

The Sherpa.ai Blind Vertical Federated Learning Paradigm to Minimize the Number of Communications

Sherpa.ai

research@sherpa.ai

Abstract—Federated Learning (FL) enables collaborative decentralized training across multiple parties (nodes) while keeping raw data private. There are two main paradigms in FL: Horizontal FL (HFL), where all participant nodes share the same feature space but hold different samples, and Vertical FL (VFL), where participants hold complementary features for the same samples. While HFL is widely adopted, VFL is employed in domains where nodes hold complementary features about the same samples. Still, VFL presents a significant limitation: the vast number of communications required during training. This compromises privacy and security, and can lead to high energy consumption, and in some cases, make model training unfeasible due to the high number of communications.

In this paper, we introduce Sherpa.ai Blind Vertical Federated Learning (SBVFL), a novel paradigm that leverages a distributed training mechanism enhanced for privacy and security. Decoupling the vast majority of node updates from the server dramatically reduces node-server communication. Experiments show that SBVFL reduces communication by $\sim 99\%$ compared to standard VFL while maintaining accuracy and robustness. Therefore, SBVFL enables practical, privacy-preserving VFL across sensitive domains, including healthcare, finance, manufacturing, aerospace, cybersecurity, and the defense industry.

1. INTRODUCTION

Federated Learning (FL) [1] enables collaborative training across multiple nodes (parties, clients, devices) while keeping raw data decentralized, sharing only model updates instead of centralizing data as in traditional Machine Learning (ML). FL is typically categorized into Horizontal FL (HFL), where nodes share the same feature space but hold different samples, and Vertical FL (VFL), where nodes hold data with different feature spaces for the same set of samples [2].

Sherpa.ai Blind VFL (SBVFL) is a novel VFL paradigm developed within the research team at Sherpa.ai to address the challenges of healthcare, security, and telecommunication systems. It has been specifically designed to achieve ad hoc performance in VFL by minimizing the communication between nodes and servers.

The cornerstone of this novel approach is the possibility for the nodes to perform independent training without the need for repeated exchanges of information about gradients and parameter updates with the central server. This is made possible by providing each node with *synthetic (surrogate) labels*, which are generated on the server via a secret procedure that prevents clients from inferring the true training labels; the server later maps them back to the original labels to complete training. In this way, SBVFL reduces the number of communication

rounds¹ while achieving predictive performance comparable to that of centralized training, and drastically enhancing privacy and security.

Our simulation experiments, conducted using a Residual Neural Network (ResNet) framework, show that SBVFL can complete a classification task with high accuracy at low computational cost. Furthermore, SBVFL is a general approach, applicable to a wide spectrum of VFL systems in which the nodes may have diverse model architectures.

1.1. Motivation and Challenges

A major obstacle in VFL is the vast number of communication rounds required between the nodes and the server during training [3]–[5]. These repeated exchanges not only increase energy consumption and system latency but also exacerbate the risks of privacy leakage through gradient or label-inference attacks [6]. This high communication overhead is especially problematic in scenarios with strict bandwidth limitations (e.g., satellite or IoT networks) or where energy efficiency is critical (e.g., mobile or defense applications) [7].

Recent efforts have proposed cryptographic approaches, such as Homomorphic Encryption (HE) or Secure Multi-Party (SMPC) Computation, to mitigate privacy risks [8], [9]. However, these methods introduce substantial computational costs and scalability issues. Hence, there is a pressing need for VFL frameworks that simultaneously minimize communication, preserve privacy, and maintain competitive predictive performance.

1.2. Contributions

We propose the SBVFL paradigm, a novel framework designed to address the inefficiencies of traditional VFL. Our main contributions are summarized as follows:

- We introduce a blind learning paradigm that utilizes server-generated synthetic (surrogate) labels, allowing nodes to train independently while significantly reducing the number of required communication rounds.
- We provide a rigorous theoretical analysis, demonstrating that SBVFL retains the classification capability of centralized models while improving privacy guarantees.

¹**Definition.** Let H be the set of training variables needed to carry out a federated training. The minimal set of communications needed to transmit all variables in H is referred to as a communication round.

- We evaluate SBVFL on real-world datasets, including image classification and financial default prediction, demonstrating that our approach drastically reduces communication by $\sim 99\%$ compared to standard VFL while achieving competitive accuracy.
- We highlight SBVFL’s practical advantages for high-stakes applications such as healthcare, finance, and national defense, where data confidentiality, communication efficiency, and robustness against adversarial attacks are paramount.

The paper is organized as follows: Section 2 shows the related work. Section 3 reviews key concepts in FL, focusing on VFL and establishing notation. Section 4 introduces synthetic label generation as a preliminary step for SBVFL, detailed in Section 5. Section 6 provides a theoretical analysis using simultaneous controllability for neural ordinary differential equations (Neural ODEs), while Section 7 discusses SBVFL’s privacy-preserving properties. Section 8 presents experimental results, and Section 9 highlights a real-world application. Section 10 discusses limitations, and Section 11 concludes with future work.

2. RELATED WORK

We provide an overview of related work in VFL, with emphasis on communication efficiency, privacy, and security, while situating our contribution within the broader literature.

2.1. Communication Efficiency in VFL

A major bottleneck in VFL is the large number of communication rounds required to exchange intermediate representations (embedding outputs) and gradients between nodes and the server. Several works have attempted to reduce synchronization costs. Asynchronous and straggler-resilient protocols, such as VAFL [5] and FedVS [3], mitigate delays by relaxing synchronous updates; however, they still require frequent interactions tied to each mini-batch, leading to high overhead. Split- or teacher–student-style variants introduce soft targets to amortize server interactions [4], although these signals can inadvertently leak label information.

Beyond scheduling, techniques from HFL have inspired communication-reduction strategies that can be transferred to VFL, such as gradient compression, quantization, and sparsification [10], [11]. While these methods reduce traffic, they typically maintain per-batch coordination, which limits scalability in scenarios where bandwidth or latency is constrained. In contrast, our approach decouples most node training from server feedback, resulting in significantly fewer communication rounds.

2.2. Privacy and Security in VFL

Even when raw data remain local, exchanged activations or gradients may leak sensitive information. Label inference attacks [6] have demonstrated that server-owned labels can sometimes be reconstructed from node–server communications. To mitigate such risks, cryptographic safeguards like HE and SMPC have been applied to VFL [8], [9]. However, these approaches often introduce high computational

and communication costs, limiting their practicality in large-scale deployments. Hybrid methods that integrate differential privacy (DP) [12] or secure entity resolution [13] offer stronger confidentiality but remain costly under real-world constraints.

Recent work seeks to limit label exposure entirely. BlindFL [14] and related methods [15], [16] differ from our use of “blind”: Fu et al. [14] depend on HE/Secret Sharing, inflating communication; Salmeron et al. [15] describe an HFL scheme without an initial model; and Razavikia et al. [16] target an over-the-air HFL setting. By contrast, SBVFL utilizes server-generated synthetic labels, preserving label privacy without requiring heavy cryptography and eliminating the need for continuous server supervision, thereby reducing communication by orders of magnitude while maintaining competitive accuracy.

2.3. Positioning and Applications

Within this landscape, recent surveys [17], [18] highlight the key challenge of jointly optimizing utility, privacy, and communication efficiency in VFL. Existing methods either reduce communication while risking label leakage or strengthen privacy at the expense of scalability. SBVFL advances the state of the art by achieving both high privacy and drastic communication reduction ($\sim 99\%$) without compromising accuracy.

These advances matter most where confidentiality and bandwidth are tight constraints: in healthcare, FL shows promise across hospitals but suffers from communication overhead [19]; in finance, adoption for credit risk and fraud is curtailed by regulation and cross-organizational sharing costs [20]; and in telecom/IoT, low bandwidth and energy limits dominate [7]. SBVFL addresses these issues with a communication-light, privacy-preserving approach that is well-suited to such settings.

In summary, prior works have significantly advanced VFL in communication scheduling, cryptographic safeguards, and privacy-preserving protocols. However, none simultaneously minimize communication, preserve privacy, and maintain predictive performance at scale. Our proposed SBVFL framework aims to bridge this gap.

3. BACKGROUND AND PROBLEM FORMULATION

FL [1], [19] is an emerging framework that aims to train an ML model (e.g., a deep neural network) on multiple local datasets located in different nodes, without explicitly exchanging data samples. This approach stands in contrast to traditional *centralized* ML techniques, where all local datasets are aggregated and uploaded to a single server. FL, instead, enables multiple nodes to build a common learning model under the supervision of a central *server*, while respecting privacy protocols like the EU General Data Protection Regulation [21], the U.S. National AI Initiative [22], or the UK Data Protection Act [23]. Its applications span a large number of fields, including defense, telecommunications, personalized healthcare, and IoT [24].

FL is typically subdivided into two broad categories, depending on the data distribution (see Figure 1):

- **HFL** [1], [11], [25], also known as the homogeneous data scenario, in which different nodes have the same feature space but little intersection in the sample space. This is a natural data partitioning when different users collaboratively train a model on the same task using FL. A typical example is a federation of hospitals collaboratively training a cancer detection model using chest X-ray images of different patients. Here, the sample IDs correspond to the individual patients in each hospital (with little or no overlap across institutions), while the feature space is the pixel intensity values of the X-ray images, which are common across all hospitals.
- **VFL** [3]–[5], [8], [9], [12], [13], [17], [18], [26], also known as the heterogeneous data scenario, is applicable when several datasets share the same sample ID space but differ in feature space. Consider, for instance, a bank and an e-commerce company in the same city. Their user sets are likely to contain most of the city’s residents, so the intersection of their user base is large. However, their feature spaces are very different: the bank records information about a user’s revenue and expenditure behavior, while the e-commerce company retains the user’s browsing and purchasing history.

