training data from the GAN conditional vector. The paper presents (1) an implementation of GTV, (2) a detailed quality evaluation of the GTV-generated synthetic data, (3) an examination of GTV framework on different data distribution and number of clients, and (4) an analysis on GTV's robustness against Membership Inference Attacks with different settings of Differential Privacy, for a range of datasets with diverse distribution characteristics. Our results demonstrate that GTV can consistently generate high-fidelity synthetic tabular data of comparable quality to that generated by a centralized GAN algorithm. The difference in machine learning utility can be as low as 2.7%, even under extremely imbalanced data distributions across clients. Code is available at: https://github.com/zhao-zilong/gtv**

***Index Terms*—Vertical Federated Learning, GAN, Privacy-preserving machine learning, Tabular data**

I. Introduction

Tabular data, organized in rows and columns, is the most prevalent data format in industry [1]. Recent years have seen the proliferation of the use of synthetically generated tabular data as a privacy-preserving approach for data analysis and product development [2], [3]. The state-of-the-art generation of synthetic tabular data utilizes Generative Adversarial Networks (GANs) [4], [5], [6], [7], [8], [9]. However, training these GANs still requires direct access to all training data, raising an additional privacy concern if the data is coming from multiple sources.

Consider the following scenario. A regional e-commerce company and a bank hold separate information for a set of shared customers. Generating a synthetic dataset that combines the bank's customer income records with the customer's purchasing history in the e-commerce company would be highly beneficial to create more valuable data analysis and associated products. For instance, e-commerce can leverage this synthetic dataset to conduct an in-depth analysis and then launch a stronger compaign at the targeted customer profile. In such cases, collecting and combining data from the bank and e-commerce company to train tabular GANs is not an option, because it involves sharing personal customer information across different organizations. In other words, we require a privacy-preserving approach to train the GANs, which should meet these specific criteria: (i) ensuring that no organization shares its local data; (ii) addressing the unique challenge of disparate features held by different organizations for the same group of individuals; (iii) maintaining the quality of synthetic data comparable to centralized training, where organizations disregard all risks and share their real data on a centralized server for GAN training.

To address the above challenges, we present a novel framework – **GTV** which provides an infrastructure to train GANs for **G**enerating **T**abular data via **V**ertical federated learning. Federated learning [10] is a widely explored technique to train machine learning models on distributed data without sharing the data. Data remains in participating parties that hold data (known as *clients*) and learning is achieved by exchanging model information with a central party (also called the *server*), without sharing the data itself. In *Vertical* Federated Learning, clients hold a local dataset with unique features pertaining to the *same* individuals or other identified subjects. This makes VFL particularly appropriate for various tabular data applications, while image or audio data would more naturally utilize *horizontal* federated learning, in which all clients have data with the same data feature structures. To train a prediction model under VFL [11], each client passes its local data through a bottom model (on the client side) and sends the output to a top model (on the server side) to calculate gradients. This ensures that local data is only accessed by the owning client, and only intermediate results are shared. Previous studies of VFL have concentrated on prediction models such as decision

[*] Corresponding author: h.wu@soton.ac.uk

trees [12] and deep neural networks [13], there is no platform to train tabular GANs using VFL.

The main challenge in deploying GTV is to determine the bottom and top models, such that no data or personal information leaks from the individual clients, and the performance is close to that of a centralized model. By partitioning both GAN generator and discriminator models, placing pieces in each client and in the server, we assure that no data needs to be exchanged but that correlations between different clients' local data can be captured by passing intermediate outcomes to the top model.

Advanced tabular GANs utilize a technique called conditional GAN (CGAN)[5] and GTV supports CGANs. In CGANs, a *conditional vector* (CV) is utilized to specify conditions related to the data being considered during each learning step, such as gender-specific limitations. In GTV, the server distributes the CV to all clients, enabling them to select data for their respective bottom models accordingly. However, this poses a notable security risk: as clients select their local data according to the CV, it is possible for the server to reconstruct the layout of the clients' local categorical features based on the selected indexes. To mitigate this risk, in the end of each training round, GTV implements a mechanism called *training-with-shuffling* in which all clients shuffle their local data using the same random seed. This ensures that the data after shuffling is still consistent across clients, but is not known to the server. Through this mechanism, the mapping between the CV and data index changes each round, making it impossible for server to reconstruct the training data of clients.

We extensively evaluate GTV using nine different partitions of the generator and discriminator neural networks between server and clients, and compare it to a centralized tabular GAN. The experiments are conducted on five commonly used tabular datasets in machine learning. We report on the machine learning (ML) utility and statistical similarity difference between the real and synthetic data. Results show that as long as the top model of the discriminator is large enough, GTV performs comparably to the centralized tabular GAN baseline. In the data partition experiments, we discover that the more imbalanced the number of data columns across the clients, the lower the ML utility of the synthetic data. However, by shifting more neural networks of the generator from bottom to top model, GTV mitigates the negative effects of this imbalance. Additionally, we find that with an increasing number of clients, expanding the size of the generator neural network helps counteract the degradation in synthetic data quality. Finally, we evaluate the vulnerability of GTV to Membership Inference Attacks (MIA). The full black-box MIA from [14] on GTV generated synthetic data shows a low success rate. Implementing Differential Privacy (DP) on GTV can further lower the success rate of MIA, albeit at the expense of GTV's performance.

The main contribution of this study can be summarized as follows:

- We present GTV, the first distributed framework of its kind to integrate state-of-the-art conditional GANs for generating tabular data within a vertical federated learning architecture.
- We introduce a *training-with-shuffling* mechanism for privacy-preserving integration of conditional vectors and propose a secure synthetic data publication strategy to reduce inference attack risks.
- We consider the semi-honest threat model and use it to motivate the design of GTV. We qualitatively analyze the potential privacy risks in GTV training. We experimentally evaluate the robustness of GTV against Membership Inference Attack. Differential Privacy is also studied to implement on GTV to provide protection.
- We evaluate GTV on 5 datasets using 8 metrics for ML utility and statistical similarity. Results show GTV reliably generates high-fidelity synthetic data even with imbalanced client distributions, and we recommend optimal setups based on resource constraints, client count, and data distribution.

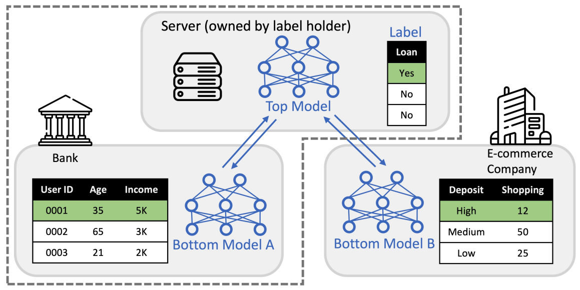


Fig. 1: Traditional VFL architecture for prediction model.

II. Preliminaries

GTV enables training SOTA tabular GANs on VFL. In this section, we describe the preliminary of VFL and GAN.

A. Vertical Federated Learning

As depicted in Fig.1, a typical VFL system involves multiple clients, such as a bank and an e-commerce company, which possess distinct features for a shared group of users. It is important to note that in the illustration, the data instances of the clients have been aligned such that each row corresponds to the same individual across all clients. Data alignment across clients can be achieved by the Private Set Intersection (PSI) technique [15], [16], which is a common assumption used by VFL studies [12], [17], [18]. We also adopt it into our GTV.

The goal of VFL is to train a common machine learning model using the features from all clients. In VFL, there are two types of models: top and bottom models as represented in Fig.1. During the training, each client passes their local data through the bottom model and sends the intermediate logits to the server. In practice, the server is typically managed by the client that holds the label column. As depicted by the dotted line in Fig.1, Bank is the label holder. The server horizontally concatenates the intermediate logits from all the clients and uses the concatenation as input for the top model. The top and bottom models are then updated using the gradients calculated from the discrepancy between the output of the top model and the label.

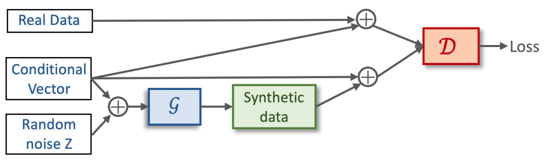


Fig. 2: Conditional GANs (centralized).

B. *Tabular Generative Adversarial Network*

Generative Adversarial Networks (GANs) [4] are a type of machine learning model designed to generate synthetic data that is similar to a given training dataset. They are composed of two parts: a generator ($\mathcal{G}$) and a discriminator ($\mathcal{D}$). The generator is trained to generate synthetic data that is similar to the real data, while the discriminator is trained to distinguish between the synthetic data generated by the generator and the real training data. This competing process continues until the generator produces synthetic data that is indistinguishable from the real data.

To better indicate the category of generation (e.g., 'male' or 'female' in *gender* column), conditional GANs are widely used for table synthesis [5], [7], [8]. Fig. 2 illustrates the structure of a conditional GAN (CGAN), which incorporates an auxiliary conditional vector (CV) to control synthetic data generation. This approach enhances diversity, fidelity, and relevance by ensuring the generated data aligns with specific attributes, surpassing the limitations of traditional GANs[19]. The generator and discriminator are conditioned on the same CV. When using the real data to train $\mathcal{D}$, once a CV is given, the real data needs to sample one row of training data whose class is corresponding to the condition indicated in the CV. It is worth noting that the $\mathcal{D}$ is trained separately by the synthetic data and real data, while during the $\mathcal{G}$ training step, only the synthetic data is used. Feature engineering is another important tool for training tabular GANs. Before entering real data to $\mathcal{D}$, [5], [7], [8], [9] use mode-specific normalization [5] to encode continuous columns, one-hot encoding for categorical columns, mixed-type encoding [7] for column contains both categorical and continuous values. Throughout this paper, we use the terms 'tabular GANs' and 'conditional GANs' interchangeably.

