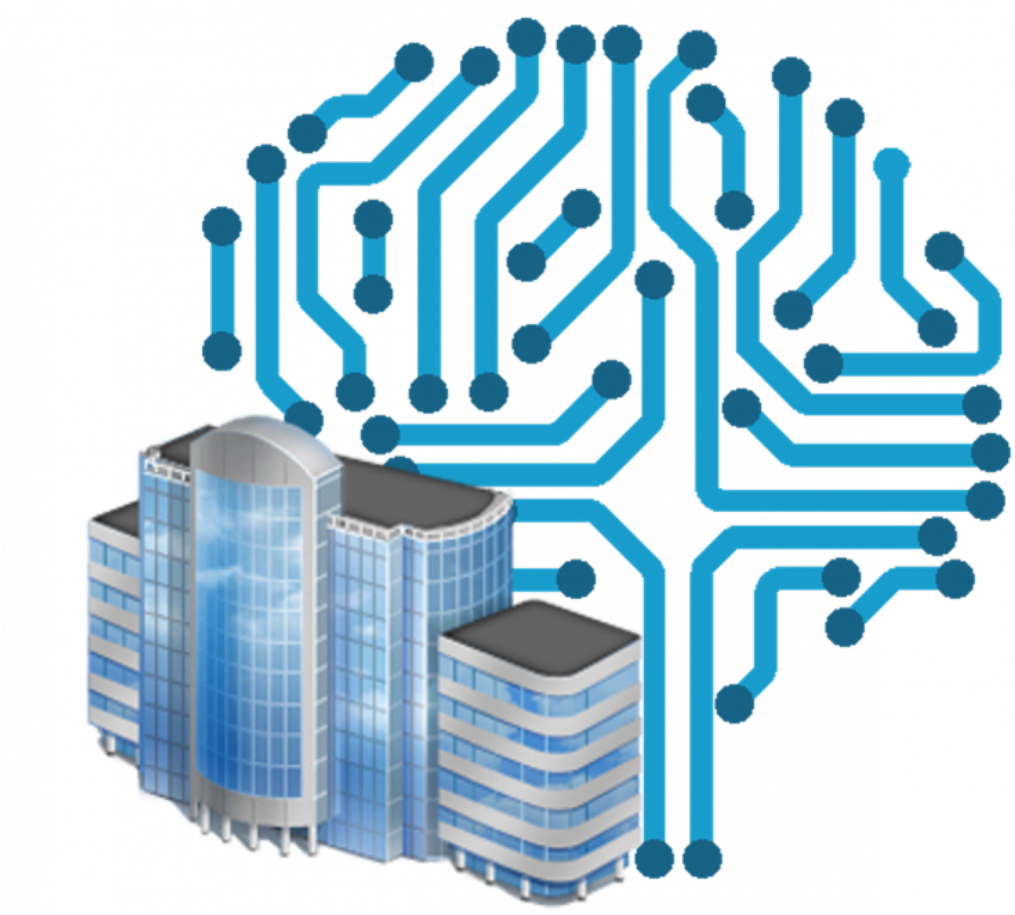
TabVFL: Improving Latent Representation Learning in Vertical Federated Learning



Mohamed Rashad

TabVFL: Improving Latent Representation Learning in Vertical Federated Learning

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Mohamed Rashad

Distributed Systems Research Group
Department of Distributed Systems
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

TabVFL: Improving Latent Representation Learning in Vertical Federated Learning

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday November 14, 2023 at 15:30.

Thesis Committee:

Chair:	Prof. Dr. Lydia Y. Chen, Faculty EEMCS, TU Delft
University Supervisor:	Dr. Jérémie Decouchant, Faculty EEMCS, TU Delft
University Co-supervisor:	Zhao Zilong, Faculty EEMCS, TU Delft
Committee Member:	Dr. Sebastian Proksch, Faculty EEMCS, TU Delft

Preface

I want to express my utmost gratitude to Lydia Chen, Jérémie Decouchant and Zhao Zilong for their profound expertise, kindness and guidance throughout the thesis journey. I also want to thank my friends and family for their invaluable encouragement, patience and persistent support in general. Finally, I want to thank Sebastian Proksch for his friendliness and willingness to be a member of the committee.

Mohamed Rashad
Delft, the Netherlands

Contents

List of Figures

Chapter 1

Introduction

Deep Learning (DL) is a branch of machine learning (ML) that specifically deals with deep neural networks. A Deep Neural Network (DNN) tries to roughly imitate the workings of a human brain. A typical DNN structure is a feed-forward neural network that is made out of interconnected nodes arranged in layers. The network is trained in an iterative fashion. In each iteration, a prediction is made by forward passing the feature values of data samples through the network. The prediction is compared to the true label value to calculate the associated gradient values using a loss function. The gradients are then used to properly update the weights of the network in the direction that minimizes the loss. What makes DNNs useful is that they can learn complex patterns in the data and make accurate predictions. They have been widely used in a plethora of tasks, including image processing using convolutional neural networks [3], speech recognition [4] and natural language processing [5]. DNNs are known to require a huge amount of high-quality data in order to attain their peak performance by generalizing to unseen data. A special type of a DNN is the autoencoder model shown in Figure 1.1. The autoencoder is designed to reconstruct the input of the data as accurate as possible. It is mainly used to extract relevant feature patterns from the data without any label information. This is done by compressing the feature dimension into a lower dimensional space in order to learn relevant latent representations present in the feature data. Consequently, the learnt underlying patterns of the data could also improve the performance for downstream tasks, such as classification or regression tasks, by capturing only relevant information from the data and discarding noise or irrelevant values.

In the traditional sense, a DNN model is trained in a centralized manner [6]. The server within an institution/company contains the model as well as the dataset used for training. External devices or organizations send their data to the server for training. This technique enjoys the benefit of the data being local to the model which makes it easier to train, preprocess and evaluate the model directly. However, the use of centrally-stored personal data to train ML models raises important privacy issues. It is crucial that this data is handled carefully and securely to protect individuals' privacy by complying with data protection regulations, e.g., GDPR [7] and CCPA [8]. This is especially important when dealing with sensitive information, such as medical records or financial data.

Vertical Federated Learning (VFL) [9, 10] is an emerging paradigm in distributed machine learning that enables the collaborative training and evaluation of a model without

sacrificing data privacy. VFL deals with vertically partitioned data, that is, each participating party has overlapping samples with most data attributes being distinct compared to other parties. An illustration of how the data is partitioned in VFL is shown in Figure A.6. Due to its characteristics, VFL is mainly applicable in situations where different regional organizations, that most likely possess data about the same users, want to collaboratively build a more powerful ML model by aggregating their feature data in a privacy preserving manner. An illustration of applying a DNN model in VFL is depicted in Figure 1.2. Generally, only one participant owns the labels for a particular ML task, while the other parties solely own feature data.

The integration of the autoencoder model in VFL is under-researched [11, 12, 13]. Currently, the imposed design assumes that each non-label holder trains a separate autoencoder locally on its local data. After the training is done, only the latent data is sent to the label holder to train a prediction model on the latent data of the non-label holders. This design could potentially break important feature correlations among the data of the non-label holders, leading to worsened predictive performance on the downstream task.

In this master thesis, we propose the novel framework `TabVFL` that addresses this gap by aggregating the feature results in order to consolidate the feature correlations by learning one latent vector. The most popular type of data in VFL is tabular data, as it is ubiquitous in the industry. Tabular data is characterized by its irregular spatial and temporal properties. In general, DNNs, such as traditional autoencoders, are not designed to capture correlations of tabular data optimally. To address this issue, we integrate the state-of-the-art neural network TabNet [14] into VFL that is specifically designed for tabular data and incorporates an autoencoder-like structure.

A problem that could naturally arise and hinder the training process in a VFL system is the failing of clients. Clients may suddenly become unavailable due to network or computational issues. This issue leads to a degraded model performance due to missing intermediate results from failed clients which are either ignored or replaced with artificial values [15, 16].

The research questions investigated in this thesis are:

1. What design can be employed to effectively capture feature correlations and improve latent representation learning while leveraging TabNet in VFL?
2. How is the data privacy preserved in the `TabVFL` design?
3. How can the disruptive effect of client failures during training be reduced in `TabVFL`?

The master thesis consists of six chapters. In chapter 2 a research paper is provided that compactly summarizes the core contributions and results of this thesis. In chapter 3, more background is given about federated learning and the TabNet model. chapter 4 provides detailed information about prediction models and client failure handling methods in VFL. Further analysis of the privacy of `TabVFL` is provided in chapter 5. In chapter 6, additional experiments are conducted and showcased. Ideas for future work and conclusive statements are given in chapter 7.

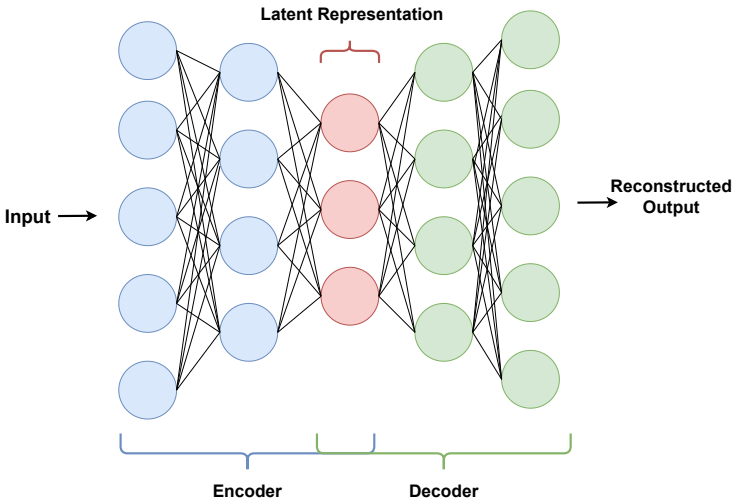


Figure 1.1: An example of the traditional autoencoder model showing the encoder, decoder and the compressed latent representation. The output layer is of the same dimension as the input layer in order to reconstruct the input features.

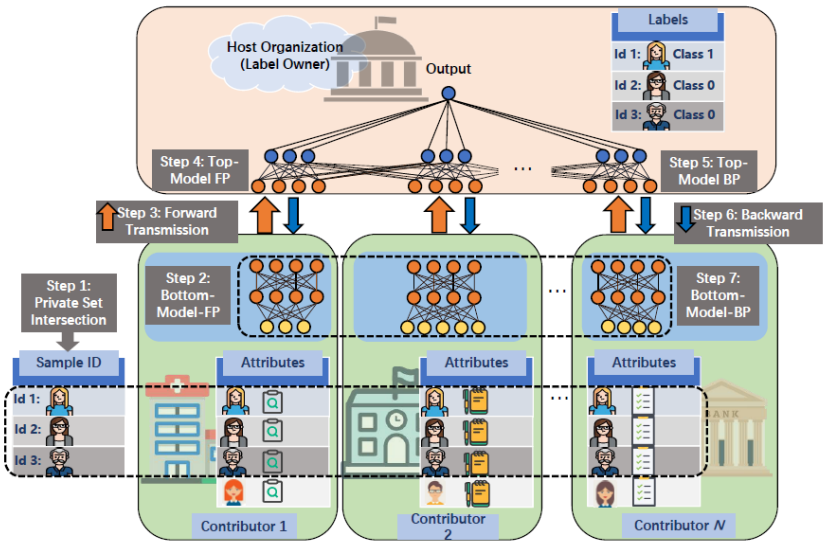


Figure 1.2: A typical VFL system for collaboratively training a neural network prediction model [1]. The neural network is split into different partitions and is assigned to different participants. Different regional contributors align their sample data to correspond feature information with the correct sample. Then, the usual procedure to train a neural network is commenced by transmitting intermediate results and gradients back and forth in each iteration for updating the model weights of each party.

Chapter 2

Research Paper

In this chapter, a research paper associated with this thesis is presented. The paper summarizes the proposed work and the motivation behind it. The results of the conducted experiments are shown with thorough analysis. The paper has been submitted to an international peer-reviewed conference.

TabVFL: Improving Latent Representation Learning in Vertical Federated Learning

***Abstract*—Autoencoders are popular neural networks that are able to compress high dimensional data to extract relevant latent information. TabNet is a state-of-the-art neural network model designed for tabular data that utilizes an autoencoder architecture for training. Vertical Federated Learning (VFL) is an emerging distributed machine learning paradigm that allows multiple parties to train a model collaboratively on vertically partitioned data while maintaining data privacy. The existing design of training autoencoders in VFL is to train a separate autoencoder in each participant and aggregate the latent representation later. This design could potentially break important correlations between feature data of participating parties, as each autoencoder is trained on locally available features while disregarding the features of others. In addition, traditional autoencoders are not specifically designed for tabular data, which is ubiquitous in VFL settings. Moreover, the impact of client failures during training on the model robustness is under-researched in the VFL scene. In this paper, we propose `TabVFL`, a distributed framework designed to improve latent representation learning using the joint features of participants. The framework (i) preserves privacy by mitigating potential data leakage with the addition of a fully-connected layer, (ii) conserves feature correlations by learning one latent representation vector, and (iii) provides enhanced robustness against client failures during training phase. Extensive experiments on five classification datasets show that `TabVFL` can outperform the prior work design, with 26.12% of improvement on f1-score.**

I. Introduction

Autoencoder [1] is a type of artificial neural network used for unsupervised learning of efficient codings. The main goal of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction or feature extraction. However, despite their success, traditional autoencoders are not specifically designed to handle tabular data effectively. Deep Neural Networks (DNNs) have achieved remarkable results in various domains, especially where data has inherent hierarchical or spatial structures, such as images, audio, and text [2]–[4]. However, when it comes to tabular data, DNNs often underperform compared to boosting-based methods like Gradient Boosted Trees (GBT) [5]–[7]. Tabular data often lacks the spatial or temporal hierarchies that DNNs particularly adapt at modeling. Additionally, DNNs can be more data-hungry and prone to overfitting, especially when the dataset is small, whereas boosting methods can be more robust and generalize better with fewer data points.

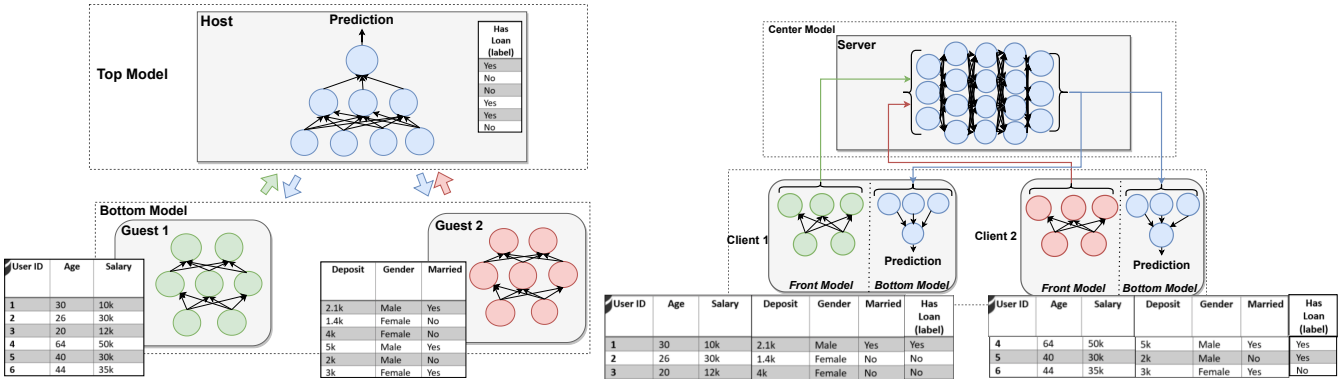
Innovative methods are needed to appropriately adapt DNNs to tabular data in order to address these issues. The incorporation of TabNet [8], a specialized neural network architecture created specifically for tabular data, is a noteworthy development in this domain. With the use of TabNet, tabular datasets can be handled effectively while utilizing the strength of deep learning. Its unique attention mechanism allows it to focus on relevant subsets of features, enabling the model to capture intricate patterns even in the absence of spatial or temporal properties of the data.

Vertical Federated Learning (VFL) [9] describes situations where multiple clients have different features for the same set of individuals. This federated learning approach is especially beneficial when diverse stakeholders or institutions possess distinct data attributes. Consider a bank and an e-commerce platform: while the bank maintains comprehensive financial records, the e-commerce platform holds data on customers' online shopping activities. Both parties aim to collaboratively develop a precise credit scoring model to gauge the risk of a customer defaulting on a loan or credit payment, without directly sharing customer data. In such scenarios, VFL becomes an invaluable tool.