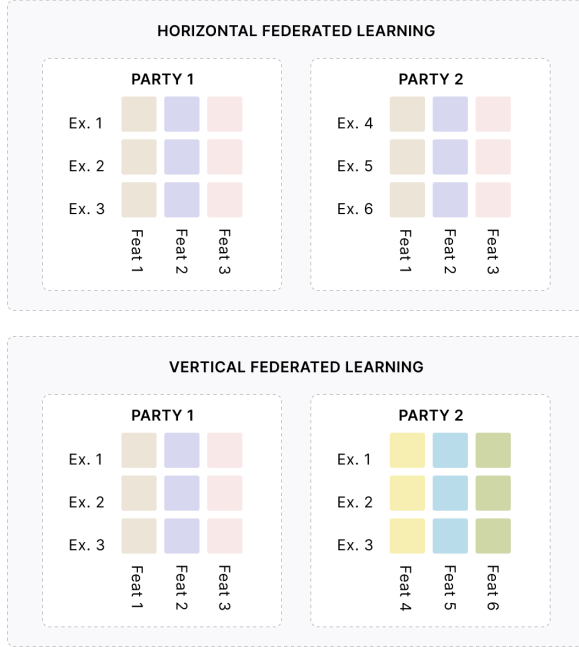


Figure 1: Graphical description of the HFL and VFL scenarios.

In this paper, we focus on VFL for classification problems, where the dataset also contains a finite number of labels and is heterogeneously distributed such that:

- Each node possesses its own portion of the input data but has no information about the other nodes’ inputs or the labels;
- The server exclusively owns the labels, while it is unaware of the input data.

In this setting, none of the federation’s components can perform independent training: the nodes cannot train without knowing the labels, and the server cannot train since it has no input data. Completing the VFL task, therefore, requires the collaborative effort of nodes and the server. This collaboration is organized in a sequence of *training rounds*, each divided into several steps.

Step 1. At the beginning of the round, each node uses its local model and input data to generate predictions on a batch, which are sent to the server.

Step 2. The server aggregates these predictions and uses the result as input data.

Step 3. Since the server now has both inputs and labels, it can evaluate a given loss function and compute gradients to update its parameters. At the same time, the server also computes gradients of the loss function with respect to the node predictions it received, and sends these gradients to the nodes, enabling them to update their parameters. We emphasize that the server is the only federation member capable of computing these gradients, as it alone possesses the full information necessary to evaluate the loss function.

Step 4. The nodes use the gradients received to update their local parameters. The round is then concluded.

Thus, VFL is a more complex process than standard centralized ML. Because of this complexity, three main aspects require special attention when designing a VFL algorithm:

Efficiency – the algorithm should perform the classification task with limited computational resources and a minimal number of communications.

Privacy preservation – privacy must be preserved at all times to ensure that nodes have no access to data they do not own.

Generalization performance – the trained federated model should generalize well, correctly classifying input data not used in its training.

These three aspects are often interlinked. High predictive performance may be achievable by allowing a large number of training rounds, but this comes at the cost of increased computational burden and vulnerability to adversarial attacks. Privacy may be enhanced by cryptographically encrypting shared data, but this raises the computational burden of the VFL process.

As demonstrated in this work, our novel SBVFL scheme enhances efficiency while preserving privacy, maintaining high predictive performance compared to standard VFL. In Ren et al. [4], the concept of a soft label is introduced in a teacher-student VFL scheme. However, unlike synthetic labels, soft labels are essentially a continuous version of the real labels and may thus violate label privacy.

The term *blind* is also used in other technologies², but should not be confused with our usage.

²For example, Fu et al. [14] use HE or Secret Sharing (SS) to preserve privacy. However, these techniques further increase the already high communication overhead of VFL. In Salmeron et al. [15], the term *blind* is used for an HFL scheme that does not require an initial model. An HFL use case is also presented by Razavikia et al. [16]. These uses of *blind* should not be confused with ours.

3.1. Mathematical Description of VFL

We present a general mathematical setting for VFL here. This discussion aims to establish some notation and highlight the key aspects of VFL, which will later facilitate a comparison with our novel SBVFL methodology.

Given an input space $\mathcal{X}$, an output space $\mathcal{Y}$, and a dataset $\mathcal{D} = \{(\vec{x}^i, \vec{y}^i)\}_{i=1}^N \subset \mathcal{X} \times \mathcal{Y}^3$ composed of N known but possibly noisy examples⁴, in a VFL scenario, a federation composed by P nodes $\{\mathcal{F}_k\}_{k=1}^P$ and a server $\mathcal{S}$ aims to approximate an *unknown function* $f : \mathcal{X} \rightarrow \mathcal{Y}$ mapping each input $\vec{x}^i \in \mathcal{X}$ to the corresponding output $\vec{y}^i \in \mathcal{Y}$ by collaboratively training a federated model

$$\text{FedMod} : \mathcal{X} \rightarrow \mathcal{Y} \quad (3.1)$$

such that

$$\text{FedMod}(\vec{x}^i) \sim \vec{y}^i.$$

In this paper, the input space will always be $\mathcal{X} = \mathbb{R}^d$ with $d \geq 2$. Moreover, since we are focusing on classification problems, the output space is a finite set of classes

$$\mathcal{Y} := \{\ell_1, \dots, \ell_M\} \subset \mathbb{R}^m, \quad (3.2)$$

with $m \geq 1$, $|\mathcal{Y}| = M$, and $\ell_p \neq \ell_q$ for all $p \neq q$ ⁵. In (3.2), we denote by $\{\ell_m\}_{m=1}^M$ the *fixed* labels associated with the M possible classes for our classification problem. This means that the labels $\{\vec{y}^i\}_{i=1}^N$ of our dataset all satisfy

$$\vec{y}^i = \ell_m, \text{ for some } m \in [M].$$

Here, we have introduced the notation

$$[q] := \{1, 2, \dots, q\} \text{ for all } q \in \mathbb{N},$$

which we employ in the rest of the paper.

The main difference of VFL with respect to standard centralized ML is that the dataset $\mathcal{D}$ is decentralized among all the members of the federation. In more detail, each training example $\vec{x}^i \in \mathbb{R}^d$, $i \in [N]$, is decomposed as

$$\vec{x}^i = (\vec{x}_1^i, \dots, \vec{x}_k^i, \dots, \vec{x}_P^i) \quad (3.3)$$

and the node $\mathcal{F}_k$ owns the k -th dataset

$$\mathcal{D}_k := \{\vec{x}_k^i\}_{i=1}^N \subset \mathbb{R}^{d_k}, \quad \text{with } \sum_{k=1}^P d_k = d.$$

The labels $\{\vec{y}^i\}_{i=1}^N$, instead, are exclusively owned by the server $\mathcal{S}$ and are kept unknown to the nodes.

³This is the training dataset. Typically, in ML, in addition to this dataset, we have the validation $\mathcal{D}_{\text{validation}} = \{(\vec{x}_{\text{validation}}^i, \vec{y}^i)\}_{i \in I_{\text{validation}}}$ and test $\mathcal{D}_{\text{test}} = \{(\vec{x}_{\text{test}}^i, \vec{y}^i)\}_{i \in I_{\text{test}}}$ datasets.

Definition 3.1. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $(\mathcal{X}, \mathcal{F}_x)$ be a measure space (typically, $\mathcal{F}_x$ is a Borel sigma-algebra) and $2^{\mathcal{Y}}$ be the power set of $\mathcal{Y}$. A pair $(\vec{x}, \vec{y}) \in \mathcal{X} \times \mathcal{Y}$ is said to be a noisy example if it can be written as $(\vec{x}, \vec{y}) = (\vec{x}_{\text{truth}}, \vec{y}_{\text{truth}}) + (W_x, W_y)$, where $(\vec{x}_{\text{truth}}, \vec{y}_{\text{truth}})$ is the ground truth, $W_x : (\Omega, \mathcal{F}) \rightarrow (\mathcal{X}, \mathcal{F}_x)$ is the random variable describing the noise on the $\mathcal{X}$ space and $W_y : (\Omega, \mathcal{F}) \rightarrow (\mathcal{Y}, 2^{\mathcal{Y}})$ is a random variable modeling noise in the labels set $\mathcal{Y}$ (see, e.g., Billings et al. [27] for probability theory).

⁵lowercase p and q index classes

Remark 3.1. In practical applications, the federation might be defined as follows.

- i) $P - 1$ passive nodes $\mathcal{F}_1, \dots, \mathcal{F}_k, \dots, \mathcal{F}_{P-1}$, where each node $\mathcal{F}_k$ owns the unlabelled dataset

$$\mathcal{D}_k = \{\vec{x}_k^i\}_{i=1}^N \subset \mathbb{R}^{d_k};$$

- ii) an active node $\mathcal{F}_A$, possessing the labelled dataset

$$\mathcal{D}_A = \{(\vec{x}_A^i, \vec{y}^i)\}_{i=1}^N \subset \mathbb{R}^{d_P} \times \mathcal{Y}.$$

In this subsection notation, this means the physical active node $\mathcal{F}_A$ contains two logical nodes

- the passive node $\mathcal{F}_P$;
- the server $\mathcal{S}$.