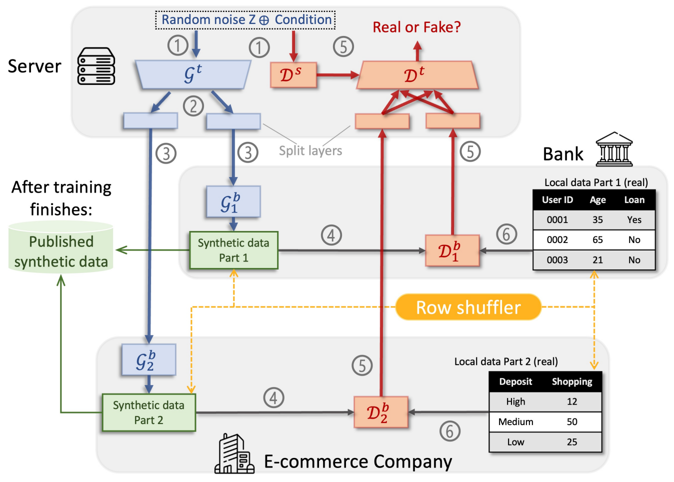
III. Methodology

In this section, we begin by providing an overview of GTV. We then justify the design of GTV by introducing the privacy and security consideration.

A. *GTV Architecture*

In this part, we begin by discussing the components of GTV with an explanation of the design rationale for the structure, then introduce the threat model of our framework followed by a walkthrough of the training process. We then delve into the specifics of training, including possible adjustments. Secure design for publishing synthetic data is explained in the end.

1) GTV overview: Fig. 3 shows an example of GTV with two clients. The notions of the components are provided in Tab. I. Each client contains unique data columns. GTV establishes a separate server instead of assigning the client that holds the label as server, as in Fig. 1. The reason is twofold: (1) In table synthesis, when label column is not needed, no label holder exists. (2) For privacy reasons, clients should not access both the CVs and selected data indexes. The server also should not have access to the `shuffle` function that the client uses to shuffle local data (we discuss this function in more detail in Section III-A4). In Fig. 3, one notable feature is the partition of the discriminator and generator between the server and the clients. The design of the bottom models, i.e., $\mathcal{D}_1^b$, $\mathcal{D}_2^b$ and $\mathcal{G}_1^b$, $\mathcal{G}_2^b$, are also motivated by privacy concerns. First, $\mathcal{D}_1^b$ and $\mathcal{D}_2^b$ are used because clients cannot directly send their real data to the server. Second, $\mathcal{G}_1^b$ and $\mathcal{G}_2^b$ are necessary because otherwise, the output from $\mathcal{G}^t$ would only go through $\mathcal{D}_1^b$, $\mathcal{D}_2^b$ and return to the server. If so, the server could reverse-engineer $\mathcal{D}_1^b$, $\mathcal{D}_2^b$ and reconstruct the clients' data. One may also question the necessity of $\mathcal{D}^t$ and $\mathcal{G}^t$. Without $\mathcal{D}^t$, all generations would be examined only by discriminators in each client. As a result, the generations from different clients would not be correlated because their correlations are never checked. Similarly, without $\mathcal{G}^t$, each client would sample a separate random noise vector for $\mathcal{G}_1^b$ and $\mathcal{G}_2^b$, which would prevent the synthetic data from learning column correlations across clients since there is no correlation between two separate random noise vectors.



Local data Part 1 (real)

User ID	Age	Loan
0001	35	Yes
0002	65	No
0003	21	No

Local data Part 2 (real)

Deposit	Shopping
High	12
Medium	50
Low	25

Fig. 3: The workflow of GTV.

2) Threat Model: In line with many previous works in the field of VFL [12], [20], [17], [18], we focus on the semi-honest model in the design of our proposed GTV architecture. Specifically, we assume that the clients and server are *honest but curious*, and their behaviors obey the following rules:

- All clients and the server comply with the GTV protocol and execute the training process honestly.
- While client or server may attempt to extract additional information from the exchanged messages, none of them intentionally modify the values to disrupt the GTV training.
- Clients do not engage in any collusion to exchange information with other clients or the server except for necessary messages (i.e., the intermediate outputs of the models).

These assumptions form our threat model for the GTV

Algorithm 1: GTV Training Process

```
Input: Number of client $N$; Server side generator $\mathcal{G}^t$ and discriminator $\mathcal{D}^t$; Conditional vector filter $\mathcal{D}^s$; Client side generator $\mathcal{G}_i^b$, discriminator $\mathcal{D}_i^b$ and data $T_i$ for client $i$, $i \in [1, N]$; Training round $R$. Discriminator training epochs per round $e$.
Output: Trained Generator $\{\mathcal{G}^t, \mathcal{G}_i^b\}$, Discriminator $\{\mathcal{D}^t, \mathcal{D}^s, \mathcal{D}_i^b\}$
Client $i$ ($i \in [1, N]$) operates:
1  Encode($T_i$);
2  while current round $< R$ do
       Train Discriminator, freeze $\{\mathcal{G}^t, \mathcal{G}_i^b\}$:
3      while current local epoch $< e$ do
           Server operates:
4          $CV_p$, $idx_p \leftarrow$ CVGeneration(N clients) where the condition column is constructed by Client $p$
5          $\{\mathcal{G}_i^t(Z, CV_p), i \in [1, N]\} \leftarrow$ Split($\mathcal{G}^t(Z, CV_p)$)
           Client $i$ ($i \in [1, N]$) operates:
6          Output $\mathcal{D}_i^b(\mathcal{G}_i^b(\mathcal{G}_i^t(Z, CV_p)))$ as $\mathcal{D}_{out}^{b,i}$ to server;
           Server operates:
7          $\mathcal{D}_{inS}^t \leftarrow$ Concat($\mathcal{D}_{out}^{b,i}$, $\mathcal{D}^s(CV_p)$), $i \in [1, N]$);
8          Output $\mathcal{D}^t(\mathcal{D}_{inS}^t)$;
           Client $i$ ($i \in [1, N]$) operates:
9          if $i == p$ then
10             Output $\mathcal{D}_p^b(T_p[idx_p])$ to server;
11         else
12             Output $\mathcal{D}_i^b(T_i)$ to server;
13         end
           Server operates:
14         $\mathcal{D}_{inR}^t \leftarrow$ Concat($\mathcal{D}_i^b(T_i)[idx_p]$, $\mathcal{D}_p^b(T_p[idx_p])$, $\mathcal{D}^s(CV_p)$), $i \in [1, N]$ and $i \mathrel{!}= p$);
15         Output $\mathcal{D}^t(\mathcal{D}_{inR}^t)$;
16         Calculate Loss$^{\mathcal{D}}$($\mathcal{D}^t(\mathcal{D}_{inS}^t)$, $\mathcal{D}^t(\mathcal{D}_{inR}^t)$) and update $\{\mathcal{D}^t, \mathcal{D}^s, \mathcal{D}_i^b\}$.
17     end
       Train Generator, freeze $\{\mathcal{D}^t, \mathcal{D}^s, \mathcal{D}_i^b\}$
       Server operates:
18     $CV_p$, $idx_p \leftarrow$ CVGeneration(N clients) where the condition column is constructed by Client $p$
19     $\{\mathcal{G}_i^t(Z, CV_p), i \in [1, N]\} \leftarrow$ Split($\mathcal{G}^t(Z, CV_p)$)
       Client $i$ ($i \in [1, N]$) operates:
20     Output $\mathcal{D}_i^b(\mathcal{G}_i^b(\mathcal{G}_i^t(Z, CV_p)))$ as $\mathcal{D}_{out}^{b,i}$ to server;
       Server operates:
21     $\mathcal{D}_{inS}^t \leftarrow$ Concat($\mathcal{D}_{out}^{b,i}$, $\mathcal{D}^s(CV_p)$), $i \in [1, N]$);
22     Calculate Loss$^{\mathcal{G}}$($\mathcal{D}^t(\mathcal{D}_{inS}^t)$) and update $\{\mathcal{G}^t, \mathcal{G}_i^b\}$.;
       Client $i$ ($i \in [1, N]$) operates:
23     Shuffle($T_i$), $i \in [1, N]$;
24 end
```

TABLE I: Summary of Notations

Notation	Description
$\mathcal{G}^t$, $\mathcal{D}^t$	top models of generator, discriminator
$\mathcal{D}^s$	conditional vector filter which runs on server
$\mathcal{G}_i^b$, $\mathcal{D}_i^b$	bottom model of generator, discriminator on client i
T_i	real tabular data held by client i
CV_p	conditional vectors constructed by client p
idx_p	data indexes, the corresponding data contains the indicating classes in CV_p
P_r	ratio vector, the number of features in each client to the total number of features across all the clients

architecture, and our design aims to address potential threats that may arise under these assumptions.