Autoencoders in VFL are substantially under-researched. In the current designs [10]–[13], each non-label holder is assumed to have a local autoencoder model which is trained only on local feature data. After local training, the learnt latent representation is sent to the label owner. Once received, the latent representations are concatenated and used for downstream tasks, e.g., for training a prediction model. The correlation between the isolated features among the non-label holders is not captured due to local latent representation learning of separate autoencoders. This lack of correlation could lead to the generation of irrelevant or noisy latent data, significantly impacting the prediction performance. To fill in this gap, we propose a joint latent representation learning design, `TabVFL`, that learns one latent representation from capturing feature dependencies across all parties involved in a privacy-preserving fashion. SplitNN [14], [15] approach is leveraged to assimilate TabNet in the VFL context. This entails the devision of TabNet into independent model parts that are assigned to each participating party.

Splitting TabNet is not a trivial task due to various dependencies between components. Careful design choices needed to be made in order to protect against the exposure of raw feature data. A fully-connected layer is added in each non-label owner model part to prevent the potential occurrence of a direct data leakage.

One major challenge that any federated learning system can suffer from is the presence of failing clients [16]–[18]. These failures may happen for a number of reasons, including network problems, hardware problems, or software crashes. Client failures can disrupt the training and hence the



(a) Classical SplitNN Architecture.

(b) SplitNN U-Shape Architecture.

Fig. 1: Predictive model designs using SplitNN: example with two clients.

performance of the system. A naive solution to this problem is replacing missing values of failed clients with vectors of zeros [19]. However, this adds bias to the training and can hinder the performance. We instead propose a caching mechanism for storing and reusing the communicated values to decrease performance degradation and enhance training stability.

We conduct extensive evaluation on `TabVFL` using five classification datasets. The results are compared with three baselines. The accuracy, f1-score and ROC-AUC metrics are reported. The evaluation results show that `TabVFL` performs considerably better than prior work for all the datasets. The highest improvement reached was 26.12% on f1-score. `TabVFL` outperforms other designs in terms of runtime and memory utilization but it has higher communication overhead compared to the baselines. The main contribution of this study can be summarized as follows:

- We design the `TabVFL` distributed framework, which incorporates the state-of-the-art tabular model TabNet into vertical federated learning (VFL) via a single latent representation of vertically partitioned data.
- We incorporate an additional fully-connected layer into `TabVFL` to prevent potential data leakage.
- We introduce a caching mechanism to mitigate the impact of client failures, enhance training process stability and minimize performance degradation
- Extensive experiments are conducted on five classification datasets. Results show that `TabVFL` demonstrates significant enhancement in latent quality, outperforming previous designs. Additionally, `TabVFL` excels in runtime and memory efficiency, and only incurs a moderately increased network consumption compared to other designs.

II. Preliminaries

Vertical Federated Learning (VFL) [9] is a distributed machine learning paradigm that focuses on collaborative model training between multiple parties that provide their locally available data while maintaining data privacy. The parties are distinguished into two types: guest clients and a host client. The host client acts as the server or the federator and the guests act as feature providers. It is generally assumed that the host is the label holder. The features and the corresponding target labels of the parties need to be aligned prior to the training process. Private Set Intersection (PSI) ([20], [21]) is the usual algorithm applied for the cryptographic alignment of samples in VFL ([22], [23]). We assume that the samples among clients are already aligned and that each client owns distinct features. Popular machine learning models for tabular data learning, such as gradient boosting decision trees (GBDTs) ([24], [25]) and generative models ([26]), have been considered in VFL. Participants in tree-based models collaborate to collectively construct a prediction tree, while generative models like GANs are employed for synthesizing tabular data that mimics the patterns in participants' original data.

SplitNN particularly focuses on the neural network based predictors [14], [15]. SplitNN enables different and flexible split configurations of a model. Typically, a splitNN model consists of a bottom and a top model as shown in Fig. 1a. Each guest client processes the local feature data through its bottom model. The generated intermediate results are sent by the guests to the host client. Aggregation takes place of the intermediate results which is further processed by the top model at the host. Finally, a prediction is made and compared to the true value for loss calculation. The acquired loss value is used to update the models by calculating the appropriate gradients. A special kind of split learning design is the U-shaped design shown in Fig. 1b. The parties are called server and client since they don't adhere to the assumed guest and host roles. Clients posses a front and bottom model parts, while the server holds the middle part. This naturally reduces computational cost for clients. The front model creates initial intermediate results from input data. Results from clients are sent to the server for processing. Server-generated results are then sent back to clients for prediction and loss calculation.

TabNet [8] shown in Fig. 2 is a state-of-the-art deep neural network that is specifically designed for tabular data. It combines advanced ensembling concepts and novel sequential attention mechanisms. TabNet training phases consists of pretraining and finetuning. During pretraining, an autoencoder

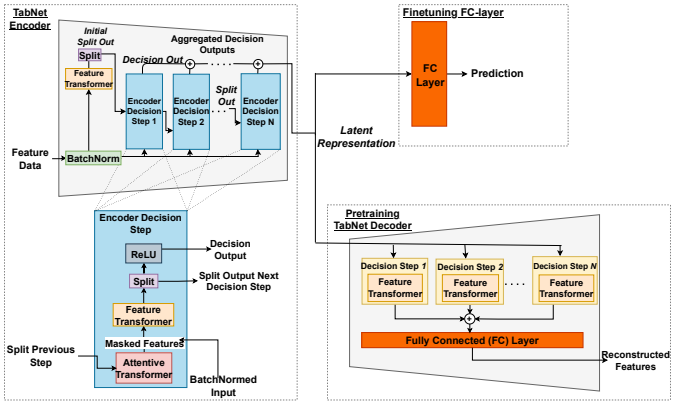


Fig. 2: Structure of TabNet.

[1] structure is employed to learn general latent feature representations. The encoder compresses input features, while the decoder reconstructs them, extracting valuable latent features. For finetuning, the decoder is disregarded and a fully-connected (FC) layer is placed directly after the encoder for predictions. The goal is to specialize the latent representation on a downstream task. TabNet dynamically learns to pick important features through the attentive layer by creating and applying feature masks in each encoder decision step, filtering irrelevant features. The encoder outputs sequential decisions that are processed by the decoder for predicting feature values. During finetuning, the encoder decision outputs are transformed into encoded representations instead.

III. Related Work

In this section, we begin by introducing tree-based prediction models tailored for tabular data within the VFL training framework. Subsequently, we delve into the exploration of deep learning frameworks highlighted in the VFL literature. Finally, client failure handling methods in federated learning frameworks are discussed.

A. Tree-based Prediction Frameworks In VFL

Pivot [24] introduced an enhanced communication protocol to mitigate data leakage from intermediate results for tree models. SecureBoost [25] is able to train a GBDT model, such as XGBoost, without sacrificing model utility by performing similar to non-federated settings. The mentioned frameworks are not efficient due to the sequential nature of training the trees. VF^{2}Boost [27] proposed a scheduler-worker design to enable concurrent computations and an ad-hoc cryptography protocol for faster communication time. A random forest [28] has also been considered in VFL using a trusted third-party to build and achieve a lossless performing model.

TabNet [8] shows superiority over tree-based model on tabular data predictions. We harness TabNet's predictive prowess, integrating it with splitNN to undertake prediction tasks within the VFL training framework.

B. Deep Learning Prediction Frameworks In VFL

SplitNN is applied in multiple VFL use cases, such as healthcare [14], [15], [29], advertisement [30] and finance [31]. In [30], transfer learning is leveraged on overlapped and non-overlapped data among participants to alleviate the issue of missing features. Different configurations for split learning are proposed in [15], [31] depending on different use cases and requirements. Different aggregation methods of intermediate results have been explored specifically in VFL context [18]. Practical application of splitNN is employed in the PyVertical framework [32] where a hybrid approach is utilized to internalize split learning in VFL along with the PSI method for entity alignment. FDML [33] is another VFL framework that also supports neural networks but assumes label availability for all parties.

FedOnce framework [34] enforces one communication round between the guests and host clients using a splitNN design. This is achieved by locally pretraining guest models using a technique called *Noise As Targets* to generate latent representations once. Khan et al. [13] follows the same procedure but a local autoencoder is assumed in each guest client for learning a compressed feature representation. After completion, the latent data is aggregated and prepared for training a separate prediction model in the host side, e.g., using XGBoost [10] on the extracted features. A similar autoencoder design is employed in [11] with the only difference being that the autoencoder is encouraged to learn a higher dimensional representation of the data. In [12] a variational autoencoder is used for learning a latent representation that closely represents a Gaussian distribution in order to ensure statistical similarity between the latent data of different samples, hindering sample distinction by attackers.

Our proposed framework extends the current autoencoder design in VFL by unifying the latent data through training one autoencoder model during pretraining of TabNet. A U-shaped splitNN design (as shown in Fig. 1b) is utilized for joint learning of the feature representations across parties. To the best of our knowledge, none of the existing VFL frameworks considers the application of U-shaped splitNN in cases of feature reconstruction.

C. Client Failure Handling Methods

Many client failure handling methods exist in the Horizontal Federated Learning (HFL) context where it is assumed that clients have overlapping feature space but distinct data samples. Existing techniques include alleviation of client failures by incorporating robust aggregation algorithms [16], [17], increasing eligibility of participating devices in scalable environments [35], preemptive client state prediction [36] and by dynamically varying training round duration [37]. An empirical study has also shown that client failures have significant impact on convergence time and performance of an FL system [38]. However, literature about this matter in VFL is scarce [18], [19], [39]. [18] demonstrated that quitting of clients leads to slower convergence rates and worse performance, amplified by simultaneous quitting of clients in a

TABLE I: Notations

Symbol	Description
PartialEnc	The encoder module of TabNet without BN layer residing in the host client.
PartialDec	The decoder module of TabNet without the FC layer residing in the host client.
$f_{\text{bottom\_repr}}$	The feature extractor containing a BN and a FC layer in the bottom model of guest clients.
$f_{\text{bottom\_repr\_bin}}$	The feature extractor containing a BN, FC and RandomObfuscator layers in the bottom model of guest clients.
$f_{\text{bottom\_rec}}$	FC layer used for feature reconstruction in the bottom model of guest clients.
$final\_mapping$	The final FC layer in the finetuning model of TabVFL used for making a prediction in the host client.

four-client splitNN configuration. Client failures in a two-party splitNN setting are also conducted where a dropout-based method is introduced to enforce robustness against a quitting client [19]. Furthermore, [39] achieves model robustness via dynamic client data re-indexing. In this study, we leverage a caching method to improve model robustness in cases of client failures within a two-party setup.

IV. Methodology

In this section, we first delve into the architecture of TabVFL, elucidating its design and rationale. We then address crucial privacy aspects of the model. Lastly, we present the training specifics, highlighting a novel approach to manage client failures.

A. TabVFL Architecture Overview

In Fig. 3, the structure of TabVFL is shown in a two client setup with an example dataset. Each guest client owns distinct feature data corresponding to the same samples. The novel design is introduced in the pretraining phase of TabVFL. The encoder and decoder components of the TabNet model are split into two parts: the top and bottom models. The majority of the model is retained in the host client. This is done to be consistent with the general model design of U-shaped neural network shown in Fig. 1b.

Many components of the TabNet pretraining model are dependent due to the sequential attention structure which makes a proper splitting of the model among the parties difficult (Fig. 2). Nonetheless, the encoder can be conveniently split by assigning the first batch normalization (BN) layer of the TabNet encoder to the guest clients and the remaining encoder components to the host client. The reason can be deduced from the valuable observation that the BN layer is the only part that every other component depends on since the normalized values are passed to each encoder decision step and to the initial split. Moreover, the decoder can be sufficiently divided by splitting it between the feature transformers and the fully-connected (FC) layer. The FC-layer is further split uniformly into partitions such that each partition corresponds to each guest client. The FC-layers are held locally in each guest. The host only instructs the guests to initialize their FC-layer with the partition dimension. The output dimension of the feature transformers blocks is equal to the latent dimension. Therefore, it should be ensured that the latent dimension is not set to a value that is lower than the number of participating guests. The top model consists of *PartialEnc* and *PartialDec*. The TabVFL pretraining model is shown in Fig. 3a.

The bottom model is represented by two parts: the feature extractor (left part of the bottom model) and the feature reconstructor (right part of the bottom model). The feature extractor is used to transform the raw feature data into intermediate logits. The *RandomObfuscator* is a non-learnable component used to generate binary masks which are directly applied to the features. The reason for the additional FC-layer is to transform the outputs of the BN layer in case it behaves as an identity function. This is possible when the features are already standardized and follow a standard normal distribution, rendering the BN layer useless for protecting against direct data leakage. The feature reconstructor is there to predict the actual feature values that were used as input. The split of the decoder is crucial for preventing the host client from reverse engineering the features of the guest clients. This is possible if the host client has a particular amount of feature samples of some or all the guest clients. The encoder split was done to allow general latent representation learning of one latent vector by aggregating the intermediate logits of all guest clients at the host. The concept of a label is absent during pretraining since the goal is to accurately reconstruct the masked features resulted from the *RandomObfuscator*.

In the finetuning design, the host is the label owner. The conventional splitNN procedure is followed in order to specialize the latent representation from the encoder to a specific task. As shown in Fig. 3b, the decoder components are disregarded including the *RandomObfuscator* component as they are not needed for finetuning.

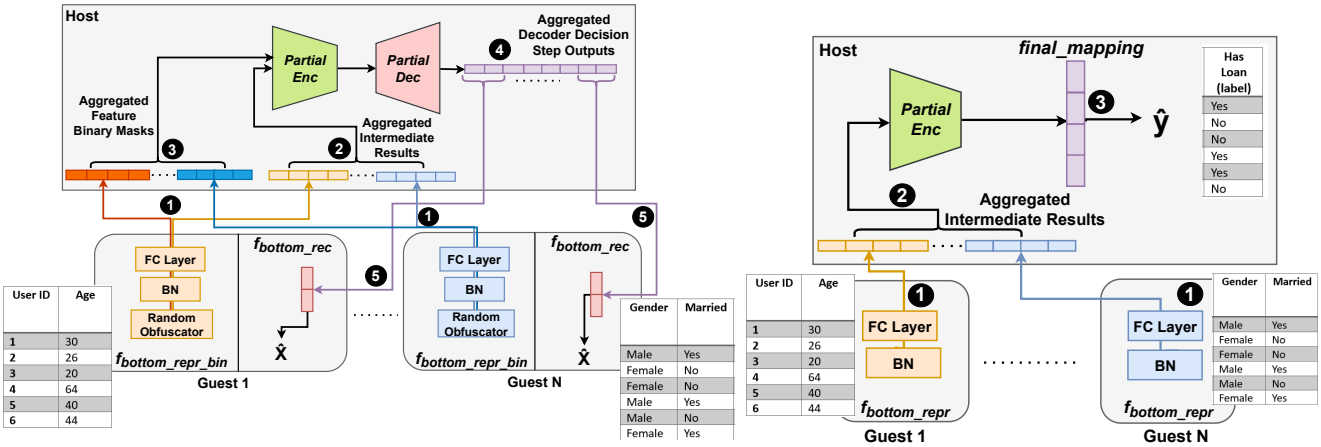
B. Threat Model

We assume honest-but-curious setting where all participants follow the framework protocol properly without any (malicious) deviation. Specifically, we assume that the guest and host clients are honest but curious, meaning that they adhere to the TabVFL protocol, but may seek to acquire additional information through the computation process [40], [41]. We do not consider colluding host and guests. Since the model weights are initialized locally and only intermediate results are exchanged, the models in each client are considered as black box to other clients.

C. TabVFL Training Procedures

Identical training steps as in the TabNet model are followed in TabVFL. In both training phases, the host client is responsible for federating the training process in mini-batch fashion. The training workflow for pretraining and finetuning are provided with an example dataset in Fig. 3. Similar step numbers indicate parallel computations/communications.

During the pretraining phase, the first step entails the generation of the binary masks and the transformed represen-



(a) TabVFL pretraining workflow.

(b) TabVFL finetuning workflow.