Generally speaking, a strategy to obtain the federated model (3.1) relies on two main ingredients:

1. The class of functions $\mathcal{C}$ to which FedMod should belong;
2. The *training method* to compute FedMod.

The choice of the class of functions $\mathcal{C}$ may depend on several factors, including prior knowledge of the dataset's intrinsic geometry and computational capabilities. Popular choices are Deep Neural Networks [28], Random Forest [29], Gradient Boosting Decision Trees [30], transformers [31], Large Language Models [32] and Residual Neural Networks (ResNets) [33]. The class considered will be denoted by

$$\mathcal{C} = \{\text{FedMod}_{\boldsymbol{\theta}} \mid \boldsymbol{\theta} \in \Theta\},$$

where $\boldsymbol{\theta}$ is a vector containing all the model parameters belonging to some Hilbert space Θ . In what follows, to simplify the notation, we omit the subscript $\boldsymbol{\theta}$ when referring to the federated model FedMod, leaving its dependence on the model's parameters understood.

As for the training method, a common approach [5] is based on *empirical risk minimization*. This consists of finding parameters $\boldsymbol{\theta}^*$ solving

$$\boldsymbol{\theta}^* \in \underset{\Theta}{\operatorname{argmin}} J(\boldsymbol{\theta}), \quad (3.4)$$

with $J : \Theta \rightarrow \mathbb{R}$ defined, for instance, by

$$J(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^N \text{loss}(\text{FedMod}_{\boldsymbol{\theta}}(\vec{x}^i), \vec{y}^i) + \alpha \|\boldsymbol{\theta}\|_{\Theta}^2, \quad (3.5)$$

where the continuous loss function

$$\text{loss} : \mathbb{R}^d \times \mathcal{Y} \rightarrow \mathbb{R}^+$$

penalizes the mismatch between the predictions $\text{FedMod}_{\boldsymbol{\theta}}(\vec{x}^i)$ and the labels $\vec{y}^i$, while the *regularization* term $\alpha \|\boldsymbol{\theta}\|_{\Theta}^2$ penalizes the parameters complexity, $\|\cdot\|_{\Theta}$ being the Hilbertian norm. The effect of this penalization is modulated by the weighting factor $\alpha > 0$.

We stress, however, that because of the heterogeneous data distribution in VFL, to solve the minimization problem (3.4)-(3.5), neither the nodes nor the server can work independently, as they all miss some fundamental piece of information about the dataset. In fact, this minimization process requires the collaborative effort of nodes and the server.

For this reason, FedMod_θ is made of the composition of nodes' models, each belonging to a function class chosen by each node. In the following sections, we will provide a brief description of how this collaborative minimization is implemented. To this end, for all $k \in \llbracket P \rrbracket$, let us denote with

$$\text{Mod}_k : \mathbb{R}^{d_k} \rightarrow \mathbb{R}^{q_k},$$

the local model of the node $\mathcal{F}_k$, $q_k \in \mathbb{N} \setminus \{0\}$ being the output dimension. We stress that, in general, this output dimension may be different from the local model's input dimension d_k . The model Mod_k belongs to a class of functions

$$\mathcal{C}_k = \{\text{Mod}_{k,\theta_k} \mid \theta_k \in \Theta_k\}.$$

Analogously, also the server $\mathcal{S}$ possesses its own model

$$\text{ServerMod} : \mathbb{R}^q \rightarrow \mathbb{R}^m,$$

with $q = \sum_{k=1}^P q_k$, which is chosen in the class

$$\mathcal{C}_s = \{\text{ServerMod}_{\theta_s} \mid \theta_s \in \Theta_s\}.$$

Hence, the federated model on the full input $\vec{x}$ can be written in the compact form

$$\text{FedMod}(\vec{x}) := \text{ServerMod}(\text{Mod}_1(\vec{x}_1), \dots, \text{Mod}_P(\vec{x}_P)), \quad (3.6)$$

the input $\vec{x}$ being decomposed as in (3.3).

As we did before, in what follows we will simplify the notation by dropping the subscripts θ_k and θ_s , indicating the nodes' and server's model parameters.

As anticipated in Section 1 [5], the collaborative empirical risk minimization process is organized in a sequence of rounds, each divided into several steps. Let $I \subset \llbracket N \rrbracket$ be the indices associated with one batch.

VFL training step 1. At the beginning of the round, each node uses its local model and input data to generate predictions that are sent to the server. For every $i \in I$ and $k \in \llbracket P \rrbracket$, these predictions are obtained by computing

$$\text{out}_k^i := \text{Mod}_k(\vec{x}_k^i) \in \mathbb{R}^{q_k}.$$

VFL training step 2. For all $i \in I$, the server aggregates the predictions $\{\text{out}_k^i\}_{k=1}^P$ into a concatenated vector

$$\vec{x}_s^i := [\text{out}_1^i, \dots, \text{out}_P^i] \in \mathbb{R}^q,$$

that is used as input for its model ServerMod .

VFL training step 3. Since the server now has both inputs and labels, it can evaluate

$$\text{FedMod}(\vec{x}^i) = \text{ServerMod}(\vec{x}_s^i)$$

and the loss function in (3.5). It can then compute gradients to update its parameters θ_s through

$$\theta_s \leftarrow \text{ServerUpdate}(\theta_s, \nabla_{\theta_s} J). \quad (3.7)$$

Here, ServerUpdate is a gradient-based update procedure, such as one iteration of Stochastic Gradient Descent. At the same time, for all $i \in I$ and $k \in \llbracket P \rrbracket$, the server also computes $\nabla_{\text{out}_k^i} J$, i.e., the gradients of J with respect to the node's predictions he received. These gradients are sent to the nodes, enabling their parameters to be updated.

VFL training step 4. In order to update its local parameters θ_k , each node $\{\mathcal{F}_k\}_{k=1}^P$ should apply

$$\theta_k \leftarrow \text{NodeUpdate}(\theta_k, \nabla_{\theta_k} J), \quad (3.8)$$

where NodeUpdate is once again a gradient-based update procedure. However, none of the $\mathcal{F}_k$ is in a position to compute the above gradients, since they do not know the labels needed to evaluate J . Furthermore, not even the server $\mathcal{S}$ can compute the required gradients, as it has no information about the nodes' local parameters. To overcome this impasse, one may observe that, for all $i \in I$ and $k \in \llbracket P \rrbracket$,

$$\nabla_{\theta_k} J = \nabla_{\text{out}_k^i} J \partial_{\theta_k} [\text{out}_k^i] \quad (3.9)$$

Now, each quantity $\partial_{\theta_k} [\text{out}_k^i]$ can be computed by the corresponding node $\mathcal{F}_k$, since it knows both its parameters and the output of its local model. As for $\nabla_{\text{out}_k^i} J$, this is exactly what the nodes receive from the server in Step 3. Thus, $\nabla_{\theta_k} J$ is obtained through the node-server collaboration. In this way, the local parameters $\{\theta_k\}_{k=1}^P$ can finally be updated. The round is then concluded.

The above four-steps procedure clearly highlights one of the main issues in VFL: to train the federated model, it is required a very large number of communications between nodes and server, depending on the number of data, the batch size (if any) set in the optimizers ServerOptim and NodeOptim , the number of nodes P and the total number of training epochs. In particular, we have

$$\# \text{ communications} \geq 2 \left\lceil \frac{N}{\text{batch size}} \right\rceil P (\# \text{ epochs}). \quad (3.10)$$

As observed by numerous studies (see, e.g., Chen et al. [10] and the references therein), this high number of communications is often a bottleneck in overall VFL performance, both in terms of computational cost and privacy preservation. Our SBVFL paradigm, described in Section 5, has been designed specifically to address these issues, as it enables minimizing the number of nodes and server communications, resulting in a computationally efficient methodology with the added advantage of increased security compared to more traditional VFL approaches.

In Section 4, we introduce a synthetic label generation mechanism that enables supervised learning on a dataset without exposing the ground-truth labels. Section 5 then presents our SBVFL federated scheme. The synthetic label mechanism is the central novelty of SBVFL, yet it is broadly applicable to scenarios where labels are unavailable or cannot be directly shared.

4. SYNTHETIC LABEL GENERATION MECHANISM

The employment of synthetic labels during the federated training is a cornerstone of our SBVFL paradigm and one of the main differences with respect to other existing VFL methodologies. Synthetic labels can be thought of as intermediate goals, replacing final goals (real labels) for label-less nodes. Their generation is entrusted to the server, through a private procedure of which the nodes are unaware. Each node

only receives the outcome of this procedure, and the server is the only one capable of associating these synthetic labels with the real ones. In this way, the privacy of real labels is preserved at all times.

We describe in detail the process of generating these synthetic labels. In this procedure, $\mathcal{S}$ only needs to know the output dimensions $\{q_k\}_{k=1}^P$ of the nodes' models, and operates in three steps.