3) GTV Training Procedure: The training process of GTV is similar to that of a centralized tabular GAN, despite the fact that GTV distributes the default tabular GAN into multiple separate components. As the example shown in Fig. 3, the workflow is numerated, parallel actions are noted with the same numbers.

During the discriminator training phase, the process starts with the generator. After initializing the random noise and conditional vectors, they are concatenated and fed to $\mathcal{G}^t$ and $\mathcal{D}^s$ (i.e., ①). The data is then passed through ② to ⑤. After ⑤, the server concatenates the intermediate logits and uses the concatenation as input for $\mathcal{D}^t$ to produce predictions for the synthetic data. At the same time, clients select their real training data following ⑥ then ⑤, and the server concatenates the intermediate logits and produces predictions for the real data. These predictions for real and synthetic data are used to calculate the gradients for updating $\{\mathcal{D}^t, \mathcal{D}^s, \mathcal{D}_1^b, \mathcal{D}_2^b\}$.

During the generator training phase, the process also begins with the generator. The concatenation of the random noise and conditional vectors is passed through ① to ⑤, and $\mathcal{D}^t$ produces predictions. These predictions are used to update $\{\mathcal{G}^t, \mathcal{G}_1^b, \mathcal{G}_2^b\}$. Once all network components are updated, one training round is complete. At the end of each training round, all clients shuffle their local training data using the same random seed to ensure data consistency between them.

4) Training Details: Algo. 1 elaborates the training flow detail. The whole process involves of several functions, namely `Encode`, `CVGeneration`, `Split`, `Concat`, Loss$^{\mathcal{D}}$, Loss$^{\mathcal{G}}$ and `Shuffle`. In step **1**, each client needs to encode their local data. One-hot encoding for categorical column, mode-specific normalization [5] for categorical column and mixted-type encoder [7] for column contains both categorical and continuous values. In step **3**, the training of discriminator needs a local training epochs e because our tabular GAN algorithm is based on Wasserstain GAN plus gradient penalty [21] loss function. This training method needs more discriminator update (by default e = 5) than generator; In step **4**, server uses `CVGeneration` to notify all the clients to generate their local conditional vector (CV) based on the CV construction method proposed by CT-GAN [5], then it randomly choose one of the clients' CVs based on the probability vector P_r calculated by the ratio of number of features in the client to the total number of features across all the clients. The selected client p does not only upload the CV_p

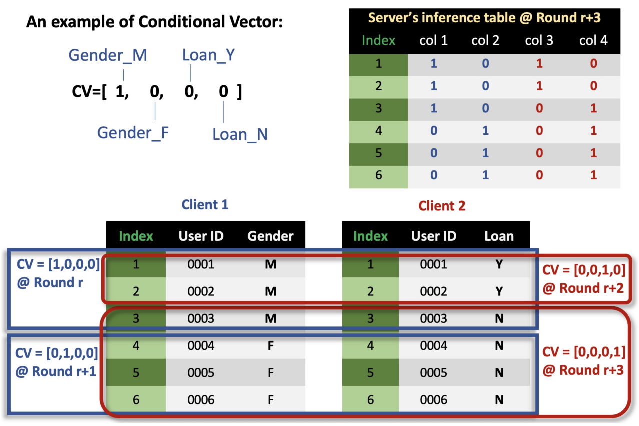


Fig. 4: When GTV works without shuffling, the server can infer the layout of categorical features.

to the server, but also the indexes of selected training data idx_p that meets the conditions represented in CV_p. Due to privacy concern, the idx_p is only kept between the selected client and server, and the potential risk is discussed in Sec. III-A6. In step **5**, `Split` function vertically splits output logits proportionally based on the ratio vector P_r. In step **7**, `Concat` function horizontally concatenates all the intermediate logits into one input $\mathcal{D}^t_{inS}$. During step **9** to **13**, for the client p, it just needs to select out the idx_p training data and passes through the $\mathcal{D}^b_p(T_p[idx_p])$. For the rest of the clients, they need to pass all their local data through the local discriminator; Then in step **14**, except the output from client p, the server needs to first use idx_p to select their outputs, then concatenate all these intermediate logits into one input $\mathcal{D}^t_{inR}$. In step **16**, With $\mathcal{D}^t_{inS}$ and $\mathcal{D}^t_{inR}$, we can use $\texttt{Loss}^{\mathcal{D}}$ to calculate gradients and update all discriminator parts. The training steps for the generator from **18** to **21** are the same as the steps from **4** to **7**. The difference is that to update the generator, function $\texttt{Loss}^{\mathcal{G}}$ (i.e., the step **22**) only needs $\mathcal{D}^t_{inS}$.

5) Training-with-shuffling: At the end of each training round, it is necessary to invoke the `Shuffle` function (represented as the *Row shuffler* in Fig. 3) on all clients to permute the order of their respective local training data's index. Notably, the `Shuffle` function is designed to synchronize the shuffling process across clients. This can be achieved by using Multi-Party Computation (MPC) technology, which has been extensively explored in the context of Federated Learning [22], [18], [23], [24]. This shuffling operation rearranges the order of data rows, while ensuring that corresponding rows across clients remain aligned. Additionally, the negotiation of the `Shuffle` function and random seed among clients is assumed to be conducted securely prior to the training process, independent of the server's involvement.

We highlight the necessity and significance of this approach by illustrating the contrast between GTV training **without** shuffling (Fig. 4) and GTV training **with** shuffling (Fig. 5). Note that the term 'User ID' in Fig. 4 and Fig. 5 serves as a reference for readers to comprehend the shuffling mechanism and is not part of the features or utilized in the GTV training process. The term 'Index' denotes the sequential order of data

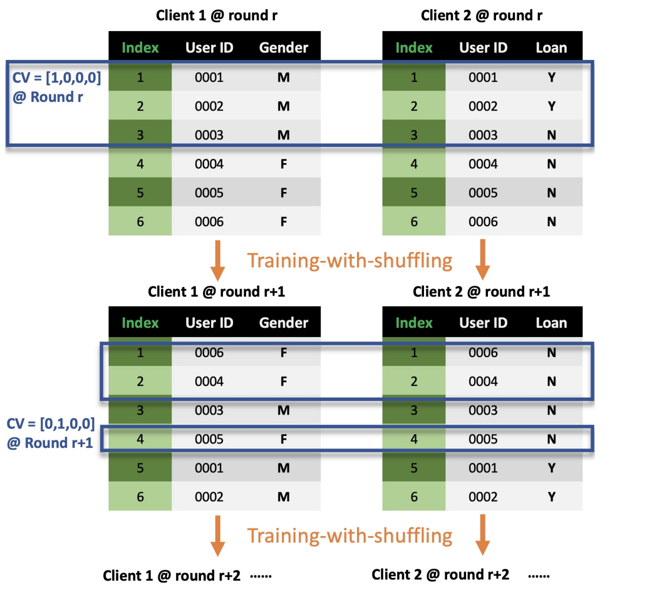


Fig. 5: When training-with-shuffling is enabled, GTV prevents privacy leakage in indexes.

points (rows) within the feature vector.

In the example depicted in Fig. 4, two clients collaborate in the GTV without shuffling. Each client holds a distinct categorical feature (Gender and Loan), and both features have two classes. According to the CV construction method of CT-GAN, the CV of our example contains four bits, each of which represents one class of the categorical columns, and we can only indicate one class per CV. For each training round, we sample new CVs, and the corresponding classes of selected local training data need to match the new CVs. For example, at round r, if our sampled CVs contain three [1,0,0,0] vectors, then the corresponding data indexes (1,2,3) also return to the server (i.e., step **4** in Algo. 1), then round $r+1$ with three [0,1,0,0] in CVs and indexes (4,5,6), round $r+2$ with two [0,0,1,0] in CVs and indexes (1,2), and finally round $r+3$ with four [0,0,0,1] in CVs and indexes (3,4,5,6).

The server can use the indexes and CVs obtained during the training process to reconstruct the entire dataset after a certain number of rounds, as illustrated in Figure 4. Despite not having knowledge of the number of categorical columns or their names in the system, the server can infer this information by analyzing the one-hot encoding of each row. In Figure 4, after round $r+3$, the server can deduce that there are two categorical columns in the system, each containing two categories. Additionally, the server can infer the ratio of categories within each column, such as a 1:1 ratio for first column and a 1:2 ratio for second column. If the synthetic data is released by clients, the server can determine the corresponding column by analyzing the ratio of categories in all categorical columns.

Fig. 5 shows an example of the GTV utilizing *training-with-shuffling*. At round r, we still sample CVs that contain three [1,0,0,0] vectors and get the corresponding data indexes (1,2,3). But this time, we launch the shuffling function in each client. At round $r+1$, we can see that the index column remains the same as last round but the data content part have shuffled, and more important the client 1 and client 2 have the same order in User ID column. If round $r+1$ sampled CVs that contain three [0,1,0,0], the server gets the indexes [1,2,4]. But these indexes in the new round correspond to different data content. Therefore, the server would fail to reconstruct clients' data.