Fig. 3: Overview of pretraining and finetuning workflow in TabVFL.

tation of the raw data by the guest clients (❶). The batch of raw features is first masked using the generated masks from *RandomObfuscator* which are processed by a BN and a FC layer and transmitted to the host. The binary masks and intermediate results from the guest clients are aggregated by concatenating them (❷, ❸). The concatenated values are processed by the *PartialEnc* and passed to the *PartialDec* to generate intermediate decoder representations by aggregating feature transformers outputs, which are then partitioned (❹). Each partition result is sent to its corresponding guest client for reconstructing the masked feature values (❺). The reconstruction values are used to calculate the gradients for updating $f_{\text{bottom\_rec}}$, *PartialDec*, *PartialEnc* and $f_{\text{bottom\_repr\_bin}}$.

During finetuning, the raw features are processed by each guest client through the pretrained BN and FC layers and communicated to the host (❶). Subsequently, the intermediate results are gathered and concatenated (❷). The concatenated results are fed through the *PartialEnc* to generate the latent representations which are processed by the $final\_mapping$ for predicting the corresponding label for each sample (❸). The predicted value is used to calculate the gradients for updating $final\_mapping$, *PartialEnc* and $f_{\text{bottom\_repr}}$.

In each training phase, one epoch/iteration is finished once the weights are updated by executing the described workflow.

D. Training Details

The details of the training phases for pretraining and finetuning are presented in Alg. 1 and Alg. 2. The training of TabVFL starts off with the pretraining phase. Each guest first initializes its own bottom models $f_{\text{bottom\_repr\_bin}}$ and $f_{\text{bottom\_rec}}$ (line **1**). Simultaneously, the host initializes the *PartialEnc* and *PartialDec* models (line **7**). The guest clients start the forward propagation by generating binary masks and apply them to the raw feature data. The masked batch is processed by $f_{\text{bottom\_repr\_bin}}$ model and sent to host (lines **4**-**6**). The host receives the intermediate results and the binary masks which are both concatenated into one vector separately (lines **13**-**16**). The results are also cached for the client handling method described in Sec. IV-E. Although the host can generate random binary masks itself, the binary masks are cached instead as well to not break its correlation with the corresponding intermediate results. The concatenated values are fed to the *PartialEnc* to generate the corresponding latent vector. The latent vector is in turn processed by the *PartialDec* to create the intermediate decoder representations (lines **17**-**18**). The decoder results are split into uniform chunks where each chunk corresponds to one guest client. The chunks are sent to each guest for reconstruction (lines **19**-**21**). The guests receive their intermediate decoder results and reconstruct the masked features using $f_{\text{bottom\_rec}}$. The output is compared with the actual feature values for calculating the reconstruction loss (lines **22**-**24**). For backpropagation, one loss value is required. To achieve this, each guest client is ordered to send its loss to the host. The accumulation of the losses takes place and the result is used to calculate the gradients and update the weights of the top and bottom models (lines **25**-**32**). A modified version of the MSE function is used in TabNet to account for the binary mask and feature ranges for a more stable loss calculation.

During finetuning, the weights of the FC and BN layers in $f_{\text{bottom\_repr\_bin}}$ are copied over in $f_{\text{bottom\_repr}}$ for further finetuning (line **1**). The host now uses the pretrained *PartialEnc* and initializes a *final_mapping* containing an FC layer for prediction (line **5**-**6**). The first step of the forward pass is sending the intermediate results, from applying $f_{\text{bottom}_{\text{r}}\text{epr}}$ to the raw features, to the host (line **4**). The host receives the transmitted data from each guest client and caches it to handle possible client failures (line **11**). Similar to pretraining, a vector with concatenated intermediate results is created which is passed to the *PartialEnc*. The *PartialEnc* now also generates *M_Loss* along with the latent vector (lines **12**-**13**). The final logits are created from passing the latent vector through the *final_mapping* layer. Afterwards, the final logits are softmaxed to create normalized prediction probabilities from logits. The

loss is calculated by comparing the prediction probabilities and the actual label values (lines **14**-**16**). Finally, the *M_loss* is multiplied by the regularization parameter λ_{sparse}. The *M_loss* is used for regularizing the sparsity of the feature selection made by the attentive layer of TabNet. This is to encourage the model to only pick important features for datasets that have few features that are relevant. The final loss value is used to calculate gradients and update the weights accordingly (lines **17**-**18**). The classification loss represents the cross entropy.

Note that the host is considered to be the federator which instructs the guest clients to send and receive data during training and inference.

E. Client Failure Handling

Usually in practical cases, variation in the systems of the guest clients can naturally lead to training errors during the training session. This error can occur when a guest client is unable to complete the training due to an unreliable channel or limited data transfer speed, resulting in a failed connection. We propose a caching method for handling abrupt failures of clients to improve robustness and stability during training. The method is implemented in the host client as that is where the intermediate results are handled of each guest client. During training, if a client goes offline then its stored/cached batch of results are reused instead (code lines **10**-**11** in Alg. 1 and **9** in Alg. 2). When a client is online again, the corresponding batches in the cache are replaced by the new intermediate results (code lines **13**-**14** in Alg. 1 and **11** in Alg. 2). Therefore, we enforce that the clients must be online during the initial epoch. To simulate the failures using the proposed framework, the guest clients are given the same probability of being considered offline. Let M be the number of guest clients. Each guest client $\boldsymbol{c} \in M$ follows a binary failure probability distribution:

$$\boldsymbol{p}_c \sim Bernoulli(P\_fail) \quad \forall c \in M$$

We simulate the setting by sampling a float value $\boldsymbol{x}$ from a random variable that follows a uniform distribution for each client, i.e. $\mathbf{X} \sim U(0,1)$. If the sampled value is less than a predefined threshold (P_fail), then the client is skipped for one whole epoch iteration. The skipped clients would not be able to generate any new intermediate results. In a practical setting, the client failures could be handled at the host by setting a certain timeout to wait a bit for a guest client to send its intermediate results. If a timeout occurs, then it is assumed that the guest client is offline or unresponsive. Furthermore, the correct batch index that is being processed by the host should also be communicated to the guest clients to keep the mini-batches aligned throughout the training process during a real deployment. This should be done to prevent offline clients from processing the wrong batch when they come online again.

V. Performance Evaluation

In this section, we first introduce the experimental setup and the baselines. We then showcase the evaluation pipeline, followed by the presentation and analysis of the results.

Algorithm 1: `TabVFL` Pretraining Process

Input:
NumClients = $\{1, 2, ...K\}$
Epochs = E
Feature Dataset X_i of party $i \in \{2, 3, ...K\}$
/* **Guests:** */
initialize local models $f_{\text{bottom\_repr\_bin}}, f_{\text{bottom\_rec}}$
for $e \in E$ **do**
  for *each mini-batch* $X_b \in B$ **do**
    Generate binary mask S_b
    $X_b^{masked} \leftarrow$ Mask X_b using S_b
    Send intermediate result $f_{\text{bottom\_repr\_bin}}(X_b^{masked})$ and S_b to host
    /* **Client 1 (Host)** */
    initialize $PartialEnc, PartialDec$
    for $c \in [2, K]$ **do**
      if c *is offline* **then**
        $\tilde{X} \leftarrow$ reuse cached intermediate results
        $\tilde{S} \leftarrow$ reuse cached binary masks
      else
        $\tilde{X} \leftarrow$ Receive and append intermediate results. Cache received results.
        $\tilde{S} \leftarrow$ Receive and append partial binary masks. Cache received binary masks.
    $X_{intermediate} \leftarrow$ concatenate $\tilde{X}$ column-wise
    $S_{complete} \leftarrow$ concatenate $\tilde{S}$ column-wise
    $Z_{latent} \leftarrow PartialEnc(X_{intermediate}, S_{complete})$
    $out_{intermediate} \leftarrow PartialDec(Z_{latent})$
    $[\tilde{out}_1, \tilde{out}_2, ..., \tilde{out}_K] \leftarrow$ Split $out_{intermediate}$ into uniform partitions for client $c \in [2, K]$
    for $c \in [2, K]$ **do**
      Send $\tilde{out}_c$ to client c
    /* **Guests:** */
    $D_b \leftarrow$ Receive mini-batch decoder intermediate result
    $\hat{X}_b \leftarrow f_{\text{bottom\_rec}}(D_b)$
    $loss_b \leftarrow reconstruction\_loss(X_b, \hat{X}_b, S_b)$
    Send $loss_b$ to host
    /* **Client 1 (Host)** */
    $total\_loss = 0$
    for $c \in [2, K]$ **do**
      if *client* c *is offline* **then**
        continue
      $loss_c \leftarrow$ Receive mini-batch unsupervised loss
      $total\_loss = total\_loss + loss_c$
    Calculate gradients and update the weights

Algorithm 2: `TabVFL` Finetuning Process

Input:
NumClients = $\{1, 2, ...K\}$
Epochs = E
Feature Dataset X_i of party $i \in \{2, 3, ...K\}$
/* **Guests:** */
set weights of $f_{\text{bottom\_repr}}$ to the pretrained BN and FC layers $f_{\text{bottom\_repr\_bin}}$
for $e \in E$ **do**
  for *each mini-batch* $X_b \in B$ **do**
    Send intermediate result $f_{\text{bottom\_repr}}(X_b)$ to host
    /* **Client 1 (Host)** */
    use pretrained $PartialEnc$
    initialize $final\_mapping$
    for $c \in [2, K]$ **do**
      if c *is offline* **then**
        $\tilde{X} \leftarrow$ append cached intermediate results
      else
        $\tilde{X} \leftarrow$ Receive and append intermediate results. Cache the received results.
    $X_{intermediate} \leftarrow$ concatenate $\tilde{X}$ column-wise
    $Z_{latent}, M\_Loss \leftarrow PartialEnc(X_{intermediate})$
    $final\_logits \leftarrow final\_mapping(Z_{latent})$
    $prediction\_proba \leftarrow softmax(final\_logits)$
    $loss \leftarrow classification\_loss(prediction\_proba, y\_true)$
    $loss = loss - M\_loss * \lambda_{sparse}$
    Calculate gradients and update the weights

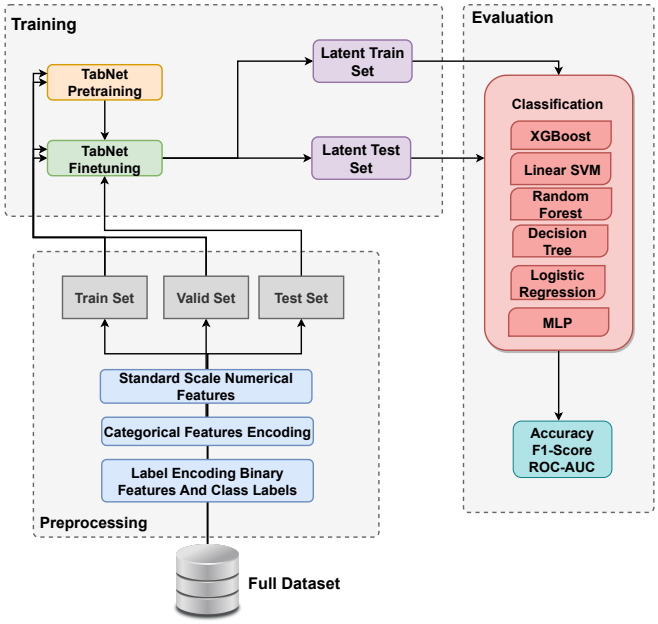


Fig. 4: Evaluation pipeline for evaluating the latent data of different experimental designs.

A. Experiment Details

Setup. `TabVFL` is compared against three baseline designs: Central TabNet (`CT`), Local TabNets (`LT`) and TabNet VFL with guests-assigned encoders (`TabVFL-LE`). `CT` is the baseline where TabNet is run on the full dataset in a non-federated setting. TabNet model implemented with PyTorch [42] is used. The code is modified to disable batch shuffling in each mini-batch iteration and a feature for measuring epoch time is added. The former is implemented to ensure that the experiments differ only in design. `LT` applies the design from prior work [10]–[13] where multiple autoencoders are trained locally in each guest client. The TabNet pretraining model is used instead of autoencoders to keep it consistent with the other baselines. During finetuning, one fully-connected (FC) layer is trained collaboratively in the host. This is done by ensuring that only latent representations are sent to the host. The latent data of each guest are aggregated to make up one latent vector which is processed by the FC-layer in the host to predict a label and calculate the loss. The decision of training one FC-layer at the host (similar to splitNN) is made to fairly compare it with other designs as it resembles the training procedure of finetuning in `TabVFL`. `TabVFL-LE` shares the pretraining model design of `TabVFL`, differing slightly in that the encoder is consolidated within each guest client, unlike TabVFL where it is split into two parts. The outputs of all decision steps are sent as a list of tensors to the host (see Fig. 3a) where the output of each decision step is aggregated by means of summation with the outputs of the same decision step from different guests. The reason is that the decoder accepts a list of encoder decision step tensors. Each encoder decision step corresponds to a decoder decision step. Due to the host receiving all encoder steps from the guests, concatenating each encoder split output and feeding it to the decoder is not feasible. This approach would create incompatible dimensions for each decoder split. Consequently, every encoder decision step output received by the host is aggregated into a single vector, ensuring compatibility with the dimensions expected by the decoder. In Fig. 2 it is shown that encoder steps are aggregated and passed to the decoder. But in the PyTorch code of TabNet [42], the aggregation happens at the decoder. The finetuning phase is identical to `TabVFL` with the only difference being that the latent vectors sent from each guest client to the host are aggregated using summation instead of concatenation as in `TabVFL`. This is done to keep the aggregation method consistent throughout the training phases. This baseline can be seen as a partial transition from `LT` design to `TabVFL` design. We find it interesting to see how the transitioning design affects the evaluation results.

Environment. The experiments have been run on a machine with NVIDIA GeForce RTX 2080 Ti GPU with 11 GB of

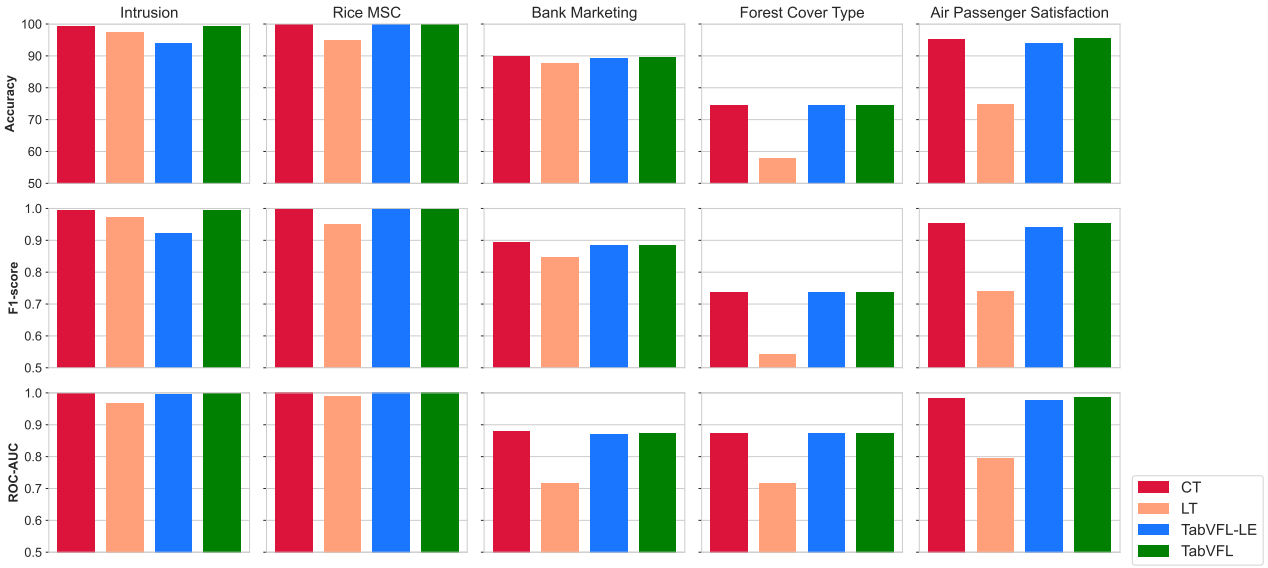


Fig. 5: Results of the latent representation quality of `TabVFL` and all baselines on different datasets.