Step 1. Given the set of real labels

$$\mathcal{Y} = \{\ell_1, \dots, \ell_M\} \subset \mathbb{R}^m,$$

the server, for each node $\mathcal{F}_k$, defines a (possibly nonlinear) matrix-valued operator

$$\Lambda_k : \mathcal{Y} \rightarrow \mathcal{M}^{Q \times q_k}(\mathbb{R}), \quad Q \in \mathbb{N} \setminus \{0\}, \quad (4.1)$$

where $\mathcal{M}^{Q \times q_k}(\mathbb{R})$ denotes the space of real-valued matrices with dimension $Q \times q_k$. In what follows, to simplify the notation, we indicate this space simply with $\mathcal{M}^{Q \times q_k}$. This operator Λ_k associates Q synthetic labels of dimension q_k to each ℓ_m , $m \in \llbracket M \rrbracket$, these Q synthetic labels being the rows $\{\mathbf{M}_m^{(j)}\}_{j=1}^Q$ of the matrix

$$\mathbf{M}_m := \Lambda_k(\ell_m) \in \mathcal{M}^{Q \times q_k}.$$

Step 2. Associated with each Λ_k , the server also defines a *privacy-preserving map*. Here, the number of rows Q of the image of Λ_k can be arbitrary. The more rows Q we take, the more laborious the task of creating synthetic labels will be, but, simultaneously, the more secure the synthetic label generation will be. Indeed, let us assume for one moment that $Q = 1$, i.e., to each real label we associate a unique synthetic one. For example, for a binary classification task, if the real label is $\{0, 1\}$, the synthetic label could be $\{1.3, -0.7\}$ or $\{0.5, 2.1\}$ depending on Q . Then, the node $\mathcal{F}_k$ could cluster its examples, depending on the received synthetic label, and it would suffice that $\mathcal{F}_k$ knows the real label for just one example of each cluster to recover all the real labels. On the other hand, increasing Q , the number of distinct clusters for the same real label, also increases, making this recovery procedure harder. Due to its importance in preserving privacy during the generation of synthetic labels, we refer to Q as the *privacy multiplier*.

$$\begin{aligned} \mathbb{P}_k : \mathbb{R}^{q_k} &\rightarrow \mathcal{Y} \\ \mathbf{M}_m^{(j)} &\mapsto \ell_m \end{aligned} \quad \text{for all } j \in \llbracket Q \rrbracket, \quad (4.2)$$

associating each synthetic label with its corresponding real one. Notice that this $\mathbb{P}_k$ can be seen as the inverse of the map $\Pi_j \circ \Lambda_k$, where for all $j \in \llbracket Q \rrbracket$, Π_j denotes the projection to the j -th row of $\Lambda_k(\ell_m)$. Moreover, we stress that both $\mathbb{P}_k$ and Λ_k are known only by the server $\mathcal{S}$.

Step 3. Assume that $Q \geq 1$ has been fixed and that the operators $\{\Lambda_k\}_{k=1}^P$ have been built so that for all $m \in \llbracket M \rrbracket$ and $p, r \in \llbracket Q \rrbracket$

$$\mathbf{M}_m^{(p)} \neq \mathbf{M}_m^{(r)}, \quad \text{if } p \neq r.$$

Now, for all $i \in \llbracket N \rrbracket$ and $k \in \llbracket P \rrbracket$, the server generates the synthetic label $\vec{z}_k^i \in \mathbb{R}^{q_k}$, associated to the real label $\vec{y}^i$ and the node $\mathcal{F}_k$, by randomly selecting one of the rows of the matrix $\mathbf{M}_{m_i} = \Lambda_k(\ell_{m_i})$, where $\ell_{m_i} \in \mathcal{Y}$ denotes the label class of $\vec{y}^i$.

There are, of course, several ways to make the random selection at Step 3 of the procedure for generating synthetic labels that we have just described. Likewise, there are many ways to construct the maps $\{\Lambda_k\}_{k=1}^P$. In fact, any vector field could be employed. For example, we could define

$$\vec{z}_k^i = \mathbf{a}_j \vec{y}^i + \mathbf{b}_j, \quad j \in \llbracket Q \rrbracket,$$

where for all $j \in \llbracket Q \rrbracket$, $\mathbf{a}_j$ and $\mathbf{b}_j$ are randomly chosen in $\mathcal{M}^{q_k \times n}$ and $\mathbb{R}^{q_k}$, respectively (see Figure 2). This would be analogous to propagating the information of the real label $\vec{y}^i$ along the characteristics of a transport Partial Differential Equation (PDE).

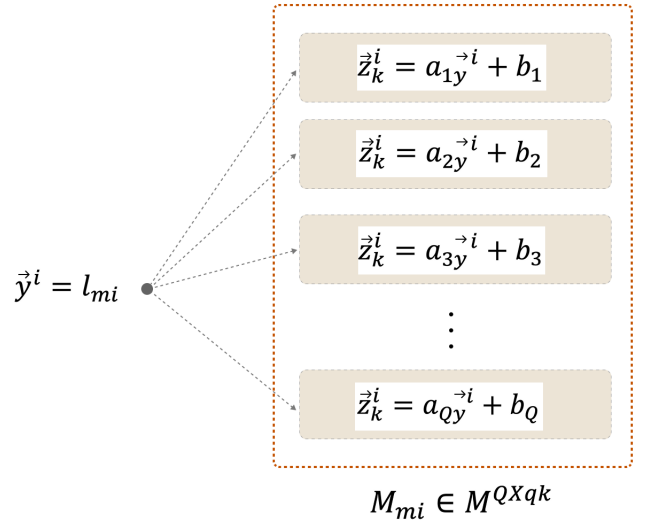


Figure 2: Synthetic labels generation through the flow of a hyperbolic PDE.

5. THE SBVFL PARADIGM

As mentioned, the core idea behind SBVFL is to simplify VFL by employing synthetic labels, as described in Section 4. This brings about a radical change in the federated training procedure, which, instead of being a *circular* process organized into many communication rounds (see Figure 3), adopts the *linear* structure illustrated in Figure 4 without any further iteration.

In particular, in SBVFL, there are only two rounds of training, one for the nodes $\{\mathcal{F}_k\}_{k=1}^P$ in parallel and the other for the server $\mathcal{S}$. In this way, communication between nodes and the server is minimized, and the overall complexity of the training procedure is drastically reduced. In fact, under the SBVFL paradigm, training the federated model requires only one repetition of the following steps.

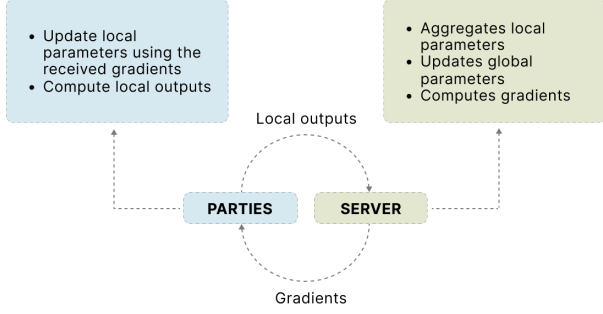


Figure 3: Scheme of the training rounds between nodes and server during VFL.

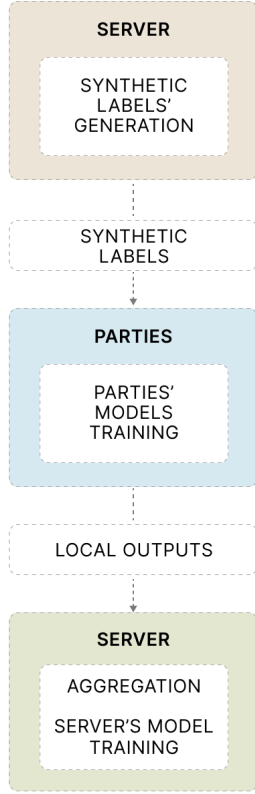


Figure 4: Scheme of the training process for SBVFL.

SBVFL training step 0. The server $\mathcal{S}$ generates synthetic labels that are distributed to all the nodes $\{\mathcal{F}_k\}_{k=1}^P$.

SBVFL training step 1. The nodes use their input data and the received synthetic labels to independently (and in parallel) train their local models $\{\text{Mod}_k\}_{k=1}^P$.

SBVFL training step 2. Once the local models are trained, the nodes in parallel generate predictions on their input data that are sent to the server. For every $i \in \llbracket N \rrbracket$ and $k \in \llbracket P \rrbracket$, these predictions are obtained by computing

$$\text{out}_k^i = \text{Mod}_k(\vec{x}_k^i).$$

SBVFL training step 3. For all $i \in \llbracket N \rrbracket$, $\mathcal{S}$ aggregates

the predictions $\{\text{out}_k^i\}_{k=1}^P$ into a concatenated vector

$$\vec{x}_s^i = [\text{out}_1^i, \dots, \text{out}_P^i] \in \mathbb{R}^q,$$

which is used as input for its model ServerMod .

SBVFL training step 4. $\mathcal{S}$ uses the aggregated input vectors $\{\vec{x}_s^i\}_{i=1}^N$ and the labels $\{\vec{y}^i\}_{i=1}^N$ to train its model.

This training procedure that we have just described is summarized in our SBVFL Algorithm 1.

Remark 5.1. The above architecture can be simplified by removing the SBVFL training steps 0 and 1. The resulting architecture is named in Sherpa.ai Light VFL (LVFL).

Let us summarize the main steps below.

LVFL training step 0. The passive nodes in parallel initialize their local models $\{\text{Mod}_k\}_{k=1}^P$ and generate predictions on their input data that are sent to the server. For every $i \in \llbracket N \rrbracket$ and $k \in \llbracket P \rrbracket$, these predictions are obtained by computing

$$\text{out}_k^i = \text{Mod}_k(\vec{x}_k^i).$$

LVFL training step 1. For all $i \in \llbracket N \rrbracket$, the server $\mathcal{S}$ aggregates the predictions $\{\text{out}_k^i\}_{k=1}^P$ into a concatenated vector

$$\vec{x}_s^i = [\text{out}_1^i, \dots, \text{out}_P^i] \in \mathbb{R}^q,$$

that is used as input for its model ServerMod .