6) Further Discussion: Incorporating CV into the training of the GTV algorithm is not a straightforward task. Since CV is a part of the generator's input, it cannot be hidden from the server. However, we have control over whether to pass selected data indexes to the server. GTV chooses to share this information with the server, so at step **12**, we allow clients who do not contribute CV to pass their entire local data through $\mathcal{D}_i^b$, and then at step **14**, the server selects the corresponding indexes data from the intermediate logits. This design increases local computation and communication between the server and clients, but it ensures that there is no privacy leakage. In order to decrease the overall training time, certain calculations can be performed in parallel. After step **4**, all the clients are aware of whether their conditional vectors have been selected. Therefore, all the clients are able to execute steps **10** or **12** right after step **4**.

An alternative design for GTV involves clients sharing selected data indexes in a peer-to-peer (P2P) manner instead of with the server. While this prevents the server from accessing index information, it introduces significant drawbacks: the exponential cost and insecurity of maintaining P2P connections and indirect privacy leaks. For instance, repeated selection of minority-class data rows can reveal common features in certain columns, making the system vulnerable to inference attacks even if shuffling is enabled. This is because index-level shuffling does not obscure feature mappings across clients.

7) Synthetic Data Publication: To enhance privacy-preservation, clients shuffle the order of their synthetic data before publication using the same `Shuffle` function as in Sec. III-A5. Without shuffling, the server could independently gain *partial black-box access* [14] to $\mathcal{G}$, assuming the server has knowledge of the input random noise, the output synthetic data, and their corresponding input-output pairs. This level of access poses substantial privacy risks. Hence, shuffling before publishing is essential to eliminate partial black-box access by preventing the server from possessing knowledge of the input-output pairs of $\mathcal{G}$. It is important to note that in GTV, nothing beyond the final synthetic data is published. The partial neural networks employed by the clients and server are considered private assets and are not deployed as part of the API.

B. Privacy Analysis in GTV Training Flow

According to the threat model, here we analyze the privacy leakage risk in GTV training, and explain the motivations behind our design. We highlight the sensitive information exchanged and used in GTV, including feedforward activations, and backpropagation process on $\mathcal{G}$ and $\mathcal{D}$, model weights, and model gradients. Here both $\mathcal{G}$ and $\mathcal{D}$ represent the combination of top and bottom generator and discriminator in GTV.

Feedforward and Backpropagation in Generator. In the feedforward phase, each client inputs random noise Z combined with a conditional vector CV into the generator $\mathcal{G}$, which passes through its neural network layers to produce synthetic tabular data, without ever using the real datasets T_i. Even if all clients except one collude with the server, they can only infer synthetic data intended for publication, not real records. During backpropagation, $\mathcal{G}$ updates its model parameters using the loss $\texttt{Loss}^{\mathcal{G}}(\mathcal{D}^t(\mathcal{D}^t_{inS}))$ via Stochastic Gradient Descent (SGD). Unlike traditional VFL classification models, there is no risk of label leakage because $\mathcal{G}$ does not use real data T_i or act as a classifier. While a client might infer its own CV through gradient changes, only the selected client knows which rows are chosen, making CV leakage an insignificant privacy concern.

Feedforward and Backpropagation in Discriminator. The discriminator $\mathcal{D}$ functions similarly to a traditional VFL classifier, but instead of using labels from the original dataset, it relies on *real* and *fake* labels generated by each client, eliminating the label inference risks discussed in [25], [26], [18]. Although intermediate logits sent from clients to the server could reveal some sensitive information, such as matching categorical values [18], GTV mitigates this by keeping the server independent and unaware of any dataset details. In backpropagation, since clients already know the *real* and *fake* labels, traditional label leakage is not a concern. While collusion between malicious clients and the server might expose some data characteristics, this information aligns with the purpose of GTV —producing a shared synthetic dataset—and is considered acceptable within its intended use case.

Model weights and model gradients. The model weights are considered intensive information in VFL studies [12], [18], since the values indicate the relative importance of each feature for the learning tasks. In GTV, this may cause feature importance leakage once the malicious clients share the trained model weights of $\mathcal{D}$ with the server. As mentioned above, the feature importance here is only in reference to the impact of the features on distinguishing real and synthetic data. Additionally, while traditional VFL tasks take the trained model as the final product and it is normal to share the model weights among clients, our GTV sees the joint synthetic dataset as the ultimate outcome. As a result, model weights are supposed to be kept safe and never exposed in GTV.

Recent studies [27], [28], [29] have shown that model gradients can lead to privacy leakage. In [29], the authors present a new algorithm CAFE for recovering large amounts of private data from shared aggregated gradients in the VFL settings. CAFE has a high recovery quality and is backed by theoretical guarantees. However, the assumptions made by the algorithm are quite restrictive and do not apply to our GTV scenarios. One of the key prerequisites in [29] is that the server knows the data index, which is not feasible in our settings. Just as with the privacy of model weights, the risk of data leakage from gradients can only occur when malicious clients are working in collusion with the server.

In summary, when malicious participants act alone, the GTV system is robust to privacy leakage issues present in traditional VFL. However, the system becomes vulnerable when multiple parties collude. This can lead to even more severe security problems, as other advanced attacks may also pose a threat to GTV. These issues are discussed in greater detail in the following section.

C. Privacy Resilience Analysis

Here, we discuss the possibility of state-of-the-art adversarial attacks in the context of GTV.

Feature and Label Inference Attacks. Feature inference attacks, such as those by Luo et al. [30], use generative regression networks (GRN) to infer feature distributions with high accuracy by leveraging correlations between the malicious and victim clients' features. Similarly, label inference attacks aim to extract label information during Vertical Federated Learning (VFL) processes [31], [26], [25]. However, these attacks often rely on unrealistic assumptions, such as the availability of trained model parameters or auxiliary labeled data [30], [25], and are largely constrained to specific scenarios like binary classification [26]. Since GTV does not share trained models or use tabular dataset labels directly in training, these attacks are ineffective and do not pose significant threats in GTV.

Membership Inference Attack. In addition to the aforementioned privacy threats on VFL, a distinct form of attack, known as the Membership Inference Attack (MIA), specifically targets generative models such as GANs [32], [14], [33]. The MIA is executed in the following manner: the adversary is granted access to the machine learning model, which could be black-box or white-box access depending on the specific scenario. The adversary also has access to a set of data records. By conducting MIA, the adversary attempts to infer whether a data record has been used to train the model. This could potentially expose sensitive information.

In GTV, the generator $\mathcal{G}$ is split across clients, making full white-box access by an attacker unlikely without collusion among all parties. Shuffling before publishing mitigates partial black-box access risks, as discussed in Section III-A7. However, within our threat model, a potential risk persists where an adversary could perform MIA using full black-box access. This access permits the attacker to solely retrieve the generated sample set from the well-trained black-box $\mathcal{G}$. In Section IV-C4, we conduct a comprehensive evaluation of GTV's resilience against [14]'s full black-box MIA, specifically assessing the efficacy of Differential Privacy (DP) as a defense measure against this attack.

IV. Empirical Evaluation

A. Experiment Setup

Datasets Our algorithm is tested on five commonly used machine learning datasets: Adult, Covertype, Intrusion, Credit, and Loan. Adult, Covertype, and Intrusion are obtained from the UCI machine learning repository[34], while Credit and Loan are obtained from Kaggle[35]. All five datasets contain a target variable. Due to computing resource limitations, we randomly sample 50K rows of data in a stratified manner with respect to the target variable for the Covertype, Credit, and Intrusion datasets. The Adult and Loan datasets are not sampled. For all the datasets, we train the algorithms 300 epochs for all the experiments. Each experiment is repeated three times and the average result is reported.

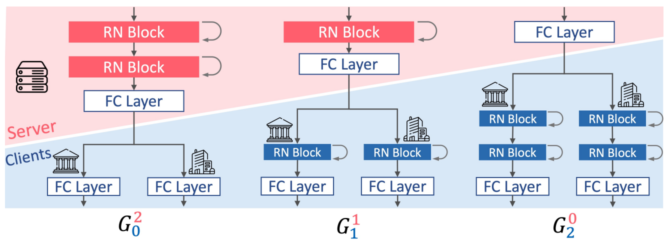
Baseline To the best of our knowledge, GTV is the first architecture to incorporate SOTA tabular GAN into vertical federated learning. Our baseline for comparison is therefore a **centralized tabular GAN**. We enhance the centralized baseline by incorporating techniques from CT-GAN [5] and CTAB-GAN [7], including one-hot encoding for categorical columns, mode-specific normalization for continuous columns, a mixed-type encoder for hybrid columns (i.e., columns contain both categorical and numeric value), and CT-GAN's conditional vector construction method. The generator and discriminator structures mirror those in CT-GAN. The generator is a Resnet-style [36] neural network comprising two residual blocks followed by a fully-connected (FC) layer. Each residual network (RN) block includes three successive layers: FC, batchnorm, and relu. The discriminator has two fully-connected network (FN) blocks followed by another FC layer. Each FN block consists of three successive layers: FC, leakyrelu, and dropout. The output dimension for all the RN block and FN block is fixed to 256.

Testbed Experiments are conducted on two Ubuntu 20.04 machines, each with 32 GB RAM, an NVIDIA GeForce RTX 2080 Ti GPU, and a 10-core Intel i9 CPU. One acts as the server, while the other hosts 2 to 5 clients.