VRAM and CUDA version 11.0. The amount of RAM is 32 GB. Intel(R) Core(TM) i9-10900KF CPU is used with speed of 3.70GHz and 20 cores.

Datasets. Five different classification datasets are considered for evaluations: Air Passenger Satisfaction (`air`), Network Intrusion Detection (`intrusion`), Bank Marketing (`bank`), Forest Cover Type (`forest`) and Rice MSC (`rice`). The datasets `bank` and `forest` are retrieved from UCI Machine Learning Repository, while `air`, `intrusion` and `rice` are acquired from Kaggle. Among the classification datasets, three are suited for binary tasks, while two are tailored for multi-class tasks. The metadata of each dataset are shown in Tab. II. All datasets, with the exception of `bank`, are down-sized to 50K samples due to the limited computational resources. Stratified random sampling is employed with regards to the target label to ensure identical class distribution between the original and the down-sized datasets. All the designs are run for 300 epochs to allow ample time for convergence both during pretraining and finetuning. Batch size of 64 is used. A latent dimension of five has been chosen for all experiments to facilitate similar model sizes across all baselines and to encourage a compressed latent representation for all datasets. Furthermore, three runs are conducted from which the average is calculated and presented to account for the inherent randomness and variability in the results. During the experiments, each dataset is split vertically and uniformly to mimic a VFL environment. Moreover, the numerical features of the datasets are preprocessed using standard scaling. The binary features and class labels are label encoded, while the categorical features are one-hot encoded. The standard scaling and label encoding are applied using sklearn [43].

B. Evaluation Pipeline

An evaluation pipeline has been devised to evaluate the effectiveness of the generated latent representation of each design. The pipeline is shown in Fig. 4. The full dataset is preprocessed and split into training, valid and test sets with a ratio of 70%-15%-15% respectively. The training set is used as input for both pretraining and finetuning in each experiment design. The valid set is used for the early stopping mechanism that we employ during training. A patience value is used as the maximum count of how many iteration to wait for the validation score to improve. The early stopping feature is implemented as a regularization method to prevent overfitting [44]. The training is stopped when the maximum amount of epochs is reached or when the patience value is reached. After training completion, the training, valid and test sets are combined and processed by the TabNet finetuning model to generate the total latent set from the encoder. The set is then split into a training and test set with ratios 70%-30%, respectively, using random shuffling to generate the latent train and test sets. The latent train set is used to train six classic machine learning algorithms: logistic regression, decision tree, random forest tree, multi-layer perceptron (MLP), XGBoost and linear support vector machine (Linear SVM). The performance, and hence the quality of the generated latent data, of each algorithm is evaluated on the latent test set using the conventional classification metrics: accuracy, F1-score, and the area under the curve of the receiver operating characteristic curve (ROC-AUC). For each experiment run, the mean of the metric results of each algorithm are reported.

TABLE II: Metadata of the datasets used for the experiments.

Dataset	#Samples	#Classes	#Categorical	#Numerical	#Binary
Intrusion	50,000	2	11	23	4
Rice MSC	50,000	5	0	106	0
Air Passenger Satisfaction	50,000	2	15	4	3
Forest Cover Type	50,000	7	44	10	0
Bank Marketing	45,211	2	6	7	3

C. Result Analysis

The experiment setup described thus far holds in any experiment unless stated otherwise. Our experiments are designed to answer the following three research questions:

- How does the *latent quality* produced by `TabVFL` compare to that of the baselines, and to what extent is it superior?
- How does the *robustness* of the cached and zero intermediate results differ in `TabVFL` against random client failures?
- How efficient is `TabVFL` in terms of runtime, network consumption and memory usage compared to the baselines?

The questions are answered by conducting three experiments. The first experiment is to evaluate the quality of the latent representation of each design using the classification metrics. To answer the second question, an experiment with different failure probability of each client is devised to compare the cache method for client failure handling with existing zero vector method in the literature. The amount of guest clients for the first and last experiments is fixed to five clients, aligning with established setting in previous research [31], [45]. The setup of the second experiment consists of one host and two guest clients to adhere to the common two-client setup in VFL [9], [32]. For the second experiment setup, each method runs for 120 epochs in total with batch size of 128. Early stopping mechanism is disabled in the second and third for fair comparison and measurements. For the third experiment, we set the batch size to 512 and amount of epochs for both training phases to 100.

1) ***Latent Quality Experiment****:* Evaluating a design's capability to learn high-quality latent representations offers valuable insights into how effective `TabVFL` is able to effectively capture underlying feature correlations compared to the baseline. The results linked to this experiment are presented in Fig. 5 as histograms. Generally, `CT` always outperforms the baseline designs as expected since it possesses all the data locally.

Furthermore, `TabVFL` clearly performs better than the prior work design (`LT`) for all the classification datasets. In terms of accuracy, f1-score and roc-auc `TabVFL` can perform up to 22.34%, 26.12% and 19.41% better than `LT` respectively. This suggests that `TabVFL` is able to capture the underlying feature correlations which leads to the generation of better latent representations of the features. A dataset where `LT` was able to perform well and approach evaluation results almost similar to other designs is the `intrusion` dataset. This could be attributed by the simplicity of the underlying patterns of the dataset which results in high performance for most of the tested designs. Although `TabVFL-LE` has lower accuracy and f1-score compared to `LT`, it still managed to be better in terms of ROC-AUC score. This means that `TabVFL-LE` is better at discriminating between negative and positive classes (in the multiclass sense) compared to `LT`.

In addition, `LT` shows the biggest discrepancies in performance for the `forest` and `air` datasets compared to `TabVFL`. A possible reason is that both datasets have a varying amounts of correlated features at different placements compared to other datasets. `LT` fails to capture general patterns in such datasets because local pretraining may result in guest clients learning entirely distinct latent representations, introducing potential biases. This could steer the finetuning model to converge at a suboptimal minima during training, leading to added noise to the latent space, therefore failing to capture common and relevant latent patterns of features scattered among clients.

2) ***Client Failures Experiment****:* In this study, we contrast the caching method for intermediate results (Sec. IV-E) with the zero intermediate results method from [19] in handling client failures. The zero method compensates for missing client results with zero values. Client failure probabilities ranged from 0.2 to 0.5 in 0.15 steps. We also ran a baseline test without client failure handling. Only the f1-score, illustrating client failure impact, is reported and can be seen in Fig. 6.

The dashed blue line represents the baseline without any client failures. The displayed f1-scores are eight-run averages for each probability (**p**) value. As the client failure probability **p** increases, the f1-score drops for both methods due to compensating missing results with outdated ones, resulting in suboptimal performance. Yet, cache method shows a more stable and less drastic decline than the zero method. Comparing to zero method, the cache method showed its best average and maximum improvements of 9.15% and 13.16%, respectively, on the `forest` dataset. In contrast, its worst performance was a 3.01% decline with the `air` dataset, whereas the zeros

TABLE III: Client failures experiment improvements of cache method compared to zeros method and maximum performance decrease of both method compared to baseline. All values are presented in percentages. Best results are highlighted in **bold**.

Dataset	Average Improvement	Maximum Improvement	Maximum Performance Decrease Zeros Method	Maximum Performance Decrease Cache Method
Intrusion	2.3	4.37	-4.78	**-0.22**
Rice MSC	3.91	9.49	-10.85	-0.32
Bank Marketing	1.51	2.04	**-2.71**	-0.65
Forest Cover Type	**9.15**	**13.16**	-17.59	-2.11
Air Passenger Satisfaction	6.31	12.73	-18.04	-3.01

TABLE IV: System performance evaluation result. The pretraining and finetuning runtimes on `rice` dataset are presented including the network consumption and memory usage.

Dataset	Pretraining	Finetuning	Total Training	Pretraining Network	Finetuning Network	Memory
CT	**151** s	**116** s	**267** s	-	-	**2.75** GB
LT	244 s	187 s	431 s	-	**0.7** MB	15.12 GB
TabVFL-LE	252 s	204 s	456 s	**14.7** MB	3.5 MB	14.73 GB
TabVFL	194 s	138 s	332 s	30.38 MB	14.84 MB	14.73 GB

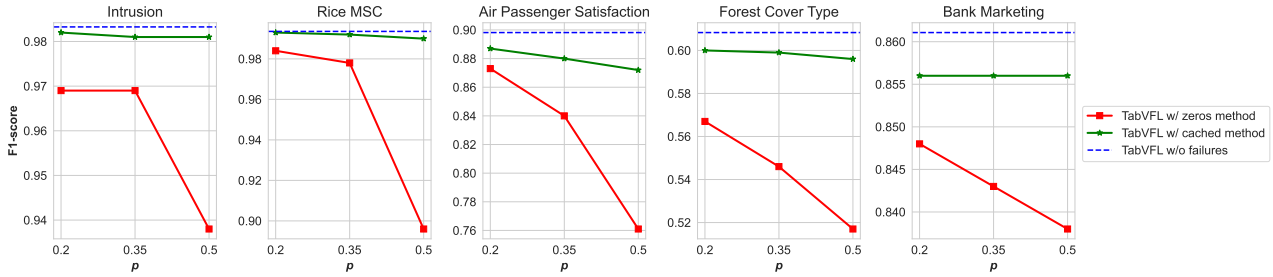


Fig. 6: Result over different client failure probabilities $\boldsymbol{p}$ on different datasets.

method dropped by 18.04% on the same dataset. The minimal performance decline was 0.22% for the cache method on `intrusion` dataset and 2.71% for the zeros method on `bank` dataset. The complete performance results can be found in Tab. III.

Remarkably, the cache method matched the baseline for the `rice` dataset at *p=0.2*, likely due to the dataset's simplicity. Furthermore, caching consistently outperformed the zero method, suggesting higher performance even with client dropouts, by using more realistic intermediate results instead of zeros. Zero vectors might introduce learning biases, producing non-existent patterns. The zero method's declining performance, especially with higher client failure probabilities, contrasts with the cache method's steadier results across datasets. By using the latest intermediate results, the cache method maintains some relevant data representations. However, as seen with the `air` dataset, performance might drop, influenced by the dataset complexity and the staleness of cached data.

*3) **Framework Efficiency and Resource Consumption Analysis**:* In this section, we measure and display key performance metrics for common VFL systems, including runtime, network consumption, and memory usage for the `TabVFL`, `LT`, and `TabVFL-LE`. For the centralized model, we exclude network consumption as it is not federated. For `LT`, as pretraining occurs locally at each guest client, we choose the client with the longest runtime as its representative. No communication means no pretraining network consumption for `LT`. The results are presented in Tab. IV. Training phase network consumption is gauged using a 4096 batch size for one iteration of each design.

We base our efficiency assessment on the `rice` dataset, which has the most data columns in our evaluations, making its results indicative. `TabVFL` achieves the fastest runtime, being 29.75% quicker than `LT` and 38.04% faster than `TabVFL-LE`. `TabVFL-LE`'s slower pace is due to its aggregation phase requiring iterative summation for each decision step. `LT`'s delay stems from the entire TabNet model pretraining at each guest client.

In terms of network consumption during pretraining, `TabVFL-LE` uses less than `TabVFL` since it transmits outputs (i.e., the latent representation) with a consistent, smaller dimension compared to `TabVFL`'s output dimension (i.e., the same dimension as input data). For finetuning, `LT` is most efficient as each guest uses only a fifth of the total latent dimension, reducing network consumption significantly.

Memory-wise, `CT` consumes the least, whereas `LT` uses the most. Although `TabVFL-LE` guest clients employ TabNet encoders, they have similar memory usage to `TabVFL` because `TabVFL`'s total input dimension equals the combined local input of all `TabVFL-LE` guest clients.

*4) **Summary**:* Across a variety of classification datasets, `TabVFL` outperforms both the prior `LT` design and the alternate `TabVFL-LE` design while maintaining nearly lossless performance in comparison to the central TabNet model (`CT`). `TabVFL` better captures the feature correlations among guest clients and produces higher-quality latent representations. The cache method shows a significant enhancement in `TabVFL`'s model robustness compared to the trivial zeros method at the expense of a higher memory utilization. Although `TabVFL` excels in terms of runtime and memory utilization compared to `LT` and `TabVFL-LE`, it incurs a higher but still practical network consumption from the extra roundtrips needed by its U-shape design.

VI. Conclusion

In this paper, we present a novel distributed framework `TabVFL` that integrates the state-of-the-art tabular model TabNet to improve latent representation learning on tabular data in the context of vertical federated learning (VFL). To protect against direct data leakage, `TabVFL` employs a fully-connected layer to preserve data privacy. This is due to the risk of direct data leakage from integrating TabNet as is in VFL. `TabVFL` consolidates intermediate results from all parties in order to learn a single latent respresentation, capturing underlying feature correlations. The framework also addresses the client failures by caching intermediate results. Comprehensive experiments across five datasets illustrate remarkable improvement up to 26.12% on f1-score in latent quality by `TabVFL`, surpassing baseline designs. `TabVFL` demonstrates superior performance in terms of runtime and memory usage, yet encounters reasonable communication overhead when compared to the baselines.

REFERENCES

[1] D. E. Rumelhart and J. L. McClelland, "Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations," 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:15291527

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:52967399

[3] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. rahman Mohamed, N. Jaitly, A. W. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, p. 82, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:7230302

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:195908774

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017.

[7] L. Ostroumova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Neural Information Processing Systems*, 2017.

[8] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," *ArXiv*, vol. abs/1908.07442, 2019.

[9] S. Yang, B. Ren, X. Zhou, and L. Liu, "Parallel distributed logistic regression for vertical federated learning without third-party coordinator," *ArXiv*, vol. abs/1911.09824, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:208248396

[10] L. Xia, P. Zheng, J. Li, W. Tang, and X. Zhang, "Privacy-preserving gradient boosting tree: Vertical federated learning for collaborative bearing fault diagnosis," *IET Collaborative Intelligent Manufacturing*, vol. 4, no. 3, pp. 208–219, 2022. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cim2.12057

[11] D. Cha, M. Sung, and Y. R. Park, "Implementing vertical federated learning using autoencoders: Practical application, generalizability, and utility study," *JMIR Medical Informatics*, vol. 9, 2021.

[12] K.-F. Chu and L. Zhang, "Privacy-preserving self-taught federated learning for heterogeneous data," *ArXiv*, vol. abs/2102.05883, 2021.

[13] A. Khan, M. ten Thij, and A. Wilbik, "Communication-efficient vertical federated learning," *Algorithms*, vol. 15, p. 273, 2022.

[14] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *ArXiv*, vol. abs/1912.12115, 2019.

[15] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *ArXiv*, vol. abs/1812.00564, 2018.

[16] F. Ang, L. Chen, N. Zhao, Y. Chen, W. Wang, and F. R. Yu, "Robust federated learning with noisy communication," *IEEE Transactions on Communications*, vol. 68, pp. 3452–3464, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:207870310

[17] Q. Chen, Z. Wang, Y. Zhou, J. Chen, D. Xiao, and X. Lin, "Cfl: Cluster federated learning in large-scale peer-to-peer networks," in *Information Security Conference*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250312010

[18] I. Ceballos, V. Sharma, E. Mugica, A. Singh, A. Roman, P. Vepakomma, and R. Raskar, "Splitnn-driven vertical partitioning," *ArXiv*, vol. abs/2008.04137, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221090598

[19] J. Sun, Z. Du, A. C. Dai, S. Baghersalimi, A. J. Amirshahi, D. Atienza, and Y. Chen, "Robust and ip-protecting vertical federated learning against unexpected quitting of parties," *ArXiv*, vol. abs/2303.18178, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257900954

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

[21] L. Lu and N. Ding, "Multi-party private set intersection in vertical federated learning," *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 707–714, 2020.

[22] N. Angelou, A. Benaissa, B. Cebere, W. Clark, A. J. Hall, M. A. Hoeh, D. Liu, P. Papadopoulos, R. Roehm, R. Sandmann, P. Schoppmann, and T. Titcombe, "Asymmetric private set intersection with applications to contact tracing and private vertical federated machine learning," *ArXiv*, vol. abs/2011.09350, 2020.

[23] Z. Mao, H. Li, Z. Huang, Y. Tian, P. Zhao, and Y. Li, "Full data-processing power load forecasting based on vertical federated learning," *J. Electr. Comput. Eng.*, vol. 2023, pp. 9 914 169:1–9 914 169:9, 2023.

[24] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 2090–2103, sep 2020. [Online]. Available: https://doi.org/10.14778/3407790.3407811

[25] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, pp. 87–98, 2019.