LVFL training step 2. $\mathcal{S}$ uses the aggregated input vectors $\{\vec{x}_s^i\}_{i=1}^N$ and the labels $\{\vec{y}^i\}_{i=1}^N$ to train its model.

The number of communications is equal to the number of nodes P . If, for each $k \in \{1, \dots, P\}$, the $\mathcal{F}_k$'s model $\text{Mod}_k : \mathbb{R}^{d_k} \rightarrow \mathbb{R}^{q_k}$ is one-to-one, it would suffice ServerMod to be complex enough to have a successful classification.

The drawback of LVFL is that the server training task may be computationally intensive, whereas in SBVFL, the passive nodes' training serves as a preprocessing step, which alleviates the server's task.

Finally, to conclude this section, let us highlight that to perform the federated training, our SBVFL Algorithm 1 requires only $2P$ node-server communications, since the only information shared is the synthetic labels sent from the server to the nodes and the local training outputs sent from the nodes to the server. This represents a substantial reduction compared to traditional VFL (see (3.10)), which yields several benefits: it reduces computing time, costs, and carbon footprint [7], and enhances privacy. Note that, in VFL, the privacy of the labels owned by the server can be in danger, as illustrated by Fu et al. [6].

6. THEORETICAL ANALYSIS OF SBVFL

To further substantiate SBVFL's effectiveness, we provide a theoretical analysis that leverages control theory to demonstrate how SBVFL can achieve the same classification capability as a centralized model.

To this end, we adopt the viewpoint of [34] and discuss the efficacy of our blind methodology by relating it to the simultaneous controllability properties of NeurODEs [35]–[37], a continuous version of ResNets.

Algorithm 1 SBVFL Training

Input: datasets $\mathcal{D}_k = \{\vec{x}_k^i\}_{i=1}^N$ owned by the node $\mathcal{F}_k$ for any $k \in \llbracket P \rrbracket$. Labels $\{\vec{y}^i\}_{i=1}^N$ owned by the server $\mathcal{S}$.

Output: trained federated model (3.1), classifying the inputs $\{\vec{x}^i = (\vec{x}_1^i, \dots, \vec{x}_P^i)\}_{i=1}^N$ to the corresponding labels $\{\vec{y}^i\}_{i=1}^N$.

Procedure:

- 1: For every $k \in \llbracket P \rrbracket$, the server $\mathcal{S}$ generates synthetic labels $\{\vec{z}_k^i\}_{i=1}^N \subset \mathbb{R}^{q_k}$ and sends them to node $\mathcal{F}_k$.
- 2: **for** $k \in \llbracket P \rrbracket$ **in parallel do**
- 3: Each node $\mathcal{F}_k$ trains its model Mod_k via empirical risk minimization, by looking for the parameters θ_k^* solving

$$\theta_k^* = \underset{\theta_k \in \Theta_k}{\operatorname{argmin}} J_k,$$

with $J_k : \Theta_k \rightarrow \mathbb{R}$ given by

$$J_k(\theta_k) = \frac{1}{N} \sum_{i=1}^N \text{loss}(\text{Mod}_k(\vec{x}_k^i), \vec{z}_k^i) + \alpha_k \|\theta_k\|_{\Theta_k}^2.$$

- 4: $\mathcal{F}_k$ sends $\{\text{Mod}_k(\vec{x}_k^i)\}_{i=1}^N$ to the server $\mathcal{S}$.
- 5: **end for**
- 6: The server $\mathcal{S}$ aggregates the received output into the vectors $\{\vec{x}_s^i\}_{i=1}^N$ with
- 7: The server $\mathcal{S}$ trains its model via empirical risk minimization, by looking for the parameters θ_s^* solving

$$\theta_s^* = \underset{\theta_s \in \Theta_s}{\operatorname{argmin}} J_s,$$

with $J_s : \Theta_s \rightarrow \mathbb{R}$ given by

$$J_s(\theta_s) = \frac{1}{N} \sum_{i=1}^N \text{loss}(\text{ServerMod}(\vec{x}_s^i), \vec{y}^i) + \alpha_s \|\theta_s\|_{\Theta_s}^2.$$

In particular, for all $k \in \llbracket P \rrbracket$, the local model Mod_k of the node $\mathcal{F}_k$ is the following NeurODE

$$\begin{cases} \dot{\mathbf{x}}_k(t) = \mathbf{w}_k(t) \sigma(\mathbf{a}_k(t) \cdot \mathbf{x}_k(t) + b_k(t)), & t \in (0, 1) \\ \mathbf{x}_k(0) = \vec{x}_k \in \mathbb{R}^{d_k}, \end{cases} \quad (6.1)$$

with controls $\mathbf{a}_k, \mathbf{w}_k \in L^\infty(0, 1; \mathbb{R}^{d_k})$ and $b_k \in L^\infty(0, 1; \mathbb{R})$, to which it corresponds a flow

$$\begin{aligned} \phi_k(\cdot; \mathbf{a}_k, b_k, \mathbf{w}_k) : \mathbb{R}^{d_k} &\rightarrow \mathbb{R}^{d_k} \\ \vec{x}_k &\mapsto \mathbf{x}_k(1). \end{aligned}$$

Here, the controls $\mathbf{a}_k$, b_k , and $\mathbf{w}_k$ play the role of the model's parameters θ_k in our previous discussion in Section 3.

Analogously, also the server's model ServerMod is given by a NeurODE

$$\begin{cases} \dot{\mathbf{x}}_s(t) = \mathbf{w}_s(t) \sigma(\mathbf{a}_s(t) \cdot \mathbf{x}_s(t) + b_s(t)), & t \in (0, 1) \\ \mathbf{x}_s(0) = \vec{x}_s, \end{cases} \quad (6.2)$$

with associated flow

$$\begin{aligned} \phi_s(\cdot; \mathbf{a}_s, b_s, \mathbf{w}_s) : \mathbb{R}^d &\rightarrow \mathbb{R}^d \\ \vec{x}_s &\mapsto \mathbf{x}_s(1). \end{aligned}$$

Within this framework, the federated model makes predictions on the input data by following a two-step procedure.

Prediction step 1. For all $i \in \llbracket N \rrbracket$ and $k \in \llbracket P \rrbracket$, the node $\mathcal{F}_k$ solves its NeurODE (6.1) with initial datum $\vec{x}_k^i \in \mathbb{R}^{d_k}$ and sends the final-time solution $\mathbf{x}_k^i(1)$ to the server $\mathcal{S}$. Notice that, this time, the output $\mathbf{x}_k^i(1)$ being given by the flow of an ODE, the input and output dimensions of Mod_k coincide ($q_k = d_k$).

Prediction step 2. The server $\mathcal{S}$ aggregates the received outputs into

$$\vec{x}_s^i = [\mathbf{x}_1^i(1), \dots, \mathbf{x}_P^i(1)] \in \mathbb{R}^d,$$

solves its NeurODE (6.2) with initial datum $\vec{x}_s^i$ and generates the prediction

$$\text{FedMod}(\vec{x}^i) = \mathbb{P}[\mathbf{x}_s^i(1)],$$

where $\mathbb{P}$ is the projector

$$\begin{aligned} \mathbb{P} : \mathbb{R}^d &\rightarrow \mathcal{Y} \\ \vec{z} &\mapsto \sum_{r=1}^M \vec{y}^{[r]} \chi_{\Omega_r}(\vec{z}), \end{aligned} \quad (6.3)$$

defined by the partition $\{\Omega_r\}_{r=1}^M$ of $\mathbb{R}^d$

$$\mathbb{R}^d = \Omega_1 \sqcup \dots \sqcup \Omega_M \quad (6.4)$$

where Ω_r has nonempty interior for any $r \in \{1, \dots, M\}$. Note that the scope of the projector (6.3) is then twofold:

- dimensional projection from $\mathbb{R}^d$ to $\mathbb{R}^m$;
- mapping to the finite set $\mathcal{Y}$.

The training of FedMod , instead, is carried out employing the simultaneous controllability algorithm defined in [34, Theorem 1], which essentially consists of a constructive procedure to compute controls $\{(\mathbf{a}_k^*, b_k^*, \mathbf{w}_k^*)\}_{k=1}^P$ and $(\mathbf{a}_s^*, b_s^*, \mathbf{w}_s^*)$ such that

$$\mathbb{P}(\mathbf{x}_s^i(1)) = \vec{y}^i \text{ for all } i \in \llbracket N \rrbracket.$$

We emphasize that, as observed in [38], optimal control theory can be used to relate this training procedure to the empirical risk minimization process. In fact, a practical procedure to compute the controls $\{(\mathbf{a}_k^*, b_k^*, \mathbf{w}_k^*)\}_{k=1}^P$ and $(\mathbf{a}_s^*, b_s^*, \mathbf{w}_s^*)$ consists precisely in minimizing the functionals $J_k(\theta_k)$ and $J_s(\theta_s)$ introduced in Algorithm 1. Under standard regularity assumptions on the map

$$\theta \mapsto \text{FedMod}_\theta(\vec{x}),$$

for instance assuming it is of class C^1 and weakly continuous for any $\vec{x} \in \mathbb{R}^d$, this procedure can be proven to be successful by means of the Direct Method in the Calculus of Variations [39].

Finally, we can employ [34, Theorem 1] to rigorously prove the efficiency of our SBVFL paradigm in solving the classification task for which it was designed. To this end, in what follows, we assume $d_k \geq 2$ for all $k \in \llbracket P \rrbracket$. This is required to apply the simultaneous controllability strategy of Ruiz et al. [34]. We stress, however, that this assumption is

not restrictive. Indeed, in case the data allocated to one node is in $\mathbb{R}$, we can easily embed them into $\mathbb{R}^2$ by applying, for instance, the map

$$\begin{aligned} E: \mathbb{R} &\rightarrow \mathbb{R}^2 \\ \xi &\mapsto (\xi, 0). \end{aligned}$$

With this said, we can prove the following result.