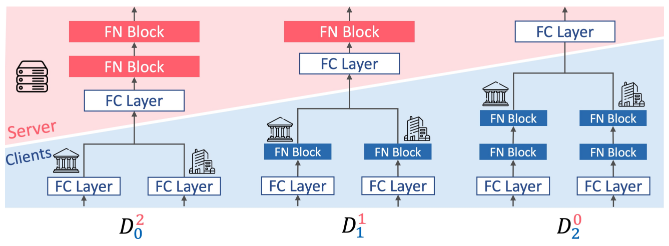
B. Evaluation Metrics

We evaluate the synthetic data in two dimensions: (1) **Machine Learning Utility** and (2) **Statistical Similarity**. The two dimensions cover the row-wise correlation and column-wise distribution. Given the local training data RD_A and RD_B for clients A and B, respectively, the synthetic data produced by the clients is denoted as SD_A and SD_B. Unless otherwise stated, in the following context, synthetic data refers to the concatenation of SD_A and SD_B, and real data refers to the concatenation of RD_A and RD_B.

1) Machine Learning Utility: To evaluate the effectiveness of synthetic data for machine learning tasks, we have designed the following evaluation pipeline. We first split the original dataset into training and test sets. The training set (i.e., RD_A and RD_B) is used as input to the GTV models to generate synthetic data of the same size. The synthetic and real training data are then used to train five widely used machine learning algorithms (decision tree classifier, linear support vector machine, random forest classifier, multinomial logistic regression, and multi-layer perceptron). The trained models are then evaluated on an *independent* real test set. Performance is measured using accuracy, F1-score, and the area under the curve (AUC) of the receiver operating characteristic curve, difference between models trained by real and synthetic data is reported. The goal of this design is to determine whether synthetic data can be used as a substitute for real data to train machine learning models.



(a) Generators under different partition strategies



(b) Discriminators under different partition strategies

Fig. 6: Neural Network Partition. $G_{n_2}^{n_1}$ denotes the generator with n_1 RN block(s) in server and n_2 RN block(s) in each client. $D_{n_4}^{n_3}$ denotes the discriminator with n_3 FN block(s) in server and n_4 FN block(s) in each client.

2) Statistical Similarity: Three metrics are used to quantify the statistical distance between real and synthetic data. (i) **Average Jensen-Shannon divergence (JSD)**. JSD measures the difference between the probability distributions of categorical columns in real and synthetic datasets. It ranges from 0 to 1, is symmetric, and is averaged across all categorical columns for a compact score. (ii) **Average Wasserstein distance (WD)**. WD quantifies the similarity between the distributions of continuous/mixed columns in real and synthetic datasets. Unlike JSD, which can be unstable for continuous columns, WD offers numerical stability. The average WD across columns forms the final score. (iii) **Difference in pair-wise correlation (Diff. Corr.)**. To evaluate the preservation of feature interactions in synthetic datasets, we compute the pair-wise correlation matrix separately for real and synthetic datasets. The dython[1] library is used to compute correlation matrix for each table. The l^2-norm of difference between the real and synthetic correlation matrices is then calculated and abbreviated as **Diff. Corr.**. In the neural network partition experiment with two clients, to examine the preservation of column correlations within each client and between two clients, we first separately calculate the **Diff. Corr.** for each client's data, and then name the average results **Avg-client**, we also report the difference of correlation matrices between both clients' data, the result is named **Across-client**. Assuming clients A and B, with their local real data RD_A and RD_B, respectively, produce synthetic data SD_A and SD_B. Avg-client averages the **Diff. Corr.** of $\{RD_A, SD_A\}$ and **Diff. Corr.** of $\{RD_B, SD_B\}$. Across-client calculates the l^2-norm of difference between the correlation matrix of $\{RD_A, RD_B\}$ (i.e., pair-wise correlation between columns in RD_A and RD_B) and the correlation matrix of $\{SD_A, SD_B\}$. These two metrics are designed to demonstrate the ability of GTV to capture intra- and inter-dependencies of training data columns within and across clients.

C. Result Analysis

The goal of the experiments is to determine the optimal configuration of GTV for synthesizing high-fidelity tabular data while protecting privacy. To this end, we aim to answer four research questions: (1) What is the optimal partition of generator and discriminator neural networks between the server and clients? (2) How does GTV respond to variations in number of data features across clients? (3) How does GTV adapt to changes in the number of clients participating in the system? (4) How to protect GTV generated data from Membership Inference Attacks? To answer these questions, we have designed four experiments. The centralized baseline serves as the basic tabular GAN algorithm for GTV to distribute and train in VFL. In practice, the number of clients in a VFL scenario is constrained by the number of features in the tabular dataset. Therefore, following the VFL experiments setup in [30], [25], [18], our (1) (2) and (4) experiments are performed in a two-client setting. The GTV architecture can be readily expanded to accommodate more clients. We assess the influence of the number of clients in Section IV-C3.

1) Neural Network Partition: For this experiment, we evenly split the training datasets for two clients (or one client may have one more column if the total number of columns is odd), and the column orders are preserved as they were downloaded. As illustrated in Fig. 6, we propose three ways to partition the neural networks of both the generator and discriminator across the server and clients. Both the generator and discriminator have two network blocks (same as in centralized baseline), so we consider three divisions for each of them: (i) all blocks in the server, (ii) one block in the server and one block in each client, (iii) all blocks in the client. This results in a total of nine combinations. The partition notions of $G_{n_2}^{n_1}$ and $D_{n_4}^{n_3}$ are explained in Fig. 6. The sum of the output dimension of partitioned RN/FN block is the same as in their centralized scenario, i.e., 256. The output dimension of each partitioned block is calculated based on the ratio vector P_r (see Tab. I) for each client.

Figure 7 shows the neural network partition results where all results are averaged over five datasets. In all three subfigures, it is clear that the centralized method performs best on all metrics, as expected. Of the other partition methods, $D_0^2G_0^2$, $D_0^2G_1^1$, and $D_0^2G_2^0$ stand out as they consistently outperform the other six configurations on all metrics. These three partition methods have all the FN blocks on the server side. Among these three, $D_0^2G_1^1$ performs slightly better in terms of statistical similarity and slightly worse in correlation difference, but clearly worse in terms of ML utility compared to the other two. $D_0^2G_0^2$ and $D_0^2G_2^0$ show similar results across all evaluation metrics and consistently outperform all other

[1]http://shakedzy.xyz/dython/modules/nominal/#compute_associations

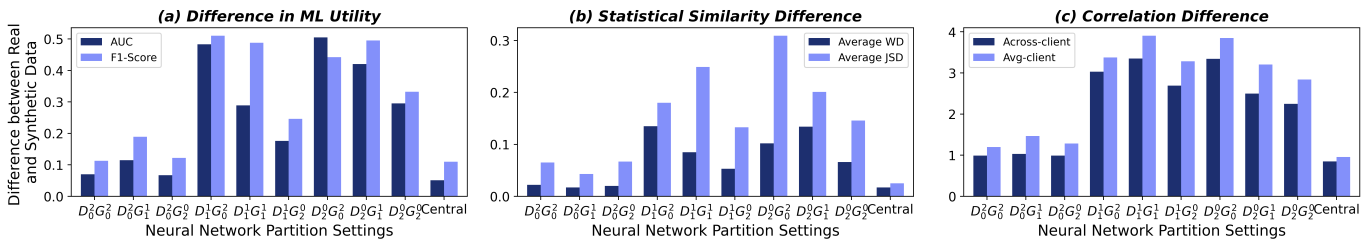


Fig. 7: Neural network partition results: the difference between real and GTV data. A lower value indicates higher quality.

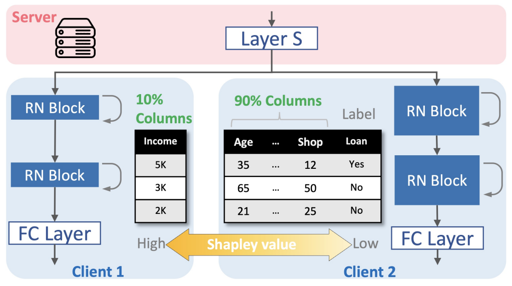


Fig. 8: An example of the G_2^0 generator partition in *1090* data partition experiment.

configurations in terms of ML utility. $D_0^2G_0^2$ achieves F1-score only 2.7% lower to the centralized method. Above result suggests that having a sufficiently large discriminator on the server side is crucial to the success of GTV, and that GTV can achieve performance similar to the centralized method if the discriminator and generator are properly positioned.

Based on the overall performance, we consider $D_0^2G_0^2$ and $D_0^2G_2^0$ to be the two best configurations for GTV. To choose between these two configurations, we must consider the computation and communication overhead of the system. In terms of communication overhead, their only difference is the size of the intermediate logits transferred between the server and clients for generator, which can be controlled by the FC layer before logits are sent from the server to the client. In current setting, $D_0^2G_0^2$ has higher communication overhead; In terms of computation overhead, if the server is powerful, placing both the generator and discriminator on the server can speed up the training process. However, many computations are now performed on the cloud, there is a cost associated with using the server. It may be more cost-effective to distribute calculations to the clients. Additionally, as the number of clients increases, the generator can become larger, making it more expensive to place the entire generator on the server side. Therefore, when $D_0^2G_0^2$ and $D_0^2G_2^0$ achieve the similar result, $D_0^2G_2^0$ is preferred.