[26] Z. Zhao, H. Wu, A. van Moorsel, and L. Y. Chen, "Gtv: Generating tabular data via vertical federated learning," *ArXiv*, vol. abs/2302.01706, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:256598036

[27] 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," *Proceedings of the 2021 International Conference on Management of Data*, 2021.

[28] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, vol. 8, pp. 843–854, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:165163667

[29] H. R. Roth, A. Hatamizadeh, Z. Xu, C. Zhao, W. Li, A. Myronenko, and D. Xu, "Split-u-net: Preventing data leakage in split learning for collaborative multi-modal brain tumor segmentation," in *DeCaF/FAIR@MICCAI*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:251741520

[30] W. Li, Q. Xia, H. Cheng, K. Xue, and S. Xia, "Vertical semi-federated learning for efficient online advertising," *ArXiv*, vol. abs/2209.15635, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252668523

[31] Z. Zhao, H. Wu, A. Van Moorsel, and L. Y. Chen, "Gtv: Generating tabular data via vertical federated learning," *arXiv preprint arXiv:2302.01706*, 2023.

[32] 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*, vol. abs/2104.00489, 2021.

[33] Y. Hu, D. Niu, J. Yang, and S. Zhou, "Fdml: A collaborative machine learning framework for distributed features," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:156053162

[34] Z. Wu, Q. Li, and B. He, "Practical vertical federated learning with unsupervised representation learning," *ArXiv*, vol. abs/2208.10278, 2022.

[35] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," *ArXiv*, vol. abs/1902.01046, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:59599820

[36] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, and X. Liu, "Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data," *Proceedings of the Web Conference 2021*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235324741

[37] L. Li, H. Xiong, Z. Guo, J. Wang, and C. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 406–418, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:203582658

[38] C. Yang, Q. Wang, M. Xu, S. Wang, K. Bian, and X. Liu, "Heterogeneity-aware federated learning," *ArXiv*, vol. abs/2006.06983, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219636005

[39] J. Hou, M. Su, A. Fu, and Y. Yu, "Verifiable privacy-preserving scheme based on vertical federated random forest," *IEEE Internet of*

Things Journal, vol. 9, pp. 22 158–22 172, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:237905041

[40] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. xilinx Wang, L. Wang, J. Zhou, and B. Zhang, "Large-scale secure xgb for vertical federated learning," *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:237396281

[41] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "Blindfl: Vertical federated machine learning without peeking into your data," *Proceedings of the 2022 International Conference on Management of Data*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249578995

[42] S. Fischman, "Pytorch implementation of tabnet paper," https://github.com/dreamquark-ai/tabnet, 2019.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *NIPS*, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:7365231

[45] A. Li, H. Peng, L. Zhang, J. Huang, Q.-W. Guo, H. Yu, and Y. Liu, "Fedsdg-fs: Efficient and secure feature selection for vertical federated learning," *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pp. 1–10, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257050801

Chapter 3

Background

In this chapter, details are discussed about the general workings and training procedures employed by federated learning and the TabNet model. Mathematical formalizations and justifications are also included.

3.1 Federated Learning

Federated Learning (FL) was first introduced by Google in the year of 2016 [17] and attracted attention from ML practitioners and researchers [18, 19, 20]. Clients in the context of FL can refer to either personal devices (cross-device) or organizations with large datasets (cross-silo). The former is mainly applicable in Horizontal Federated Learning (HFL) where devices share common features but different subsets of the data, while the latter is mostly the case in the VFL setting. A depiction of how the data is partitioned in HFL is shown in Figure A.5. A key advantage of FL is that it allows for the training of models on a much larger dataset than would be possible if all the data was centrally located. This is because FL conveniently aggregates the data from multiple decentralized sources, which can collectively represent a larger and more diverse dataset without the need to move raw data to a central server. FL also has the potential of being more efficient than traditional centralized machine learning methods, as it allows for the distributed training of models on multiple devices, which can speed up the training process.

A client-server architecture is mainly considered for the distributed training of a ML model in FL [21]. Initially, a global model is initialized at the server and distributed to all or a subset of clients for training. The selected clients either train the received model using their local data or only calculate the gradients using their model. Afterwards, the partially optimized local model parameters or the gradients are sent to the server. The server then aggregates all the incoming client results using an aggregation algorithm. After aggregation, the global model parameters are updated accordingly and shared with all the participants. The aforementioned steps are iterated until the global model converges to a stable state (where the global model parameters do not change or do not change significantly anymore). Two of the most well-known aggregation methods in FL are FedAvg and FedSGD. In FedSGD, the gradient calculations of each client are transmitted to the server.

The server then updates its global model weights using Stochastic Gradient Descent (SGD) by averaging the gradients. For a neural network model, SGD is widely used as an iterative optimization algorithm for finding the minimum of a function. The formula applied by SGD for updating the model weights is as follows:

$$\theta_{i+1} = \theta_i - \alpha \nabla Loss(\theta_i) \tag{3.1}$$

Here, $\nabla Loss(\theta_i)$ indicates the gradients of the loss function with respect to the model weights θ at iteration i and α is the learning rate. In FedSGD, the average gradients of each client $c \in M$ over its local data are calculated by each client:

$$grad_c = \frac{1}{n_c} \nabla Loss(\theta_i^c) \tag{3.2}$$

The average gradients are sent to the server where the weighted average is taken and summed up for updating the global model as follows:

$$\theta_{i+1}^{global} = \theta_i^{global} - \alpha \sum_{c=1}^{M} \frac{n_c}{N} grad_c \tag{3.3}$$

Here, n_c means the amount of samples selected by client c and N is the total amount of samples of all clients combined. The fraction $\frac{n_c}{N}$ is used for averaging since the samples that are considered by a client c play a significant role in the effectiveness of the model training.

On the other hand, FedAvg is a natural extension of FedSGD that involves sending the whole local model parameters of each client to the server after conducting multiple local SGD updates:

$$\theta_{i+1}^{local} = \theta_i^{local} - \alpha \nabla Loss(\theta_i^{local}) \tag{3.4}$$

This makes FedAvg more efficient in terms of communication as the local model weights are only transmitted to the server after the clients are done with multiple local model iterations. The server then only needs to calculate the weighted average of the local models:

$$\theta_{i+1}^{global} = \sum_{c=1}^{M} \frac{n_c}{N} \theta_{i+1}^c \tag{3.5}$$

3.2 TabNet

An architecture that is somewhat similar to the autoencoder is TabNet [14]. TabNet is a popular DNN model that is specifically designed for tabular data. It utilizes a special attention mechanism to capture inter-dependencies of features and focus only on the important parts of the input data. This could help improve the extraction of contextual information and increase the prediction quality on tabular data. Similar to the autoencoder architecture, TabNet also has an encoder and a decoder but both components differ dramatically from the traditional autoencoder in terms of design and information flow. The usual workflow of TabNet consists of pretraining it on the input data using the encoder-decoder design and then finetune the model on a classification problem. First, the pretraining phase is explained.

During the first step through the encoder, the input data flows through a batch normalization layer. Afterwards, the normalized values are dispersed to a feature transformer and multiple sub-networks called decision steps. A feature transformer is responsible for further processing of the normalized features. It does this using fully-connected (FC) layers, batch normalization (BN) layers and gated linear units (GLU). GLU is an activation function that works like a filter for the processed values, allowing only some parts of the values through it. The combination of the three components is called a block and there can be a variable amount of them in the feature transformers. What makes these blocks so special is that they can be made either dependent or independent. Independent blocks are specific for each decision step and the dependent blocks are shared throughout the decision steps. The reason for this is to improve the overall model capacity and to create an efficient way of handling the model parameters. Furthermore, there exists a normalized residual skip connection between each block except the first one in a feature transformer. This approach is adopted to accomplish a stable model performance by preventing potential fluctuations in the variance of the model. The variance is an indication of how sensitive a model is to changes in the input data.

Before the feature transformer in each decision step, the information is processed by an attentive transformer. It consists of four layers: FC, BN, sparsemax and prior scales layer. The prior scales layer is a matrix that keeps track of the utilization of the features from previous steps. It also uses a relaxation term to indicate whether to use a particular feature at multiple decision steps or only at one. The input of the previous decision step first passes through the FC and BN layers, which is then multiplied with the prior scales. The result is then processed by the sparsemax layer by which the values are normalized using a sparse variation of the softmax algorithm. A sparse mask matrix is generated that is applied for soft instance-wise selection of the most prominent features. The reason behind the feature selection is efficiency as unnecessary computations on irrelevant features are avoided.

The outcome is proceeded to the feature transformer and then split into a part that determines the outcome of a decision step and a part that is sent to the next decision step for further processing. The former part is fed through a ReLU activation function and determines the contribution of the decision step towards the final decision. The latter part is fed into the attentive transformer of the next decision step.

Formally, The full operation that is conducted by the attentive transformer on a particular decision step is as follows:

$$attentive\_out = f * M \tag{3.6}$$

Here, f is the combined function of both FC and BN layers. M is the masked matrix resulting from the sparsemax operation on multiplication of the prior scale matrix P and the result of f on the split feature transformer output from the previous decision step:

$$M = sparsemax(P \cdot f(X_{split})) \tag{3.7}$$

P at a decision step is the difference of the relaxation term ρ with the mask matrices of the previous decision steps N:

$$P = \prod_{step=1}^{N} (\rho - M_{step}) \tag{3.8}$$

Then the attentive transformer output is processed by a feature transformer $feat$ of the current decision step and split into values that are sent towards the next decision step and values that are used for the decision making:

$$feat\_out = feat(attentive\_out) \tag{3.9}$$

$$out_{decision}, out_{step+1} = split(feat\_out) \tag{3.10}$$

The information is sequentially processed through all the decision steps. Finally, the decision outcomes are aggregated by means of summation to obtain the resulting decision which represents the latent representation of the input samples:

$$latent\_out = \sum_{step=1}^{total\_steps} ReLU(out_{decision}) \tag{3.11}$$

Moreover, the features are masked randomly using a Bernoulli distribution before entering the encoder during pretraining. A binary mask **S** is created with the same shape as the input data. The entries containing zeros represent masked feature values and entries with ones indicate the known features. The encoder is then trained on the features that are known by setting the prior scales in the attention transformers to (1 - **S**). The task of the decoder is then to reconstruct and thus predict the missing masked values from the encoded representations of the encoder. This helps in learning the feature representation of the input data before training on a classification task to lower the training time and enhance the prediction quality. The decoder has the same multi-decision step structure as the encoder but it only contains a feature transformer and a FC layer in each step. The output of each step is summed to reconstruct the masked feature values.

$$\hat{f} = \sum_{i=1}^{total\_steps} FC_i(feat_i(latent\_out)) \tag{3.12}$$

Where $\hat{f}$ constitutes the predicted masked feature values and $feat_i$ along with FC_i are the feature transformer and the fully connected layer function respectively at decision step i.

The specific loss function used by TabNet calculates the difference of the predicted and known feature values while taking the initial binary mask matrix into consideration. It additionally normalizes the feature predictions using the standard deviation of the known feature values to account for different feature value ranges:

$$\sum_{b=1}^{B}\sum_{j=1}^{D}\left|\frac{(\hat{f}_{b,j} - f_{b,j}) \cdot S_{b,j}}{\sqrt{\sum_{b=1}^{B}(f_{b,j} - 1/B\sum_{b=1}^{B} f_{b,j})^2}}\right|^2 \tag{3.13}$$

$\hat{f}_{b,j}$ stands for the reconstructed feature j of sample b. $f_{b,j}$ is the true feature value j of sample b. S represents a the binary mask matrix. B is the batch size.

During finetuning, only the encoder is used in combination with a FC layer for predictions and to further finetune the model as follows:

$$\hat{y} = FC_{pred}(latent\_out) \tag{3.14}$$

The default classification loss used by TabNet is cross entropy on the normalized predicted vector $\hat{y}$ of a classification task with K classes:

$$\mathcal{L}_{cross} = -\sum_{c}^{K} y_c log(\hat{y}_c) \tag{3.15}$$

A sparsity regularization term based on entropy (measure of disorder in the sparsemax mask matrix) is added to the classification loss:

$$\mathcal{L}_{sparse} = \sum_{i=1}^{total\_steps} \sum_{b=1}^{B} \sum_{d=1}^{D} \frac{-M^i_{b,d} log(M^i_{b,d} + \varepsilon)}{total\_steps \cdot B} \tag{3.16}$$

The total finetuning loss is evaluated using the following terms:

$$\mathcal{L}_{total} = \mathcal{L}_{cross} + \mathcal{L}_{sparse} * \lambda_{sparse} \tag{3.17}$$

The sparsity regularization term offers more control over the feature selection of the sparsemax function. λ_{sparse} can be tuned to either encourage or discourage sparsity.

The attention mechanisms in TabNet also provide a built-in approach of deducing feature importance from the generated masks at each step. However, feature interpretability is beyond the scope of this thesis and hence not considered.

Chapter 4

Extended Related Work

This chapter presents an extended overview of frameworks that focus on prediction models related to VFL. The existing client failure handling methods are also covered.

4.1 Prediction models in VFL

	Trains One Latent Representation	Client Failure Handling	Tabular Model	Imposes Autoencoder Structure
CE-AE-VFL [13]	✗	✗	✗	✓
DIMIP [16]	✗	✓	✗	✗
FedOnce [22]	✗	✗	✗	✗
AE-VFL [11]	✗	✗	✗	✓
TabVFL	✓	✓	✓	✓

Table 4.1: Table showing differences in framework features between relevant related work in VFL and current work (TabVFL).

4.1.1 Linear models

Until now, frameworks for VFL are quite limited compared to other FL types. This is due to design challenges and specific settings that should be adhered to. Nonetheless, multiple frameworks have been implemented to realise the learning process of ML models in a VFL setting.

There have been efforts to utilize the logistic regression model in a VFL context [23]. Within the framework, an architecture is employed featuring a coordinator and two parties. The coordinator can be used as a reliable entity with the main responsibility of providing a secure communication of intermediate results between parties and for creating the encryption keys. In addition to the encryption, a random mask is added to the gradient information in order to prevent one of the parties knowing the the true gradient values which can lead to a potential privacy breach. The coordinator can be a separate node or a logical unit in one of the parties.

There also exists a simplified VFL design based on logistic regression that omits the coordinator and utilizes peer-to-peer communication instead [24]. In this case, the host is

responsible for a secure communication between it and the guest client. Only the loss values of host and the prediction probabilities of the guest are exposed. This is done in an effort to reduce the risk of data leakage by minimizing the number of parties involved in the system and decreasing the number of communication round-trips. Moreover, this measure also aims to address possible trust issues that may occur between the parties and the coordinator. The VFL architecture is also applied for other classical ML models such as SVM [25] and Linear Regression [26] to name a few.

4.1.2 Tree-based models

The linear ML models cannot handle large and complex data with non-linear patterns well enough which is quite common in real-world tabular datasets. Tree-based models have shown state-of-the-art results in many ML competitions [27, 28, 29]. For this reason, training tree-based models in order to exploit their predictive power while maintaining data privacy is of great importance for businesses in VFL.

Work has been done to integrate the decision tree model into VFL [30]. This is done using the well-known algorithm ID3 [31] for building decision trees for classification tasks. ID3 is a recursive process where the dataset is split into subsets based on the most significant feature(s) at each node of the tree. The splitting occurs by calculating the entropy of before and after a particular split. The best split is considered the one with the highest information gain or the lowest entropy value. The framework also extends the two-party setting by supporting multiple parties.

A random forest has also been implemented into the VFL context [32]. It uses a third-party server for encrypting the communicated values securely. Each party trains only partial tree while only one party is able to build the complete tree. All parties have access only to the encrypted labels. The usual bagging method is used to randomly choose subsets of features to process for splitting the trees. The framework has been shown to achieve comparable results to the non-federated setting.

SecureBoost [33] is able to train a tree-based boosting model, such as XGBoost, without sacrificing model utility. It is shown that SecureBoost performs similar to the non-federated setup of XGBoost and other GBDT models. Unlike the ID3 decision tree implementation in VFL, it can handle both discrete and continuous data by using Gini index for splitting. However, similar to the implementations of ID3 decision trees and random forest in VFL, the communicated results have the potential to leak sensitive information about the participants' data. The data distribution statistics of the guest clients can be revealed to the host by sharing the split information unencrypted. Moreover, the host is given the coordinator role for the training process which exposes prediction paths of the tree, leading to potential inference attacks on the feature values of the guests.