Theorem 6.1. *Suppose that $d_k \geq 2$ for all $k \in \llbracket P \rrbracket$. Assume that the centralized dataset $\{(\vec{x}^i, \vec{y}^i)\}_{i=1}^N$ has distinct input data $\vec{x}^i \neq \vec{x}^j$ for $i \neq j \in \llbracket N \rrbracket$. Then, the federated model FedMod, trained by Algorithm 1, is able to classify each input into the corresponding label. Namely,*

$$\text{FedMod}(\vec{x}^i) = \vec{y}^i, \quad \text{for all } i \in \llbracket N \rrbracket.$$

Proof. The proof follows directly from the simultaneous controllability results for NeurODEs obtained by Ruiz et al. [34]. In fact, we only have to show that the server $\mathcal{S}$, after having received from the nodes the outputs $\{\phi_k(\vec{x}_k^i)\}$ for all $i \in \llbracket N \rrbracket$ and $k \in \llbracket P \rrbracket$, is capable of classifying the aggregated vectors

$$\vec{x}_s^i = [\phi_1(\vec{x}_1^i), \dots, \phi_P(\vec{x}_P^i)] \in \mathbb{R}^d$$

by computing controls $\mathbf{a}_s$, b_s and $\mathbf{w}_s$ such that

$$\phi_s(\vec{x}_s^i) = \vec{y}^i, \quad \text{for all } i \in \llbracket N \rrbracket.$$

To this end, it is enough to show that the input data of the server are all different, that is,

$$\vec{x}_s^i \neq \vec{x}_s^j \quad \text{for all } i \neq j \in \llbracket N \rrbracket. \quad (6.5)$$

Indeed, once (6.5) is satisfied, we can apply the constructive procedure of [34, Theorem 1] to determine the desired controls.

Now, since by assumption $\vec{x}^i \neq \vec{x}^j$ for all $i \neq j \in \llbracket N \rrbracket$, there exists at least one $k \in \llbracket P \rrbracket$ such that

$$\vec{x}_k^i \neq \vec{x}_k^j.$$

Therefore, using the uniqueness of solutions to (6.1), we get

$$\phi_k(\vec{x}_k^i) \neq \phi_k(\vec{x}_k^j),$$

from which (6.5) follows immediately. Our proof is then concluded. $\square$

Remark 6.1. In light of remark 5.1, the conclusions of Theorem 6.1 hold if SBVFL is replaced with LVFL.

Indeed, the proof only relies on the one-to-one property of the passive nodes' models and the controllability properties of the server model.

7. PRIVACY AND SECURITY ANALYSIS OF SBVFL

When presenting SBVFL in Section 5, we have claimed that our approach enhances privacy in VFL. The aim of this section is to provide a more concrete ground for these claims.

To this end, we consider a scenario in which one of the members of the federation, either a node or the server, is a malicious agent that wants to acquire information about data that is not in its possession. This can be translated into three possible types of attacks.

Node-to-server attack: one of the nodes $\mathcal{F}_k$ wants to recover the data of the server $\mathcal{S}$.

Server-to-node attack: the server $\mathcal{S}$ wants to recover the data of one of the nodes $\mathcal{F}_k$.

Node-to-node attack: one of the nodes $\mathcal{F}_k$ wants to recover the data of another node $\mathcal{F}_j$.

In what follows, we adopt the NeurODE perspective from Section 6 to investigate whether these attacks can be successful under the SBVFL protocol that we have designed.

7.1. Node-to-server Attacks

In a node-to-server attack, one of the nodes $\mathcal{F}_k$ tries to recover the data in possession of the server $\mathcal{S}$, that is, the labels $\{\vec{y}^i\}_{i=1}^N$ (see Fu et al. [6] for label inference attacks against VFL).

Let us recall that, in the VFL scenario we are considering in this paper, the attacker $\mathcal{F}_k$ knows nothing about the labels, except for the synthetic labels it possesses. Hence, the only information this node can use to launch its attack is the synthetic labels it received from the server at the beginning of training. Nevertheless, this information is not enough to successfully complete the attack. Indeed, from the viewpoint of $\mathcal{F}_k$, these synthetic labels are simply some vectors $\vec{z}_k^i$, whose correlation with the real labels is made possible only through the projection $\mathbb{P}_k$ defined in (4.2), that is known only by the server.

Notwithstanding, we stress that, in some real-life scenarios, the attacker $\mathcal{F}_k$ may know some labels $\{\vec{y}^i\}_{i \in I}$, where $I \subset \llbracket N \rrbracket$. In this case, whenever the attacker receives a synthetic label $\vec{z}_k^j$, with

$$\vec{z}_k^j = \vec{z}_k^i, \quad \text{for some } i \in I,$$

it can deduce that the label for the local input $\vec{x}_k^j$ is $\vec{y}^i$. These attacks can be fully neutralized by generating distinct synthetic labels.

$$\vec{z}_k^i \neq \vec{z}_k^j, \quad \text{for all } i \neq j \in \llbracket N \rrbracket.$$

In summary, our SBVFL paradigm is secure against node-to-server attacks, provided that synthetic label generation is performed appropriately. On the other hand, the classical VFL approach may be vulnerable to these kinds of attacks, as demonstrated by Fu et al. [6].

7.2. Server-to-node Attacks

In a server-to-node attack, the server $\mathcal{S}$ wants to recover the data of one of the nodes $\mathcal{F}_k$, that is, the local dataset $\{\vec{x}_k^i\}_{i=1}^N$. To do that, $\mathcal{S}$ needs to use the final value $\mathbf{x}_k(1)$ of the victim's NeurODE

$$\dot{\mathbf{x}}_k(t) = \mathbf{w}_k(t) \sigma(\mathbf{a}_k(t) \cdot \mathbf{x}_k(t) + b_k(t)), \quad t \in (0, 1). \quad (7.1)$$

The adversary $\mathcal{S}$ is not aware of the specific controls $\mathbf{a}_k$, b_k , and $\mathbf{w}_k$, but knows the architecture of (7.1), namely the space dimension d_k and the activation function σ . Besides, let us assume that $\mathcal{S}$ also knows that each example in the secret $\{\vec{x}_k^i\}_{i=1}^N$ must obey some constraints. We model this assuming the existence of a subset $\mathcal{V} \subseteq \mathbb{R}^{d_k}$, such that $\vec{x}_k^i \in \mathcal{V}$, for any

$i \in \{1, \dots, N\}$. We can then formulate the attack of $\mathcal{S}$ to $\mathcal{F}_k$ in the following way.

Definition 7.1 ($\mathcal{F}_k \leftarrow \mathcal{S}$ attack). *Let $\mathcal{V} \neq \{\mathbf{0}\}$ be a non-empty subset of $\mathbb{R}^{d_k}$. Given the final values $\{\mathbf{x}_k^i(1)\}_{i=1}^N$ of the victim's Neural ODE (7.1), find all possible datasets $\{\vec{\mathbf{x}}_{k,sh}^i\}_{i=1}^N \subset \mathcal{V}$ such that there exist controls $\mathbf{a}_{sh}, \mathbf{w}_{sh} \in L^\infty(0, 1; \mathbb{R}^{d_k})$ and $b_{sh} \in L^\infty(0, 1; \mathbb{R})$ for which the solution to the shadow NeuroODE*

$$\begin{cases} \dot{\mathbf{x}}_{sh}^i(t) = \mathbf{w}_{sh}(t) \sigma(\mathbf{a}_{sh}(t) \cdot \mathbf{x}_{sh}^i(t) + b_{sh}(t)), & t \in (0, 1) \\ \mathbf{x}_{sh}^i(0) = \vec{\mathbf{x}}_{k,sh}^i \end{cases} \quad (7.2)$$

satisfies the final condition

$$\mathbf{x}_{sh}^i(1) = \mathbf{x}_k^i(1), \quad (7.3)$$

for all $i \in \llbracket N \rrbracket$.

According to this Definition 7.1, an adversarial server-to-node attack essentially consists in solving an inverse problem: knowing the final states of (7.1), $\mathcal{S}$ uses (7.2) to try discovering the $\mathcal{F}_k$'s secret $\{\vec{\mathbf{x}}_k^i\}_{i=1}^N$ by computing all the possible initial data leading to those final configurations. Under this optics, an attack would then be unsuccessful whenever (7.2) admits multiple solutions satisfying (7.3), in which case the attacker $\mathcal{S}$ cannot identify the secrets $\vec{\mathbf{x}}_k^i$. This motivates the following definition of $\mathcal{F}_k$ security.

Definition 7.2 ($\mathcal{F}_k$ security). *We say that node $\mathcal{F}_k$ is secure in regard to the server $\mathcal{S}$ if the adversarial attack of Definition 7.1 admits multiple solutions, i.e., if there exist at least two distinct shadow datasets leading to the given outputs.*

We have the following result of privacy preservation, in the spirit of Definition 7.2.