2) Training Data Partition: The experiment in Sec. IV-C1 suggests that $D_0^2G_0^2$ and $D_0^2G_2^0$ are two optimal configurations for GTV when data features are randomly and evenly distributed among two clients. In real-world cases, feature distribution across clients is often highly imbalanced. Therefore in this experiment, we examine the impact of data partition on these setups of GTV, in order to assess its robustness against imbalanced feature distribution among clients.

For each dataset, we use the Shapley value [37] to evaluate the importance of each feature in predicting the target column using an MLP (Multi-layer Perceptron) model with one hidden layer containing 100 neurons. We sort the features importance and consider three different feature divisions: (i) **1090**: the 10% most important features and the remaining 90%, (ii) **5050**: 50% most important features and 50% remaining features, and (iii) **9010**: 90% most important features and 10% remaining features. For each division, we assign the two parts to two clients. The target column is always located on the client WITHOUT the most important features. The intuition behind these designs is that it is easier to learn the correlations among features within a single client than learning correlations among features distributed across multiple clients. Fig. 8 shows an example of *1090* data partition for G_2^0. It is worth noting that when partitioning the data, we also split the RN block proportionally among the clients based on the ratio vector P_r. But the sum of the output dimension of the partitioned block is still the same as the centralized baseline, i.e., 256.

Fig. 9 shows the results for $D_0^2G_2^0$ on five datasets. Focusing on the ML utility metrics, we see that data partition has a smaller impact on the final synthetic data for the Adult and Covtype datasets, but a much larger impact on the Loan, Intrusion, and Credit datasets, where the *9010* partition leads to significantly worse results compared to the other two configurations. Regardless of the size of the impact, we observe on almost all datasets that the *1090* partition consistently performs better than the *5050* partition, which in turn performs better than the *9010* partition. This is because each dataset contains several important features for predicting the target column, and as these features are excluded step-by-step from the label holder, it reduces GTV's ability to fully capture the correlations between these features and the label. The results for statistical similarity are similar across all datasets and configurations. Diff.Coff. results align with the results of ML utility, the variations on Loan, Intrusion, and Credit datasets are more obvious than on Adult and Covtype datasets. This result is intuitive since the preservation of column correlations should be related to the ML utility.

The results for $D_0^2G_0^2$ on the five datasets are shown in Figure 10. In terms of ML utility, especially the F1-Score, the *1090* partition performs best on all datasets. The results on Loan dataset are still affected by the data partition, However, it is also clear that $D_0^2G_0^2$ is much less impacted compared to $D_0^2G_2^0$. The results for statistical similarity for $D_0^2G_0^2$ are similar to those for $D_0^2G_2^0$, with little impact from the different data partitions on GTV's ability to recover the column-wise

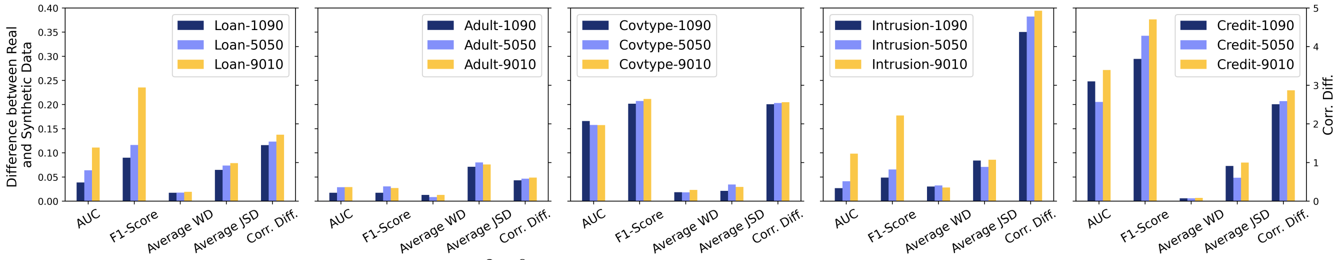


Fig. 9: Data partition results with $D_0^2G_2^0$: the difference between real and GTV data. Corr.Diff. (right axis).

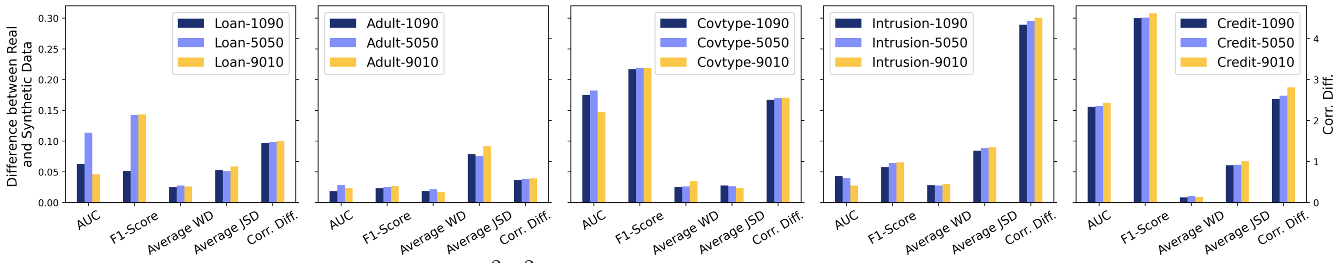


Fig. 10: Data partition results with $D_0^2G_0^2$: the difference between real and GTV data. Corr.Diff. (right axis).

distribution in the synthetic data. The Diff. Corr. shows that the *1090* partition still outperforms the other two configurations. But comparing to $D_0^2G_2^0$, $D_0^2G_0^2$ is less affected by data partition. The reason is that the larger generator on the server side is able to better capture the column correlations across clients. Comparing the result of $D_0^2G_2^0$ and $D_0^2G_0^2$ in the data partition experiment provides additional information to help us choose between these two configurations. In neural network partition experiment, when features are evenly and randomly distributed to clients, $D_0^2G_2^0$ and $D_0^2G_0^2$ achieve similar result, leading us to prefer $D_0^2G_2^0$ due to its lower cost and better scalability. However, when the data partition is extremely imbalanced, especially when the label holder contains significantly fewer features than other clients, $D_0^2G_0^2$ becomes the preferred configuration. One possible solution to increase the performance of $D_0^2G_2^0$ is to increase the size of neural network on the client who contains fewer features, this left for future work.

3) Scalability w.r.t No. of Clients: For this experiment, we study the impact of number of clients on GTV. We randomly and evenly distribute data columns into 2, 3, 4 and 5 clients on $D_0^2G_0^2$ and $D_0^2G_2^0$ configurations. For each dataset, the number of columns is fixed, the more clients in the system, the less data columns for each client. To address the potential performance degradation caused by an increasing number of clients in the system, we introduced two settings for the generator: (1) *default* and (2) *enlarged*. In the *default* setting, for $D_0^2G_2^0$, as the number of clients increases, we simply create more partitions of the original RN block. However, the sum of the output dimension of the divided RN block remains constant, at 256, as in the centralized case. The output dimension of all RN blocks is also 256 for $D_0^2G_0^2$. *Default* setting ensures, no matter how many clients join the system, the communication overhead between server and clients remains the same for $D_0^2G_0^2$ and $D_0^2G_2^0$, respectively; For *enlarged* setting, we increase the output dimension of RN block to 768 (3×256) for both $D_0^2G_0^2$ and $D_0^2G_2^0$, We will explain the reason for choosing this number with the results later.

Fig. 11 and 12 present the results on five metrics, with solid lines representing the results of the *default* setting and dashed lines representing the results of the *enlarged* setting. All results are averaged over five datasets. With *default* setting, the results show that with more clients in the system, i.e., each client contains less features, the ML utility of synthetic data becomes slightly worse. For example, as the number of clients increases from 2 to 5, $D_0^2G_0^2$ and $D_0^2G_2^0$ increase their F1-Score from 0.11 and 0.12 to 0.19, respectively; With *enlarged* setting, ML utility also decreases with increasing number of clients, but to a significantly lesser extent; The results of the Average WD and Average JSD indicate that the statistical similarity on both generator settings for both $D_0^2G_0^2$ and $D_0^2G_2^0$ remains largely unchanged regardless of the number of clients. Again, Coff.Diff. results are inline with ML utility. The result becomes worse with increasing number of clients for both settings. In most of scenarios, $D_0^2G_2^0$ slight outperforms $D_0^2G_0^2$.

Therefore, in that case, $D_0^2G_2^0$ is still the preferred configuration. The results for two settings on generator reveal that the *enlarged* setting has less variation as the number of clients increases. However, when there are only two clients in the system, the *enlarged* setting performs worse on the Loan dataset. The output dimension of the *enlarged* setting is determined through trial and error, highlighting the importance of considering the trade-off between the size of the neural network and the model convergence. It shows that increasing the size of the neural network does not always lead to improved results. Given that the Loan dataset only contains 5000 data instances, increasing the neural network size resulted in slower convergence for the generator in the *enlarged* setting. Optimal neural network size exploration for GTV under different dataset and different number of clients is left for future work.