Pivot [34] is a framework that is built to handle ensemble models and classic tree-based models. Its main contribution is the enhanced communication protocol that mitigates the leakage of the transmitted results. It also protects against label leakage in cases where clients collude and work together to infer the leaf node paths of other parties. The label

and feature information of all participants are hidden by encryption. The encryption is achieved by using a special form of Homomorphic Encryption (HE) [35], allowing limited computations on the encrypted data without incurring high communication overhead. The splits and leaf node information of the trees are also encrypted to protect the tree models. This prevents a party from knowing the full decision tree paths and leaf node values during communication. The framework also has the ability to utilize another cryptographic technique called Secure Multi-Party Computation (MPC) [36], which enables multiple parties to collaboratively compute a function over their inputs while keeping those inputs private. The downside of using MPC is that it is computationally inefficient compared to HE. For this reason, Pivot supports both HE and MPC to enhance security and improve efficiency. More specifically, HE is mainly used for local calculations on encrypted data, while MPC is used during joint operations, e.g., deciding the best split of the tree. This prevents the involved parties from deducing and reconstructing sensitive information from the shared calculations.

The mentioned frameworks are not efficient due to the sequential nature of training the trees. VF^{2}Boost [37] is another gradient boosting framework that focuses mainly on improving efficiency and speedup of the training process. The framework makes use of a scheduler-worker design to enable concurrent computations and synchronizations throughout all the parties. Moreover, further optimizations are implemented that specifically speed up the cryptographically-secured communication of the computations. These optimizations entail the expedition in histogram construction through the grouping of multiple histograms into one and the re-ordering of the gradient accumulations in the histogram bins. The latter is done to minimize scaling operations in the HE calculations.

4.1.3 Deep learning models

With the increasing popularity of DNNs, most of the recent work in VFL focuses on applying neural networks into VFL. To achieve this, splitNN architecture is mainly applied. In [38], HE is applied for protecting the transmitted value during the forward propagation step of the training. However, due to the iterative training process and matrix operations, most cryptographical encryption techniques are rarely considered because of the high computational cost and large communication overhead [39, 40, 41]. Hence, intermediate results are transmitted in plaintext in most of the related papers. Still, splitNN is privacy preserving as only the transformed intermediate results are sent by passing the raw data through the local part of the total network. A neural network model can be split in different shapes ([42, 43]) which makes it adaptable to the required model structure.

In [43], different aggregation methods in splitNN are experimented with to provide a valuable insight into their effectiveness and impact on performance. The authors tested the following aggregation techniques on intermediate results: summation, average pooling, multiplication, max pooling and concatenation. Max pooling is found to yield the highest performance where, in some cases, it could outperform a centralized model with a slight margin on some datasets. However, this could be caused by insufficient re-runs of the

experiments which causes variability in the evaluation results influenced by random factors. It is argued that the use of average pooling method is preferable in order to ensure a secure aggregation protocol at the expense of a small performance hit [44].

There have also been efforts to alleviate the communication overhead present in VFL. In [13], separate autoencoders are utilized for training. Each guest possesses an autoencoder model which is trained on the local feature data. Each autoencoder learns a compressed latent representation of the local feature data for feature extraction. Afterwards, only the latent representation of the samples are sent to the host client for training a prediction model, e.g., logistic regression. The latent dimension is varied using different compression rates. A well-known feature extraction technique called Principal Component Analysis (PCA) is also considered. PCA works by reducing the dimensionality of the original data through the projection of the data onto new linear and uncorrelated axes (principal components). For most of the tested datasets, both the autoencoder and PCA compression methods lead to decreased performance compared to the centralized model. The autoencoder results show that compressing the data leads to worsened performance when the compression is increased. This is due to the loss of relevant patterns in the data. Also, the data privacy is preserved since only the compressed representation of the raw data is transmitted.

In [11], the authors impose the training of multiple autoencoders with higher latent dimension compared to the input dimension of features. It is claimed that the larger latent dimension would decrease the information loss compared to using a compressed latent dimension while ensuring privacy of the raw data. Notably, higher computation costs are required because of the higher latent dimension. Remarkably, performance degradation is also noticeable like in [13] from the added noise to the high-dimensional latent representation. It is mentioned that a tabular neural network is used for predictions and for evaluation, but it is not indicated what tabular model is used. This makes the method with which the evaluation results are gathered ambiguous.

Adapting the Variational Autoencoder (VAE) model into VFL is also considered [12]. The goal is to learn a latent representation that resembles the Gaussian distribution. A gaussian distribution is a popular probability distribution that is centered around the mean of the data (where most of the data points lie). The same autoencoder architecture in VFL is used, but now the latent representations are generated by random sampling from the gaussian distribution. The forward propagation results are encrypted while the gradients are also masked to prevent reverse engineering attempts.

FedOnce framework [22] enforces one communication round between the guests and host clients using a splitNN design. The guest clients first train their own partial neural network using an unsupervised technique called Noise As Targets. The technique is generic and different from autoencoders as it does not need partial supervision from reconstructing features to be able to learn meaningful representations. The host only needs to collect the representations from all the guests once to train a combined model using its own label information and local feature data. This prevents repeated communications of intermediate results in each iteration, meaning that only one communication round is required to send the learnt local representations in the process. Furthermore, an extension to differential privacy (DP) [45], a method for protecting data privacy by the addition of noise, is proposed that is able to reduce the privacy loss between parties.

BlindFL [46] is another prediction framework in VFL that enhances the splitNN design with encryption techniques in a two-party setting, preserving data and model privacy. It extends the splitNN architecture by introducing source layers, which combine the data of participants for proper feature processing of categorical, numerical and sparse feature values. The source layers also provide computational privacy by employing HE and MPC methods to preserve data privacy. HE is applied to enable local calculations on encrypted data. MPC is used to protect the model gradients and weights by ensuring that no party can infer valuable information about the full model. BlindFL achieves remarkable performance with notable privacy guarantees at the cost of communication and computational overhead.

4.2 Client Failure Handling in VFL

Most of the VFL research papers assume that clients stay online during training and inference. However, this assumption does not always hold in practice due to heterogeneous systems and network. Nonetheless, only a few papers have considered researching the impacts of client failures in VFL [47, 48, 16]. In this section, the words ”client failure” and ”client dropout” are used interchangeably.

In [47], multiple host clients are supported and client failures that could occur during training of a tree-based random forest model are handled through dynamic joins and dropouts mechanisms. The federator efficiently modifies client-specific statistics by tracking joined and dropped clients. The calculations specific to the dropped clients are disregarded and the index data is updated by the server. This leads to less computations with the expense of fewer trees being generated, possibly degrading the predictive power. In cases where a host client is dropped, the federator executes additional synchronization steps by also updating the label indices. Entity alignment is also executed in cases of multiple participating host clients. The evaluation results of the dynamic joins and dropouts feature show a stable accuracy trajectory in situations where high amounts of clients are dynamically changed between ”joined” and ”dropped” states.

The impact of client failures in a splitNN setting within the VFL context has been studied for the first time in [48]. The methodology consists of dropping clients in a random fashion during training and inference time. The setup entails four clients. The outcomes of the experiments have shown that client failures can negatively impact the performance of a DNN model in VFL. This effect is exacerbated when the failed clients increase. Though, the performance drop is more noticeable during inference phase than during training. Remarkably, client failures also impair the convergence rate of the model leading to longer runtimes and inefficient training. Furthermore, different aggregation methods have been researched in cases of client failures. The average pooling method, which downsamples the batch of samples using averaging, is shown to be the most robust during inference. However, no clear or significant differences in evaluation results could be identified during training between the tested methods.

In [16], the negative effects of client failures in a splitNN setup are addressed. The authors claim that the performance degradation of the VFL model is due to the dependability of the host client on the intermediate representations of the guest clients. A method is

proposed that employs a dropout layer to the local DNN model of the host client. During training, the host then naturally mimics the failing of a client by replacing some representations from the guest clients with zeros while preserving model performance. This way, the host client becomes more independent and robust against client failures during inference. The experiments are conducted in a two client setup, with one of the clients acting as the host. The results show that adapting the proposed dropout method could significantly improve the accuracy of the model by more than 7% without the contribution of a failed client, enhancing the model robustness.

Chapter 5

Privacy Analysis Of The Proposed Framework

In this section, the privacy of `TabVFL` is analysed and the proposed solution is further elaborated while adhering to the threat model described in the research paper (chapter 2). Further privacy discussion is given at the end by involving privacy related splitNN work in VFL.

5.1 BatchNorm Direct Data Leakage

Due to the sequential dependencies in the TabNet model, splitting it to make it applicable in the VFL context was not trivial. Assigning the BN layer to each guest client given the split of the encoder is not sufficient to preserve the data privacy of the guest clients. The data leakage from the guest-assigned BN layer of the original encoder is possible in specific cases during forward propagation. To show how that is possible, the following BN formula is analysed:

$$\text{BatchNorm}(x) = \frac{x - E[x]}{\sqrt{Var[x] + \varepsilon}} \cdot \gamma + \beta \tag{5.1}$$

Here, $E[x]$ represents the mean and $Var[x]$ represents the variance of some batch x. The mean and variance are estimated using running averages to obtain consistent inference behaviour. The standardization occurs feature-wise. Furthermore, each feature is scaled by the corresponding γ factor and shifted using the corresponding β value. Both γ and β are initially set to a vector of one and a vector of zero respectively. Both vectors are learned and updated throughout the training process.

The data leakage risk is much higher at the start of the training as γ and β do not contribute towards the batch normalization. Moreover, it is also possible that later during the training, the values of γ and β are not significantly updated which could also lead to zero contribution towards the BatchNorm calculation. This allows for a data leakage problem in cases where the data is already standardized which could be from preprocessing or that (a subset of) the features happen to follow a standard normal distribution. Following the equa-

tion, the x representing the feature information in batches remains unaltered, effectively causing BatchNorm to behave as an Identity function, wherein the output equals the input. Hence, the original local data of guest clients is leaked to the host client.

5.2 FC layer Inclusion

To prevent the mentioned data leakage, an effective solution would be to add a trainable FC layer after the BatchNorm layer in each guest client. The FC layer is able to transform the BatchNorm using the following matrix multiplication:

$$Wx^T = z \tag{5.2}$$

Here, W is the weight matrix of the FC layer, x is the input batch of samples and z is the transformed output. The input and output dimension of the FC layer is set to be exactly the same as the original raw feature data to mitigate any information loss or possible noise addition in the embedding. This is also mandatory from the fact that the encoder in the host client expects the binary mask dimension to be the same as the intermediate results, which prohibits the possibility for choosing a different output dimension of the FC layer. The initialization of weights is done using Xavier normal distribution [49] and no bias terms are included to be consistent with the initialization of TabNet components. During the training phases, a transformed representation of the original feature data is sent to the host from the learnt embedding of the FC layer. Therefore, the BatchNorm cannot leak any data outside the guest clients.

5.3 Further Privacy Discussion

The finetuning structure is known to be susceptible to label inference attacks from exposing gradients to guest clients [50, 51]. [50] assumes binary classification task for label inference which is not always practical. Also, one of the proposed attacks utilizes a model to infer labels. Although it is difficult to defend against model-based label inference attacks, they heavily depend on the availability of auxiliary data. The utilization of an autoencoder to hinder the efficacy of model-based label inference attacks in particular cases has shown to be possible [52]. In [51], colluding parties are assumed for inferring labels which is not applicable given our threat model. Possible solution is to add Gaussian noise [53] to the communicated gradients in order to complicate the label inference process for attackers. Correlation reduction methods also exist for defending against label inference attacks by decreasing the correlation between the intermediate embeddings from guest clients and the label information in the host [54].

Reconstruction of the feature data corresponding to guest clients is also found to be possible in strict settings [55, 56], imposing a serious threat to the data privacy. In [55], the authors assume that the model parameters are known by the server which is not possible in our case according to the threat model. [56] utilizes a generative model and auxiliary data as a model inversion technique for inferring the feature data of guest clients. However, the authors assume that the adversary already has a collection of predictions and access to

the model parameters. [57] focuses on recovering features from intermediate results sent to the host client by assuming that the guest clients possess binary features. This is done by solving inverse equation of the linear transformation of a FC layer. The authors also claim that it is not possible to recover the data when nothing is known about feature data of guest clients since there are infinitely many solutions. Reducing the correlation between the raw data and intermediate embeddings can be a viable solution for minimizing the risk of feature inference as well [58]. However, inferring feature information is difficult to achieve in practice since the models are typically black box to the adversary and strong assumptions about the features need to be made.

Since the finetuning structure of `TabVFL` is not the main focus of this study, we do not apply the mentioned solutions. Also, incorporating the defence methods would unfairly and significantly effect the comparison of latent quality with the prior work due to the utility-privacy trade-off [59]. All the papers about privacy leakage attacks in VFL presume that label information is available to one of the parties. To the best of our knowledge, no attack exists that considers the proposed pretraining setting of `TabVFL` where label information is absent.

Chapter 6

Additional Experiments

This section goes into the details of the additional experiments conducted on TabVFL and the baseline designs. More specifically, convergence rates of all designs are measured and compared against TabVFL. Moreover, ablation studies are conducted on TabVFL to acquire better insight into how its components impact the performance. Lastly, the latent quality experiments are also conducted in the conventional two party setting to show how that affects the model performance in each design. To recap, CT represents the centralized TabNet model without federated learning, LT is the prior work design in VFL and TabVFL-LE is the alternative design of TabVFL. For further details about the baselines, refer to the research paper chapter 2. Furthermore, the designs of TabVFL-LE and LT are shown in Figure A.3 and Figure A.4 respectively.

6.1 Convergence Analysis

For this experiment we report the training loss and f1-score validation metrics during fine-tuning of all designs. For this experiment, no early stopping is used in order to show the full trajectory of the evaluation values. In Figure 6.3, the results of the convergence of each design are shown over the amount of epochs. The training loss progression as well as the validation metric scores are plotted for each dataset.

Both CT and TabVFL generally converge to the same point in terms of training loss and validation metrics. This is expected as the latent quality results of both models are overly consistent. For most of the datasets, TabVFL also exhibits a convergence rate that closely aligns with CT, indicating nearly identical efficiency.

A notable observation can be made for the validation metrics of the intrusion and bank datasets. TabVFL-LE shows to be noisy and behaves erratically instead of smoothly converging like in other designs. This behaviour could be attributed by overfitting. The structure for training TabVFL-LE with summation as aggregation method might be too complex for or sensitive to the relatively simple intrusion and bank datasets which leads to the model capturing the noise in the data instead of relevant patterns. Overall, TabVFL-LE seems to have the highest convergence rate compared to other designs. A reason for this could be that the feature correlations are captured per decision step by aggre-

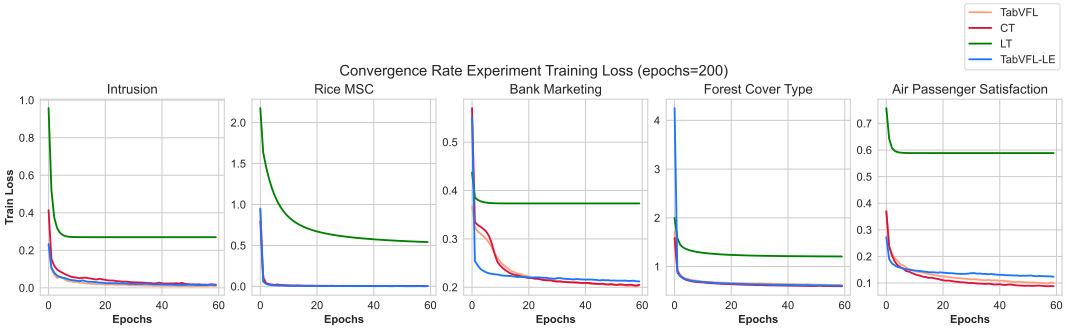


Figure 6.1: Finetuning training loss plots.