Theorem 7.3. *Let $\mathcal{V} \neq \{\mathbf{0}\}$ be a non-empty subset of $\mathbb{R}^{d_k}$, with cardinality $|\mathcal{V}| \geq 2$. Then, for all $k \in \llbracket P \rrbracket$, node $\mathcal{F}_k$ is secure in regard to $\mathcal{S}$.*

Proof. Take any shadow dataset $\mathcal{D}_{sh} := \{\vec{\mathbf{x}}_{k,sh}^i\}_{i=1}^N \subset \mathbb{R}^{d_k}$, with

$$\vec{\mathbf{x}}_{k,sh}^i \neq \vec{\mathbf{x}}_{k,sh}^j \quad \text{whenever } \mathbf{x}_k^i(1) \neq \mathbf{x}_k^j(1). \quad (7.4)$$

Then, by means of the work done by Ruiz et Zuazua [34, Theorem 2], there exist controls $\mathbf{a}_{sh}, \mathbf{w}_{sh} \in L^\infty(0, 1; \mathbb{R}^{d_k})$ and $b_{sh} \in L^\infty(0, 1; \mathbb{R})$, such that (7.2) admits a solution.

Hence, by Definition 7.1, the shadow dataset $\mathcal{D}_{sh}$ is a solution to the adversarial attacks. Nevertheless, the choices of $\mathcal{D}_{sh}$ fulfilling (7.4) are multiple, since at least the first example $\vec{\mathbf{x}}_{k,sh}^1$ can be chosen arbitrarily in $\mathcal{V}$ (which, by assumption, has at least two elements). This means that there are (at least) two solutions to the adversarial attack and, therefore, the node $\mathcal{F}_k$ is secure in the sense of Definition 7.2. $\square$

Theorem 7.3 can now be used to assess the security of our proposed SBVFL approach against server-to-node attacks. In fact, thanks to the employment of synthetic labels, for all $i \in \llbracket N \rrbracket$ the only information the server $\mathcal{S}$ has about the node $\mathcal{F}_k$ is the final state $\mathbf{x}_k^i(1)$. According to Theorem 7.3, this is not enough for $\mathcal{S}$ to break the security of $\mathcal{F}_k$.

In this regard, in practical applications, NeurODEs are most often replaced by some discretized counterpart like the so-called Residual Neural Network

$$\begin{cases} \mathbf{x}^{t+1} = \mathbf{x}^t + \mathbf{w}^t \sigma(\mathbf{a}^t \cdot \mathbf{x}^t + \mathbf{b}^t), & t \in \{0, \dots, \mathcal{T}\} \\ \mathbf{x}^0 = \vec{\mathbf{x}}_0, \end{cases}$$

In this setting, the control strategy defined by Ruiz et al. [34, Section 4] can still be applied, provided that the controls satisfy some constraints associated with the number $\mathcal{T}$ of layers in the neural network. In view of this, simultaneous controllability holds only for special initial and final data, which are distributed according to a sufficiently simple geometry. The greater the number of layers $\mathcal{T}$, the larger is the space of initial data that are simultaneously controllable. This, clearly, also has effects on privacy, which is enhanced by the model's complexity.

7.3. Node-to-node Attacks

In a node-to-node attack one of the nodes $\mathcal{F}_k$ wants to recover the data of another node $\mathcal{F}_j$, that is, the local dataset $\{\vec{\mathbf{x}}_j^i\}_{i=1}^N$. Nevertheless, the only communications among $\mathcal{F}_k$ and $\mathcal{F}_j$ happen via the server $\mathcal{S}$, and we have already shown in Subsection 7.2 that through server-to-node communications, adversarial attacks are impossible. Hence, SBVFL is a secure environment even against node-to-node attacks.

8. EXPERIMENTS ON BENCHMARK DATASET

We present the results of some simulation experiments performed to test our proposed SBVFL methodology. In particular, we compare the performance of SBVFL with a centralized ML scenario, in which all data are gathered in a single location, as well as with the standard VFL methodology presented in Section 3.

8.1. Dataset and Models

We applied SBVFL to a wide variety of use cases. In this section, we present an application for image classification. In Section 9, we illustrate an application to banking-insurance.

To this end, we simulate a federation of seven nodes $\{\mathcal{F}_k\}_{k=1}^7$ and a server $\mathcal{S}$ collaborating to solve an image classification task. The dataset for our experiments will be EMNIST Digits [40], composed of a total of 280,000 handwritten digit characters ranging from 0 to 9, and the corresponding 10 balanced classes of labels.

Out of these 280,000 images, 240,000 will be used for training and the remaining 40,000 ones will be reserved for testing. Moreover, to adapt the dataset to the federated scenario we are considering, we have performed a partition of the EMNIST Digits images into seven vertical slices (see Figure 5), and assigned each one of these slices to one of the nodes $\{\mathcal{F}_k\}_{k=1}^7$. The labels, instead, are given to the server $\mathcal{S}$. Similar experiments (but with a different number of nodes) have been performed in [41]. In this manuscript, the number of nodes $P = 7$ is chosen to ensure a sufficiently large gap between one-node-only performances and centralized performances.

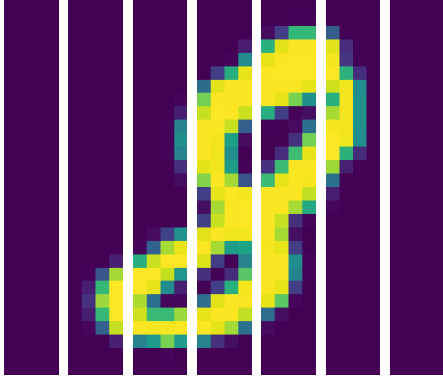


Figure 5: Example of EMNIST Digits image split in vertical slices, to be distributed among the nodes $\{\mathcal{F}_k\}_{k=1}^7$.

Finally, as ML models for our training, we have used ResNets (see Ruiz et al. [34, Formula (1.2)]). Unless stated otherwise, all experiments were conducted using the following hyperparameters: a batch size of 128, a learning rate of 10^{-3} , a weight decay coefficient of 10^{-3} , a hidden dimension of 32, and a network depth of 10 layers.

8.2. Experimentation Testbed

Our experiments were conducted on a Dell computer equipped with a 3.6 GHz Intel Core i7-7700 processor and 32GB of RAM, using Python 3.8.

8.3. Results

In Figure 6 and Table 1, we display the mean accuracy for twenty experiments of centralized ML, VFL, and SBVFL. Notice that the centralized case is displayed for benchmarking purposes. However, it is not possible in practice, since it violates privacy regulations. In Table 1, the standard deviation (STD) of these experiments is also displayed.

We evaluated four training and evaluation configurations.

- 1) **Single node:** Each node $\mathcal{F}_k$ trains its own model locally on its partition of the data; inference is performed on the held-out test set using the corresponding local model.
- 2) **Centralized ML:** Data from all nodes are pooled to train a single model; inference on the test set uses this centralized model.
- 3) **Standard VFL:** A federated model is trained jointly across all nodes; inference on the test set uses the resulting federated model.
- 4) **SBVFL:** A federated model is trained jointly across all nodes under the SBVFL protocol; inference on the test set uses the resulting federated model.

We can clearly observe that both VFL and SBVFL are capable of obtaining high accuracy, at a level that is close to the centralized scenario. Moreover, the low STD values (Table 1) we have obtained certify the stability of our experimental results over multiple repetitions.

Notwithstanding, the real gain of SBVFL with respect to standard VFL is appreciated in the limited number of node-server communications that our approach needs to complete a

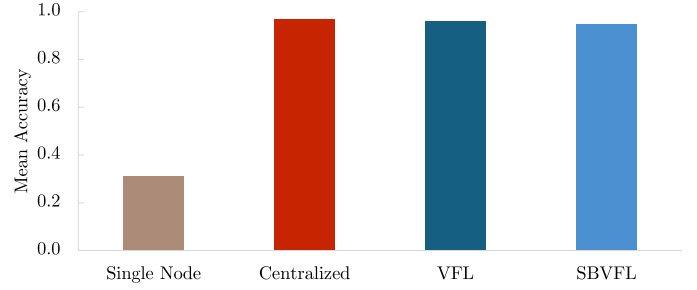


Figure 6: Mean accuracy for twenty random trials of Single Node, centralized ML, VFL, and SBVFL.

Metric	Single node (STD)	Centralized (STD)	VFL (STD)	SBVFL (STD)
Accuracy	0.3105 (0.006)	0.97 (0.0015)	0.96 (0.0053)	0.95 (0.0053)

Table 1: Mean accuracy and Standard Deviation (STD) for twenty random trials of centralized ML, VFL, and SBVFL.

federated training. This is clearly observed in Figure 7, where we can see that our blind methodology allows for a $\sim 99\%$ reduction of these communications, whose number drastically drops from 19,698 to only 14 (that is, twice the number of nodes used in our experiments).

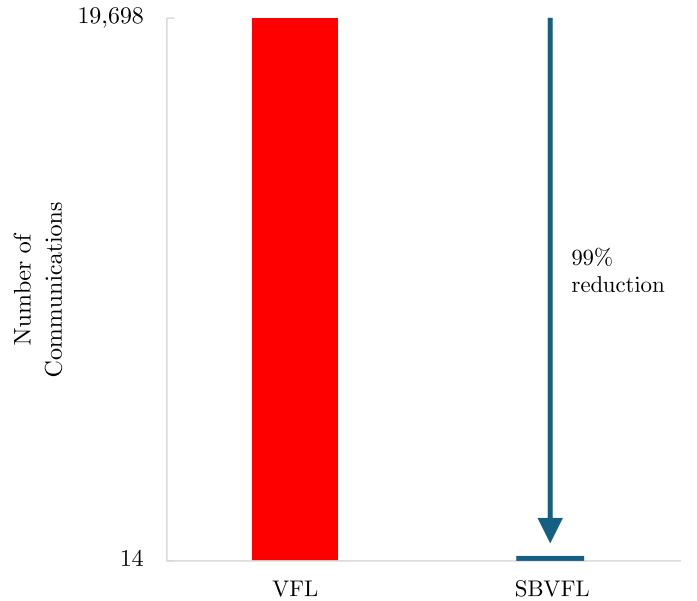


Figure 7: Total number of communications between the nodes and the server to complete the training in VFL and SBVFL.