Note GTV is primarily tailored for cross-silo collaboration

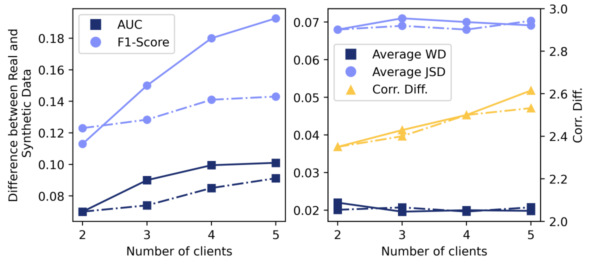


Fig. 11: $D_0^2G_0^2$ setting, the diff. between real and GTV data over varied numbers of clients: default generator (firm line), enlarged generator (dashed line), Corr.Diff. (right axis).

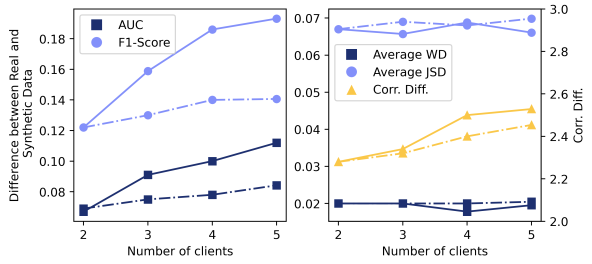


Fig. 12: $D_0^2G_2^0$ setting, the diff. between real and GTV data over varied numbers of clients: default generator (firm line), enlarged generator (dashed line), Corr.Diff. (right axis).

among enterprise-level stakeholders, rather than for cross-device FL systems that typically involve hundreds of clients. In real-world scenarios, involving five clients in GTV would require identifying five distinct organizations, each possessing unique feature sets for the same group of individuals - a condition that is exceedingly rare. Additionally, unlike cross-device FL, where computational resources and network stability are significant constraints due to the involvement of numerous lightweight clients (e.g., sensors, mobile phones), GTV operates under the assumption that substantial computational resources are available. In this sense, GTV can be considered a lightweight framework for enterprises, offering an efficient and scalable solution for cross-silo scenarios.

4) Membership Inference Attack on GTV with Differential Privacy: As we discussed in Section III-C, GTV can be vulnerable to the full black-box MIA [32], [14], [33]. In our experiments, we conduct [14]'s full black-box MIA on GTV, and evaluate the performance of this attack on the five datasets. Moreover, we introduce Differential Privacy (DP) into GTV as it is considered the most effective defense mechanism against MIA [38], [39], [40]. Specifically, we use the DP stochastic gradient descent (DP-SGD) [41] for training the discriminator $\mathcal{D}$ (both client and server side) since the real training data is only fed into $\mathcal{D}$ as shown in Fig. 3. The algorithm can be summarized into a two-step process. Firstly, the per-sample gradient computed during each training iteration is subject to clipping by its $L2$ norm using a predetermined threshold. Secondly, calibrated random noise is deliberately incorporated into the gradient to introduce stochasticity, thereby safeguarding privacy during the training process. In DP-SGD, the privacy budget ϵ determines the level of privacy protection. Smaller ϵ values offer stronger privacy guarantees but introduce larger amounts of noise into the model, leading to a notable decrease

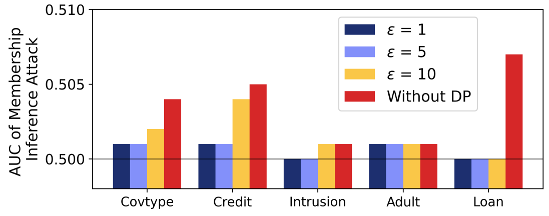


Fig. 13: The performance of Membership Inference Attack (full black-box generator) [14] on GTV ($D_0^2G_0^2-5050$ setting), under different Differential Privacy settings.

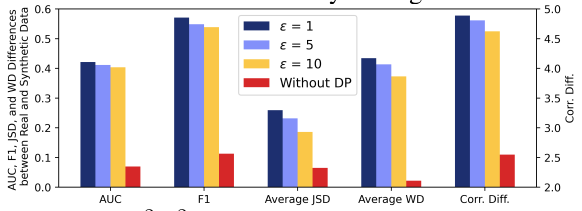


Fig. 14: Under $D_0^2G_0^2-5050$ setting, The impact of Differential Privacy on GTV performance. Corr.Diff. (right axis).

in model performance. In our experiments, we fix the privacy budget ϵ to three commonly used values (i.e., 1, 5, and 10) [42] and observe the impacts.

Fig. 13 presents the performance of MIA (evaluated via AUC scores) under different DP settings, while the impact of DP on VT-GAN performance is evaluated through synthetic data quality in Fig. 14. The experiments are conducted on the $D_0^2G_0^2$ setting, which demonstrates superior performance based on Sec. IV-C1, using randomly and evenly split features across two clients. Notably, even in the absence of DP, the effectiveness of MIA is limited (random guess achieving AUC of 0.5), which is consistent with the findings in [14]. The AUC of MIA for the Loan dataset is slightly higher than that for the other datasets. This is mainly due to the smaller size of the Loan dataset, which makes it more vulnerable to attack [14]. Nonetheless, despite the effect being weak, we observe that for the Covtype and Credit dataset, adopting DP-SGD reduces the AUC of MIA for the Covtype and Credit datasets, with smaller ϵ values providing stronger protection.

The results presented in Fig. 14, averaged across five datasets, demonstrate a significant decrease in the quality of synthetic data when adopting DP-SGD. All evaluated metrics show a notable impact. Specifically, the F1-score difference between models trained on real data and GTV-generated data exceeds 0.5 after applying DP-SGD, which is over five times larger compared to the case without DP. While the impact on JSD is relatively less pronounced, it still triples in magnitude with DP-SGD. Furthermore, a clear trend is observed, indicating that smaller values of the privacy budget ϵ lead to poorer synthetic data quality. Increasing ϵ from 1 to 10 results in some enhancement, yet the improvement remains insufficient when compared to the scenario without DP.

Take Aways The experiments show that a key factor for a successful GTV is having a sufficiently large discriminator model on the server side. When the number of data columns is evenly distributed among two clients in GTV, both $D_0^2G_2^0$ and $D_0^2G_0^2$ exhibit similar performance and are comparable to the centralized algorithm. In cases where these two configurations yield comparable results, $D_0^2G_2^0$, i.e., distributes most of the generator network to client, is preferable due to its superior scalability and cost-effectiveness. However, when data is unevenly distributed among clients, then $D_0^2G_0^2$, i.e., distributes most of the generator network to server, is preferred. If control over the distribution of data columns among clients is possible, e.g., excluding certain overlapping data columns in certain clients, it is recommended to balance the number of columns among the clients in the system. With a fixed total number of features in the system, an increasing number of clients in GTV reduces the quality of synthetic data. Strategically increasing the generator model size can effectively counter this performance degradation. For the security analysis, despite the limited effectiveness of DP in countering MIA, its adoption in GTV significantly diminishes the quality of synthetic data, leading us to discourage the use of strong DP measures.

V. Related Work

A. Vertical Federated Learning

While Horizontal Federated Learning (HFL) has been thoroughly studied [43], [44], Vertical Federated Learning (VFL) starts to gain more attention more recently [45], [12], [23], [46], [17], [18], [47]. Among them, homomorphic Encryption (HE) is employed in BlindFL[18] to safeguard against privacy attacks, while [47] reduces communication bottlenecks through cached-enabled local updates. Note that none of these studies covers generative models like GANs within a VFL framework.

FedDA [48] trains separate GANs for each participant in a VFL setting to achieve data augmentation. However, the study focuses primarily on image generation, which is not a typical application of VFL. Moreover, it is limited to two clients and does not utilize a conditional vector. Similarly, VFLGAN [49] and VertiGAN [50] both target image synthesis in a VFL context and rely on traditional GAN architectures. None of these VFL frameworks are designed to support state-of-the-art tabular conditional GANs. In contrast, GTV employs a single GAN, integrates a conditional vector, and supports collaboration among multiple clients.

B. Privacy Preserving Techniques

Differential Privacy (DP) [51] has been extensively studied for GANs [38], [39], [40] and in HFL scenarios [52], [53], [54], [55] to provide provable privacy guarantees. In VFL with graph data processing, DP can be achieved by injecting noise into the sample representation [56]. Local Differential Privacy (LDP) has also been applied to VFL tree boosting models in a recent study [57]. However, there is currently no clear guidance on how to apply DP in VFL when dealing with both GANs and tabular data, i.e., the scenario of GTV. Moreover, DP's utility is significantly limited due to the negative impact of noise injection on model performance, leading most VFL studies to deprecate its use [18], [17], [12], [23], [24], [48]. Another solution to enhance privacy in GTV is through the use of multi-party computation (MPC) techniques including homomorphic encryption (HE) and secret sharing (SS) [22], [18], [23], [24]. BlindFL[18] presents the federated source layer based on the HE and SS techniques to achieve promising privacy guarantees without affecting VFL model accuracy. Although this approach is also compatible with the GTV architecture, its high communication cost makes it less desirable compared to the privacy guarantees already provided by the GTV architecture.

C. GANs for Tabular Data

GANs are initially known for its success in image synthesis. However, with its ample application scenarios in areas such as medicine [2] and finance [58], research on GAN for tabular data synthesis also gains attention. Previous research on GANs for tabular data synthesis CT-GAN [5], CTAB-GAN [7] and CTAB-GAN+ [8], which improve data synthesis through preprocessing steps for categorical, continuous, or mixed data types and the use of a conditional vector to reduce mode collapse on minority categories. While there are different generative models, e.g., variational autoencoders (VAE)[5], diffusion models[59], and large language models [60], to synthesize tabular data, GAN remains the most widely used and efficient approach in this area, motivating us to incorporate the GAN structure into VFL.