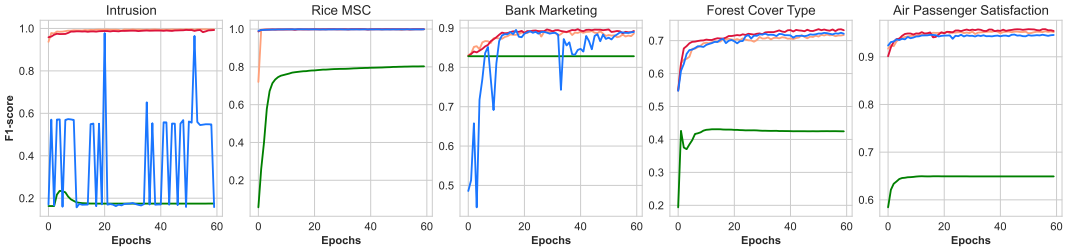


Figure 6.2: Finetuning validation f1-score plots.

Figure 6.3: Finetuning convergence plots show trajectory of training loss and f1-score validation metric over 60 epochs for each design on different datasets.

gating each guests' encoder decision step outputs. Since each decision step selects different features for processing, an optimal set of feature correlations can be found faster during pretraining. This facilitates the finetuning process leading to faster convergence rate. This can be seen by the fact that the training loss of the model in all datasets decreases and converges to an optimal solution faster than other designs. The plots also reveal that `TabVFL-LE` manages to converge at the same point in terms of training loss as `TabVFL` and `CT` for the datasets `forest`, `rice` and `intrusion`. This indicates that the data patterns in the aforementioned datasets are simple to capture than other datasets, leading to identical convergence points. The overfitting and the suboptimal convergence on most datasets could also be caused by the model complexity of `TabVFL-LE`.

One can also observe that `LT` performs the worst compared to other designs. This is evident by the higher training loss and the low validation scores when the model converges. In addition, `LT` has the slowest convergence rate for most of the datasets. During pretraining step, each guest client learns a separate latent representation of its own local feature data. When the models are collaboratively finetuned, it becomes considerably difficult to find one overarching latent representation using multiple distinct local models which leads to suboptimal convergence.

As expected, `TabVFL` and other designs never surpass the baseline model `CT` in terms of training loss and validation scores. To conclude, `TabVFL-LE` is shown to achieve the

Table 6.1: Evaluation results of the ablation studies on TabVFL. The presented values are averages of the six predictors. The best results of each evaluation metric are highlighted in **bold** per dataset.

Dataset	TabVFL								
	baseline			w/o pretraining			w/o finetuning		
	Accuracy (%)	F1-score	ROC-AUC	Accuracy (%)	F1-score	ROC-AUC	Accuracy (%)	F1-score	ROC-AUC
Intrusion	**98.23**	**0.982**	**0.991**	98.23	0.982	0.991	87.14	0.848	0.898
Rice MSC	**99.79**	**0.998**	**1.0**	99.79	0.998	1.0	86.13	0.859	0.962
Bank Marketing	**89.14**	**0.881**	**0.863**	88.52	0.869	0.817	86.82	0.826	0.596
Forest Cover Type	**72.04**	**0.712**	**0.852**	71.02	0.702	0.842	50.76	0.441	0.627
Air Passenger Satisfaction	**93.85**	**0.938**	**0.976**	92.45	0.924	0.968	59.82	0.583	0.621

highest convergence rate compared to TabVFL and the other models but it tends to exhibit symptoms of overfitting due to its complexity. In terms of convergence, TabVFL demonstrates stable convergence while consistently achieving high validation scores and has either consistent or lower training loss for most of the datasets compared to other designs (excluding CT).

6.2 Ablation Studies

For this experiment, different components applied in the TabVFL framework are investigated to show their impact on the model performance. The components to be investigated are (1) **finetuning** and (2) **pretraining**. The baseline is chosen to be the standard configuration of TabVFL with pretraining and finetuning enabled. In the case where pretraining is disabled, TabVFL is randomly initialized for finetuning without inheriting the weights of the pretrained model. On the other hand, when finetuning is disabled, the model is pretrained and the latent of only the pretrained model is used for evaluation. The average evaluation results of the six predictors are reported. The results of the ablation study are shown in Table 6.1. Important observation can be made regarding the significance of the finetuning component in terms of performance. The biggest drop in performance for most datasets is caused by removing finetuning from the training process. For both datasets intrusion and rice, the finetuning performance is consistent with the baseline results. This is most likely attributed by their simplicity. The results demonstrate that finetuning is essential in TabVFL for specializing its latent representations to a certain task for improved performance. Moreover, removing pretraining does not lead to performance drops as severe as when finetuning is removed. During finetuning, the model still has task-based knowledge and can compensate for the non-existent general feature representation from pretraining. In contrast, when the model only relies on pretraining, the generated latent representation could be scattered and not adapted to the task at hand which leads to low quality latent data and hence low performance. The ablation study accentuated the greater importance of the finetuning component in contrast to the pretraining component concerning the enhancement of latent representation quality.

6.3 Latent Quality Evaluation In Two Guest Client Setup

Until now, we evaluated the latent quality of `TabVFL` and other baseline designs in VFL using a five client setup. However, most of the VFL frameworks experiment with two client setup [60, 61, 24]. To adhere to this trend, we present additional latent quality experiments in a two client setup. A batch size of 64 is used with max epochs of 300. To be consistent with the setup of the research paper, we also apply early stopping and keep latent dimension value set to five for all the designs. The evaluation pipeline introduced in the research paper is also followed here. The results of this experiment are shown in Figure 6.4. The increase/decrease in terms of accuracy, F1-score and ROC-AUC in the two client setup compared to the five client setup are shown in the tables Table 6.2, Table 6.3 and Table 6.4 respecively. In this section, the results are analysed and justified.

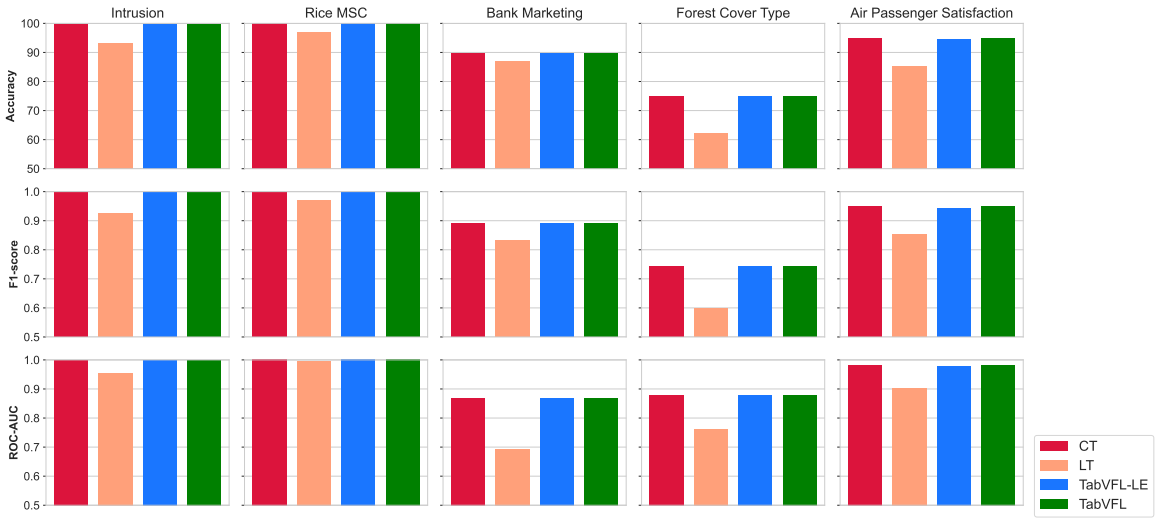


Figure 6.4: Latent quality results of `TabVFL` and other baselines on five classification datasets in two guest client setup. The average accuracy, F1-score and ROC-AUC scores of the six ML predictors used in the evaluation pipeline are reported for each design.

In general, `LT` shows slight improvement in latent quality results for all the metrics compared to the five client setup presented in the research paper chapter 2. It is more likely that correlated features end up in the same guest client which improves the capturing of those correlations, leading to enhanced latent representation quality. This is however also dependent on the quality of the datasets. A decline in accuracy, F1-score and ROC-AUC can be observed for `LT` on the `intrusion` and `bank` datasets. This can occur due to the inherent noisy features in the mentioned datasets which can be captured with ease when the latent dimension is increased, risking overfitting and degraded latent representation quality. In cases of two guest clients and a latent dimension of five, the latent dimension of guest one and guest two are assigned as three and two respectively. This is higher than latent dimension of one that was assigned for each guest in the five clients setup. Even though

the performance of LT increases on the majority of the datasets, it still results in inferior performance compared to other designs.

Furthermore, performance increase and decrease for TabVFL-LE is insignificant on most of the datasets. However, the design performance increases significantly on the intrusion dataset. This phenomenon could be attributed by the decrease in model complexity from the fact that only two encoders need to be trained with the same latent dimension distribution as LT compared to five encoders in the five clients setup. TabVFL-LE was able to better capture the feature correlations and learn an improved latent representation regardless of the noisy features. This indicates that the way TabVFL-LE trains is more robust than LT. The noise could be lessened or eliminated by taking each encoder decision step into account and aggregating them, improving the latent quality in the process. Notably, TabVFL-LE shows consistent performance that is on par with TabVFL and CT.

Lastly, TabVFL did not show any significant performance differences in the two clients setup. The latent quality is barely affected and the differences are negligibly minuscule due to the aggregation of intermediate results and the learning of one latent representation. The reason is that the correlations between client features are captured regardless of where they are located.

In conclusion, it has been shown that the latent quality results of TabVFL are stable and consistent with the five client setup. The latent quality is improved on one dataset in the case of TabVFL-LE, resulting in overall consistent performance with TabVFL and CT. The prior work design LT shows significant improvement on most of the datasets due to the enhanced correlation capturing from improved feature localizations.

Table 6.2: **Accuracy** increase/decrease of the latent quality in case of two guest client setup compared to the five guest client setup. The reported values are in percentages.

Dataset	**Accuracy**		
	LT	TabVFL-LE	TabVFL
Intrusion	-4.19	5.90	0.26
Rice MSC	2.13	-0.05	-0.05
Bank Marketing	-0.75	0.28	0.16
Forest Cover Type	7.49	0.89	0.88
Air Passenger Satisfaction	13.74	0.33	-0.53

Table 6.3: **F1-score** increase/decrease of the latent quality in case of two guest client setup compared to the five guest client setup. The reported values are in percentages.

Dataset	**F1-score**		
	LT	TabVFL-LE	TabVFL
Intrusion	-4.89	8.26	0.30
Rice MSC	2.18	-0.10	-0.10
Bank Marketing	-1.47	0.68	0.45
Forest Cover Type	9.92	0.95	0.95
Air Passenger Satisfaction	14.98	0.32	-0.52

Table 6.4: **ROC-AUC** increase/decrease of the latent quality in case of two guest client setup compared to the five guest client setup. The reported values are in percentages.

Dataset	**ROC-AUC**		
	LT	TabVFL-LE	TabVFL
Intrusion	-1.19	0.30	0.20
Rice MSC	0.58	0	0
Bank Marketing	-3.30	0	-0.57
Forest Cover Type	6.04	0.46	0.46
Air Passenger Satisfaction	13.67	0.20	-0.30

Chapter 7

Conclusions and Future Work

In this chapter, the conclusive remarks are given by revisiting and answering the formulated research questions using the gathered results and properties of the proposed framework. Furthermore, the future work ideas are discussed.

7.1 Conclusions

1. What design can be employed to effectively capture feature correlations and improve latent representation learning while leveraging TabNet in VFL?

`TabVFL` successfully integrates TabNet into the VFL context by splitting the encoder and decoder among the host and guest clients using a U-shaped splitNN design. This structure allows the intermediate results of all participating parties to be combined in order to train one model and capture the overall correlation between tabular features. Furthermore, one latent representation is learned to extract relevant feature patterns among the parties instead of isolated latent latent representations as in prior work.

2. How is the data privacy preserved in the `TabVFL` design?

From the privacy analysis discussed in chapter 5, it is made evident that the batch normalization (BN) layer that is assigned to each guest client causes a serious risk for a direct data leakage. This could happen in cases where the features are already standardized, which renders the BN layer useless. This privacy issue is resolved by adding a fully-connected (FC) layer after the BN layer. The FC-layer is able to transform the outputs of the BN layer in order to prevent it from leaking data directly.

3. How can the disruptive effect of client failures during training be reduced in `TabVFL`?

The existing literature has proposed a method for addressing missing values of failed clients by replacing them with zero values as a form of compensation. However, this method adds

a substantial bias to the learning process which deteriorates the model performance and the learned latent representation. With the aim of minimizing the bias issue, we proposed a caching method that continuously stores intermediate results of guest clients in an iterative basis. When a guest client goes offline, its corresponding cached results are utilized to maintain the highest level of correlation between the processed intermediate results. It was empirically shown that the proposed method significantly stabilized the model performance compared to the prior method, even in cases where it was highly probable that clients could fail.

The development of `TabVFL` sets forth a considerable step forward in designing VFL compatible latent representation learning systems for tabular data, offering elevated performance and stability in real-world applications.

7.2 Future work

Two configuration have been considered for implementing TabNet into VFL, namely `TabVFL` and `TabVFL-LE`. However, TabNet could take on another design in VFL. The whole decoder component could be held locally by each guest client, while the encoder is split among clients as in `TabVFL`. This would minimize the risk for reconstruction at the host site as it has no knowledge about the decoder weights. However, this design would incur higher communication overhead during pretraining compared to `TabVFL` and `TabVFL-LE`. Each encoder decision split output is sent to each guest for the decoder to process which is larger than the split decoder output in `TabVFL`. In addition, the total dimension of the combined intermediate results of the guest clients is larger than the latent dimension that is sent to the host in `TabVFL-LE`. Regardless of the expected communication overhead, the design might perform well in terms of latent quality which is to be verified through experimentation.

Furthermore, diverse decoder splits in `TabVFL` can be explored instead of uniform splits. How different decoder splits affect the performance of the proposed design is still unknown, offering an opportunity for extended experimentations. If the assigned split dimension for a guest client surpasses its input dimension, then the expected outcome is an improvement in performance on complex datasets due to the enlarged model capacity. Though, this could lead to worse performance due to higher risk of overfitting on inherently simplistic datasets. On the other hand, if a small split dimension is assigned to the guest client, the model capacity decreases which might improve performance on simple datasets but not so much on complex ones.

Attacks and defence methods could also be investigated for different threat models on the pretraining component of `TabVFL`.

Improving the communication overhead of `TabVFL` is also crucial for future work. Some possibilities of minimizing the communication overhead are the compression of intermediate results and the usage of secure redundancy of data and model parameters to allow for independent local updates.

To mitigate the assumption that one epoch is required for the caching mechanism, a

hybrid solution can be considered by replacing the missing values with zeros if the cache is empty. Moreover, client failures could also be simulated exclusively during inference to show the effectiveness of the cache mechanism.

Lastly, different aggregation methods, such as averaging, could be studied on different datasets to investigate their effect on the latent quality. Changing the aggregation method could possibly enhance the overall performance of `TabVFL` as the intermediate representations of the feature values could be improved.

Bibliography

[1] 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*, vol. abs/2202.04309, 2022.

[2] L. Xia, P. Zheng, J. Li, W. Tang, and X. Zhang, "Privacy-preserving gradient boosting tree: Vertical federated learning for collaborative bearing fault diagnosis," *IET Collaborative Intelligent Manufacturing*, 2022.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[4] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. rahman Mohamed, N. Jaitly, A. W. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, 2012.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv*, vol. abs/1810.04805, 2019.

[6] S. Abdulrahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet of Things Journal*, vol. 8, pp. 5476–5497, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232373329

[7] P. Voigt and A. von dem Bussche, "The eu general data protection regulation (gdpr)," 2017.

[8] D. L. Kiselbach and C. E. Joern, "New consumer product safety laws in canada and the united states: Business on the border," *Global Trade and Customs Journal*, 2012.

[9] K. Atarashi and M. Ishihata, "Vertical federated learning for higher-order factorization machines," in *Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part*

II. Berlin, Heidelberg: Springer-Verlag, 2021, p. 346–357. [Online]. Available: https://doi.org/10.1007/978-3-030-75765-6_28

[10] B. Tan, B. Liu, V. W. Zheng, and Q. Yang, "A federated recommender system for online services," *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020.

[11] D. Cha, M. Sung, and Y. R. Park, "Implementing vertical federated learning using autoencoders: Practical application, generalizability, and utility study," *JMIR Medical Informatics*, vol. 9, 2021.

[12] K.-F. Chu and L. Zhang, "Privacy-preserving self-taught federated learning for heterogeneous data," *ArXiv*, vol. abs/2102.05883, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231879990

[13] A. Khan, M. ten Thij, and A. Wilbik, "Communication-efficient vertical federated learning," *Algorithms*, vol. 15, p. 273, 2022.

[14] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," *ArXiv*, vol. abs/1908.07442, 2019.

[15] S. Li, D. Yao, and J. Liu, "Fedvs: Straggler-resilient and privacy-preserving vertical federated learning for split models," *IACR Cryptol. ePrint Arch.*, vol. 2023, p. 597, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258332102

[16] J. Sun, Z. Du, A. C. Dai, S. Baghersalimi, A. J. Amirshahi, D. Atienza, and Y. Chen, "Robust and ip-protecting vertical federated learning against unexpected quitting of parties," *ArXiv*, vol. abs/2303.18178, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257900954

[17] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics*, 2016.

[18] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, pp. 1 – 19, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:219878182

[19] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, pp. 50–60, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:201126242

[20] R. S. Antunes, C. A. da Costa, A. Küderle, I. A. Yari, and B. Eskofier, "Federated learning for healthcare: Systematic review and architecture proposal," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, pp. 1 – 23, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246531818

[21] K. M. J. Rahman, F. Ahmed, N. Akhter, M. Z. Hasan, R. Amin, K. E. Aziz, A. M. Islam, M. S. H. Mukta, and A. N. Islam, "Challenges, applications and design aspects of federated learning: A survey," *IEEE Access*, vol. 9, pp. 124 682–124 700, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:237521286

[22] Z. Wu, Q. Li, and B. He, "Practical vertical federated learning with unsupervised representation learning," *ArXiv*, vol. abs/2208.10278, 2022.

[23] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *ArXiv*, vol. abs/1711.10677, 2017.

[24] H. Sun, Z. Wang, Y. Huang, and J. Ye, "Privacy-preserving vertical federated logistic regression without trusted third-party coordinator," *2022 The 6th International Conference on Machine Learning and Soft Computing*, 2022.

[25] H. Yu, J. Vaidya, and X. Jiang, "Privacy-preserving svm classification on vertically partitioned data," in *Advances in Knowledge Discovery and Data Mining*, W.-K. Ng, M. Kitsuregawa, J. Li, and K. Chang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 647–656.

[26] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Secure linear regression on vertically partitioned datasets," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 892, 2016.

[27] A. Ustimenko, L. Prokhorenkova, and A. Malinin, "Uncertainty in gradient boosting via ensembles," *ArXiv*, vol. abs/2006.10562, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219792740

[28] P. Bahad and P. Saxena, "Study of adaboost and gradient boosting algorithms for predictive analytics," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:213007856

[29] S. Georganos, T. Grippa, A. N. Gadiaga, C. Linard, M. Lennert, S. Vanhuysse, N. Mboga, E. Wolff, and S. Kalogirou, "Geographical random forests: a spatial extension of the random forest algorithm to address spatial heterogeneity in remote sensing and population modelling," *Geocarto International*, vol. 36, pp. 121 – 136, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:260501090

[30] J. Vaidya, C. Clifton, M. Kantarcioglu, and A. S. Patterson, "Privacy-preserving decision trees over vertically partitioned data," in *TKDD*, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:1297171

[31] B. Hssina, A. Merbouha, H. Ezzikouri, and M. Erritali, "A comparative study of decision tree id3 and c4.5," *International Journal of Advanced Computer Science and Applications*, vol. 4, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:8800056

[32] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, vol. 8, pp. 843–854, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:165163667

[33] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, pp. 87–98, 2019.

[34] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proceedings of the VLDB Endowment*, vol. 13, pp. 2090 – 2103, 2020.

[35] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *IACR Cryptology ePrint Archive*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:31616696

[36] R. Cramer, I. Damgård, and J. B. Nielsen, "Secure multiparty computation and secret sharing," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:28750103

[37] 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," *Proceedings of the 2021 International Conference on Management of Data*, 2021.

[38] Y. Kang, Y. Liu, Y. Wu, G. Ma, and Q. Yang, "Privacy-preserving federated adversarial domain adaption over feature groups for interpretability," *ArXiv*, vol. abs/2111.10934, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:242381627

[39] Q. Zhang, B. Gu, Z. Dang, C. Deng, and H. Huang, "Desirable companion for vertical federated learning: New zeroth-order gradient based algorithm," *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:240230891

[40] N. Hashemi, P. Safari, B. Shariati, and J. K. Fischer, "Vertical federated learning for privacy-preserving ml model development in partially disaggregated networks," *2021 European Conference on Optical Communication (ECOC)*, pp. 1–4, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:244506180

[41] Y. Kang, Y. Liu, and T. Chen, "Fedmvt: Semi-supervised vertical federated learning with multiview training," *ArXiv*, vol. abs/2008.10838, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221293279

[42] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *ArXiv*, vol. abs/1912.12115, 2019.

[43] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *ArXiv*, vol. abs/1812.00564, 2018.

[44] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey, "No peek: A survey of private distributed deep learning," *ArXiv*, vol. abs/1812.03288, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54461890

[45] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:207241585

[46] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "Blindfl: Vertical federated machine learning without peeking into your data," *Proceedings of the 2022 International Conference on Management of Data*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249578995

[47] J. Hou, M. Su, A. Fu, and Y. Yu, "Verifiable privacy-preserving scheme based on vertical federated random forest," *IEEE Internet of Things Journal*, vol. 9, pp. 22 158–22 172, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:237905041

[48] I. Ceballos, V. Sharma, E. Mugica, A. Singh, A. Roman, P. Vepakomma, and R. Raskar, "Splitnn-driven vertical partitioning," *ArXiv*, vol. abs/2008.04137, 2020.

[49] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:5575601

[50] 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," *ArXiv*, vol. abs/2102.08504, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231942635

[51] 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 *USENIX Security Symposium*, 2022.

[52] T. Zou, Y. Liu, Y. Kang, W. Liu, Y. He, Z. qian Yi, Q. Yang, and Y.-Q. Zhang, "Defending batch-level label inference and replacement attacks in vertical federated learning," *IEEE Transactions on Big Data*, 2022.

[53] B. Balle and Y.-X. Wang, "Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising," in *International Conference on Machine Learning*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:21713075

[54] J. Sun, X. Yang, Y. Yao, and C. Wang, "Label leakage and protection from forward embedding in vertical federated learning," *ArXiv*, vol. abs/2203.01451, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247223065

[55] X. Jin, P.-Y. Chen, C.-Y. Hsu, C.-M. Yu, and T. Chen, "Cafe: Catastrophic data leakage in vertical federated learning," *ArXiv*, vol. abs/2110.15122, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:240070357

[56] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 181–192, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:224803115

[57] P. Ye, Z. Jiang, W. Wang, B. Li, and B. Li, "Feature reconstruction attacks and countermeasures of dnn training in vertical federated learning," *ArXiv*, vol. abs/2210.06771, 2022.

[58] P. Vepakomma, O. Gupta, A. Dubey, and R. Raskar, "R educing leakage in distributed deep learning for sensitive health data," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:201635379

[59] Y. Kang, J. Luo, Y. He, X. Zhang, L. Fan, and Q. Yang, "A framework for evaluating privacy-utility trade-off in vertical federated learning," *ArXiv*, vol. abs/2209.03885, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252118465

[60] Z. Zhao, H. Wu, A. van Moorsel, and L. Y. Chen, "Gtv: Generating tabular data via vertical federated learning," *ArXiv*, vol. abs/2302.01706, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:256598036

[61] 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*, vol. abs/2104.00489, 2021.

Appendix A

This appendix contains supplementary figures that provide additional details and visual support for the topics discussed in the main body of the thesis. Figure A.3 shows the detailed workflow of both pretraining and finetuning steps followed by the `TabVFL-LE` baseline design. In Figure A.4, the detailed workflows of the prior work design `LT` are presented. Finally, Figure A.7 illustrates the difference between the data partitioning imposed by the HFL and VFL scenarios.

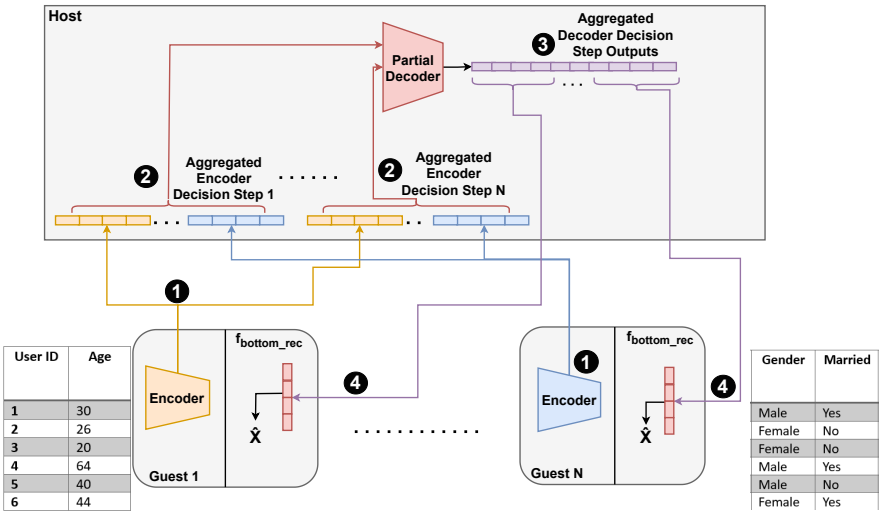


Figure A.1: `TabVFL-LE` pretraining workflow. Each guest sends its encoder decision steps outputs to the host. The outputs are aggregated using summation. The aggregated results are passed to the partial decoder for generating the intermediate results. The results are split uniformly and distributed among the guest clients for reconstruction.

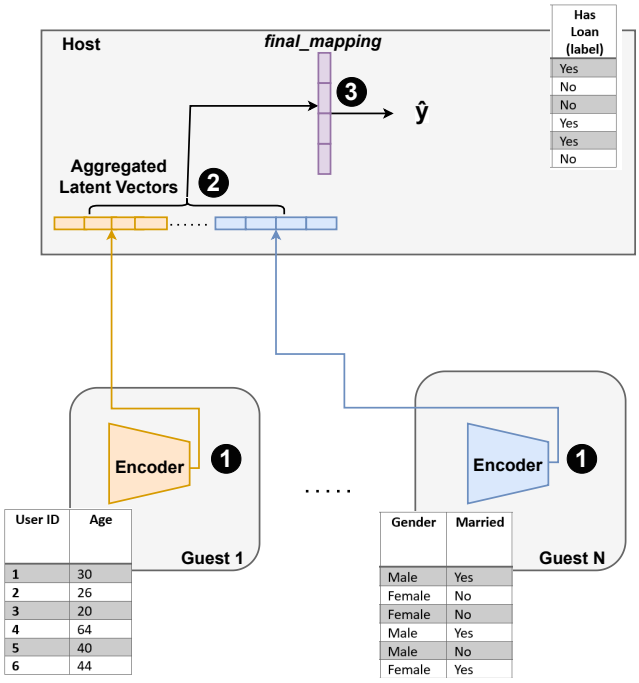


Figure A.2: `TabVFL-LE` finetuning workflow. The encoders in each guest client generate latent representations vectors that are sent to the host. The results are summed up into one representation vector and forwarded to the final mapping FC layer for prediction.

Figure A.3: The pretraining and finetuning workflows of TabNet in VFL with the encoder (`TabVFL-LE`) component residing in each guest client (feature holders).

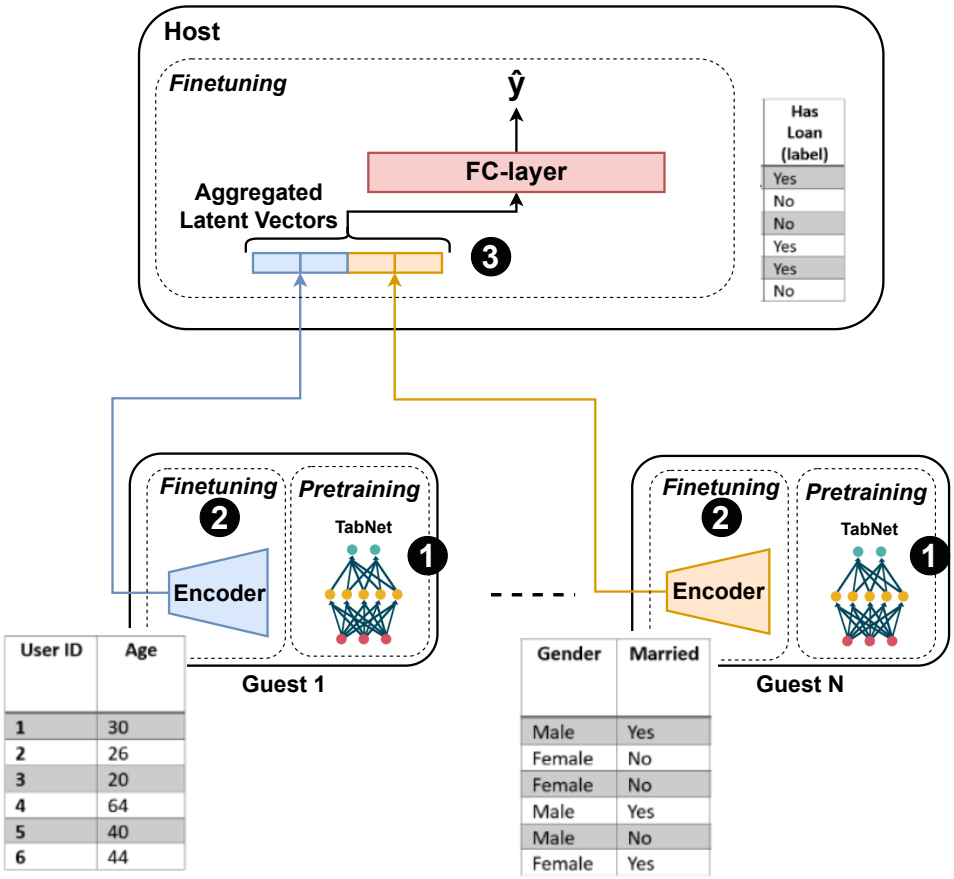


Figure A.4: The pretraining and finetuning workflows of prior work design in VFL using TabNet (`LT`). First, TabNet is pretrained on locally available data by each guest client. The encoder of the pretrained TabNet is reused for finetuning. During finetuning, the latent representations are concatenated at the host and passed through a FC layer for prediction.

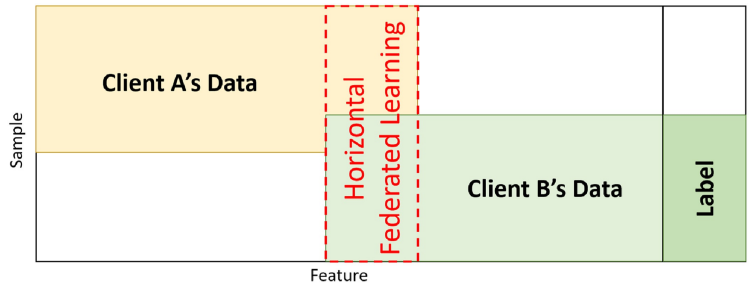


Figure A.5: Horizontal Federated Learning (HFL).

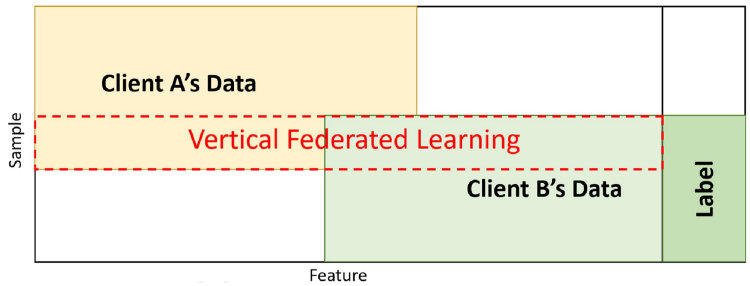


Figure A.6: Vertical Federated Learning (VFL).

Figure A.7: An image showing the difference in the data being considered for the most common data partitioning scenarios in FL [2].