VI. Conclusion

In this paper, we propose the GTV framework for training tabular data synthesizers using vertical federated learning. GTV introduces a privacy-preserving architecture to train state-of-the-art tabular GANs across distributed clients with unique features and a novel *training-with-shuffling* mechanism to incorporate conditional vectors. Our results demonstrate that GTV effectively captures column dependencies across clients and achieves synthetic data quality comparable to state-of-the-art tabular GANs trained in centralized mode. GTV remains stable even with imbalanced data and varying client numbers, and Differential Privacy against Membership Inference Attacks, while with reduced data utility. Additionally, GTV enables distributed training for high-dimensional tabular data, addressing memory constraints in centralized training. Benchmarking and refining GTV for such efficiency purpose are left for future work.

VII. Acknowledgements

This work was supported in part by the Asian Institute of Digital Finance (AIDF) under grant A-0003504-09-00. This research was partly funded by the NWO Perspectief project, DepMAT, and the SNSF project, Priv-GSyn 200021E-229204. This work was funded in part by UK EPSRC grant "AGENCY: Assuring Citizen Agency in a World with Complex Online Harms" under grant EP/W032481/2.

References

[1] S. O. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," *arXiv preprint arXiv:1908.07442*, 2019.

[2] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart, and J. Sun, "Generating multi-label discrete patient records using generative adversarial networks," *arXiv preprint arXiv:1703.06490*, 2017.

[3] A. Mottini, A. Lheritier, and R. Acuna-Agost, "Airline passenger name record generation using generative adversarial networks," *arXiv preprint arXiv:1807.06657*, 2018.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th NIPS - Volume 2*, Cambridge, MA, USA, 2014, p. 2672–2680.

[5] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," in *NeurIPS*, 2019.

[6] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," *Proc. VLDB Endow.*, vol. 11, no. 10, p. 1071–1083, 2018.

[7] Z. Zhao, A. Kunar, R. Birke, and L. Y. Chen, "Ctab-gan: Effective table data synthesizing," in *Proceedings of The 13th Asian Conference on Machine Learning*, vol. 157, 17–19 Nov 2021, pp. 97–112. [Online]. Available: https://proceedings.mlr.press/v157/zhao21a.html

[8] Z. Zhao, A. Kunar, R. Birke, H. Van der Scheer, and L. Y. Chen, "Ctab-gan+: Enhancing tabular data synthesis," *Frontiers in big Data*, vol. 6, p. 1296508, 2024.

[9] J. Lee, J. Hyeong, J. Jeon, N. Park, and J. Cho, "Invertible tabular gans: Killing two birds with one stone for tabular data synthesis," *NeurIPS*, vol. 34, pp. 4263–4273, 2021.

[10] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[11] K. Wei, J. Li, C. Ma, M. Ding, S. Wei, F. Wu, G. Chen, and T. Ranbaduge, "Vertical federated learning: Challenges, methodologies and experiments," *arXiv preprint arXiv:2202.04309*, 2022.

[12] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," in *The 46th International Conference on Very Large Data Bases (VLDB)*, 2020, pp. 2090–2103.

[13] D. Romanini, A. J. Hall, P. Papadopoulos, T. Titcombe, A. Ismail, T. Cebere, R. Sandmann, R. Roehm, and M. A. Hoeh, "Pyvertical: A vertical federated learning framework for multi-headed splitnn," *arXiv preprint arXiv:2104.00489*, 2021.

[14] D. Chen, N. Yu, Y. Zhang, and M. Fritz, "Gan-leaks: A taxonomy of membership inference attacks against generative models," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 343–362.

[15] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 789–800.

[16] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1243–1255.

[17] F. Fu, Y. Shao, L. Yu, J. Jiang, H. Xue, Y. Tao, and B. Cui, "Vf2boost: Very fast vertical federated gradient boosting for cross-enterprise learning," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 563–576.

[18] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "Blindfl: Vertical federated machine learning without peeking into your data," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 1316–1330.

[19] M. Mirza, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[20] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, and B. Zhang, "Large-scale secure xgb for vertical federated learning," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 443–452.

[21] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," in *the 31st NIPS*, 2017, p. 5769–5779.

[22] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 35–52.

[23] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, 2020.

[24] Q. He, W. Yang, B. Chen, Y. Geng, and L. Huang, "Transnet: Training privacy-preserving neural network over transformed layer," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 1849–1862, 2020.

[25] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A. X. Liu, and T. Wang, "Label inference attacks against vertical federated learning," in *31st USENIX Security Symposium (USENIX Security 22), Boston, MA*, 2022.

[26] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang, "Label leakage and protection in two-party split learning," in *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[27] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

[28] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[29] X. Jin, P.-Y. Chen, C.-Y. Hsu, C.-M. Yu, and T. Chen, "Cafe: Catastrophic data leakage in vertical federated learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 994–1006, 2021.

[30] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 181–192.

[31] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar, "Reducing leakage in distributed deep learning for sensitive health data," *ICLR AI for social good workshop*, 2019.

[32] B. Hilprecht, M. Härterich, and D. Bernau, "Monte carlo and reconstruction membership inference attacks against generative models." *Proceedings of Privacy Enhancing Technologies*, vol. 2019, no. 4, pp. 232–249, 2019.

[33] A. Hu, R. Xie, Z. Lu, A. Hu, and M. Xue, "Tablegan-mca: Evaluating membership collisions of gan-synthesized tabular data releasing," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2096–2112.

[34] U. M. L. Repository, "Uci machine learning repository," https://archive.ics.uci.edu/ml, 2024, accessed: Dec. 3, 2024.

[35] Kaggle, "Kaggle: Your machine learning and data science community," https://www.kaggle.com, 2024, accessed: Dec. 3, 2024.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[38] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, "Differentially private generative adversarial network," *arXiv preprint arXiv:1802.06739*, 2018.

[39] D. Chen, T. Orekondy, and M. Fritz, "Gs-wgan: A gradient-sanitized approach for learning differentially private generators," *arXiv preprint arXiv:2006.08265*, 2020.

[40] A. Torfi, E. A. Fox, and C. K. Reddy, "Differentially private synthetic medical data generation using convolutional gans," *arXiv preprint arXiv:2012.11774*, 2020.

[41] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[42] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth, "Differential privacy: An economic method for choosing epsilon," in *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE Computer Society, 2014, pp. 398–410. [Online]. Available: https://doi.org/10.1109/CSF.2014.35

[43] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[44] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[45] Y. Hu, D. Niu, J. Yang, and S. Zhou, "Fdml: A collaborative machine learning framework for distributed features," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2232–2240.

[46] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.

[47] F. Fu, X. Miao, J. Jiang, H. Xue, and B. Cui, "Towards communication-efficient vertical federated learning training via cache-enabled local updates," in *The 46th International Conference on Very Large Data Bases (VLDB)*, 2022.

[48] J. Zhang and Y. Jiang, "A data augmentation method for vertical federated learning," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–16, 2022.

[49] X. Yuan, Y. Yang, P. Gope, A. Pasikhani, and B. Sikdar, "Vflgan: Vertical federated learning-based generative adversarial network for vertically partitioned data publication," *arXiv preprint arXiv:2404.09722*, 2024.

[50] X. Jiang, Y. Zhang, X. Zhou, and J. Grossklags, "Distributed gan-based privacy-preserving publication of vertically-partitioned data," *Proceedings on Privacy Enhancing Technologies*, 2023.

[51] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[52] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" *arXiv preprint arXiv:1911.07963*, 2019.

[53] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.

[54] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[55] M. Naseri, J. Hayes, and E. De Cristofaro, "Local and central differential privacy for robustness and privacy in federated learning," *arXiv preprint arXiv:2009.03561*, 2020.

[56] P. Qiu, X. Zhang, S. Ji, T. Du, Y. Pu, J. Zhou, and T. Wang, "Your labels are selling you out: Relation leaks in vertical federated learning," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[57] X. Li, Y. Hu, W. Liu, H. Feng, L. Peng, Y. Hong, K. Ren, and Z. Qin, "Opboost: A vertical federated tree boosting framework based on order-preserving desensitization," *Proc. VLDB Endow.*, vol. 16, no. 2, p. 202–215, nov 2022. [Online]. Available: https://doi.org/10.14778/3565816.3565823

[58] S. A. Assefa, D. Dervovic, M. Mahfouz, R. E. Tillman, P. Reddy, and M. Veloso, "Generating synthetic data in finance: Opportunities, challenges and pitfalls," in *ICAIF*, New York, NY, 2020. [Online]. Available: https://doi.org/10.1145/3383455.3422554

[59] A. Kotelnikov, D. Baranchuk, I. Rubachev, and A. Babenko, "Tabddpm: Modelling tabular data with diffusion models," *arXiv preprint arXiv:2209.15421*, 2022.

[60] V. Borisov, K. Sessler, T. Leemann, M. Pawelczyk, and G. Kasneci, "Language models are realistic tabular data generators," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=cEygmQNOeI