This is undoubtedly the most significant advancement of SBVFL, which, as discussed in the previous sections, leads to several benefits, including reduced computing time, costs, and carbon footprint, as well as enhanced privacy.

After validating our approach on image data, we next demonstrate SBVFL on a real-world financial dataset in Section 9.

9. REAL-WORLD CASE STUDY (FINANCIAL DEFAULT PREDICTION)

Sherpa.ai’s FL technology is already being applied across a wide range of industries, including healthcare, banking and insurance, Industry 4.0, aerospace, and cybersecurity/defense—thanks to its strong focus on data privacy and security, readiness for enterprise deployments, and user-friendliness.

SBVFL achieves a communication reduction of $\sim 99\%$, delivering significant advantages in scenarios where:

- 1) Data security and privacy are paramount;
- 2) Data transfer between distributed sources is costly or infeasible, such as with low-bandwidth satellite communications;
- 3) Energy efficiency is critical due to hardware limitations, like constrained battery capacity.

These conditions are especially relevant to make predictions in sectors such as banking and insurance, industrial environments, aerospace, and highly sensitive domains like cybersecurity and national defense.

In this use case, we evaluate model performance across various training scenarios involving a bank and an insurance company, focusing on their ability to predict customer behavior while adhering to data privacy regulations. The scenarios range from no data sharing (single node training), where the bank trains its model independently, to full data sharing (centralized training), where both institutions pool their data. FL provides an optimal balance, achieving high predictive accuracy while preserving data privacy. For the FL approach, we implement two methods: standard VFL and SBVFL.

We have used four configurations for the process of training and evaluation:

- 1) **Single Node:** The bank trains a separate model locally using its partitioned data. Inference is conducted on the test subset using the local model.
- 2) **Centralized:** Data from both nodes is pooled into a single repository to train a unified model. Inference is performed on the test subset using the Centralized model.
- 3) **Standard VFL:** the model is trained with data from both nodes. Inference is performed on the test subset using the federated model. The data split across the nodes is detailed in the following section.
- 4) **SBVFL:** the model is trained with data from both nodes. Inference is performed on the test subset using the federated model. The data split across the nodes is detailed in the following section.

These models have been trained using the Sherpa.ai platform and evaluated over a centralized held-out test dataset.

9.1. Problem Formulation

In the domain of consumer finance, predicting the likelihood of credit card default is a crucial task for financial institutions seeking to mitigate risk and maintain portfolio stability. The problem addressed herein is classifying credit card customers into two categories—defaulters and non-defaulters—based on historical and demographic data.

The objective is to develop a predictive model that accurately estimates the probability of a customer defaulting on their credit card payment in the upcoming month. The dataset used for this task consists of 30,000 observations of credit card customers in Taiwan, collected by a major financial institution. Each observation includes 23 explanatory variables, encompassing customer-specific features such as

- demographic attributes (e.g., age, gender, education, and marital status),
- financial data (e.g., credit limit, payment history, and bill statement amounts), and
- behavioral data over the preceding six months (e.g., payment amount and delay status).

The target variable `default payment next month` is a binary indicator of default payment in the next month (1 = default, 0 = non-default).

In the context of Section 3.1, in the centralized case, we would minimize a functional

$$J(\theta) := \frac{1}{N} \sum_{i=1}^N \text{loss}(\text{Mod}_{\theta}(\bar{x}^i), \bar{y}^i) + \alpha \|\theta\|_{\Theta}^2, \quad (9.1)$$

where

- the input space is $\mathcal{X} = \mathbb{R}^{23}$ (the dimension $d = 23$);
- the labels set is $\mathcal{Y} = \{0, 1\}$ (binary classification).

Goal: learn a mapping

$$f : \mathbb{R}^{23} \longrightarrow \{0, 1\} \quad (9.2)$$

such that $f(\bar{x}^i)$ predicts whether the customer i will default.

This classification problem can be addressed using various ML models, including Deep Neural Networks [28], Random Forests [29], Gradient Boosting Decision Trees [30], Large Language Models [32], and Residual Neural Networks (ResNets) [33]. The dataset is imbalanced (the number of defaulting customers is 22.12% of the total). Therefore, metrics for imbalanced datasets, such as the area ratio (as in the seminal paper by Yeh and Lien [42]), F1-score, or AUC, should be adopted to assess model prediction performance.

9.2. Description of the dataset

For this experiment, we use the Default of Credit Card dataset [42], accessible via the UCI ML Repository. This dataset, collected from 30,000 Taiwanese credit card holders between April and September 2005, includes 23 features such as demographic details (age, gender, education, marital status), credit information (credit limit, payment history), and six months of billing and payment records.

The original study aimed to assess the predictive performance of various data mining methods for forecasting default payment probabilities. This dataset is a robust resource for classification tasks in credit risk modeling and financial analytics.

9.3. Federated Setup

This section describes the federated setup for VFL and SBVFL experiments, detailing the data split across the two nodes and the techniques employed. Below, we propose an

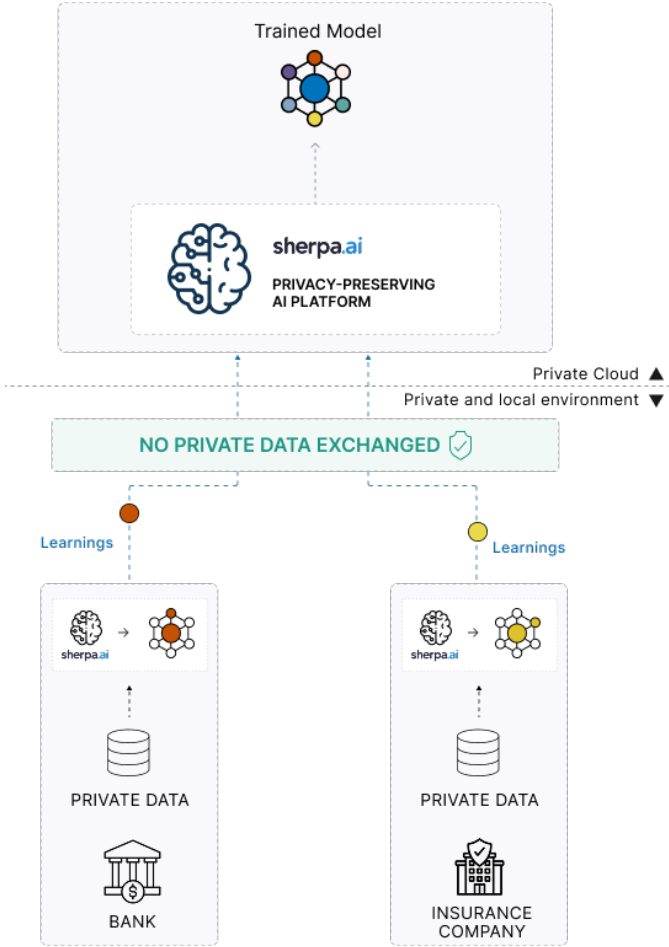


Figure 8: Proposed Architecture for FL.

architecture for FL designed for collaborative model training between two nodes: a bank and an insurance company (see Figure 8).

- **Active Node (Bank):** As in remark 3.1, the bank is a physical *active node* $\mathcal{F}_A$ hosting two logical nodes (the passive node $\mathcal{F}_2$ and the server $\mathcal{S}$), since it possesses both input features and labels. The features contained by the Bank include: BILL_AMT6, BILL_AMT5, AGE, SEX, PAY_AMT5, PAY_AMT3, PAY_5, PAY_6, PAY_4, PAY_AMT6 and the target default payment next month;
- **Passive Node (Insurance Company):** The insurance plays the role of passive node $\mathcal{F}_1$. The features contained by the Insurance Company include: PAY_0, BILL_AMT1, PAY_AMT2, PAY_AMT1, PAY_2, PAY_AMT4, BILL_AMT4, BILL_AMT3, LIMIT_BAL, MARRIAGE, BILL_AMT2, PAY_3, EDUCATION.

9.4. Experiment

In this section, we detail the models and evaluation methods employed in the experiment, which investigates collaborative training between a bank and an insurance company under various data-sharing scenarios.

We optimized model architectures for each training scenario to maximize predictive performance while adhering to privacy constraints. The configurations are as follows:

- For the local (single node) and centralized scenarios, we used a stacked ensemble model combining a Random Forest classifier, a Gradient Boosting classifier, and a Neural Network with an adaptive learning rate to leverage diverse learning strengths.
- For the VFL scenario, we employed Neural Networks, as they meet the differentiability requirements of the VFL architecture (see Subsection 3.1).
- For the SBVFL scenario, we utilized the same stacked ensemble model as in the local and centralized cases, exploiting SBVFL’s flexibility. Additionally, a Neural Network was implemented to enable direct comparison with VFL under identical conditions.

Note that VFL’s federated scheme (described in Subsection 3.1) mandates models to be differentiable with respect to their parameters, limiting the choice of models compared to SBVFL’s more flexible framework.

a) Experimentation Setup.: In all experiments, 20% of the dataset is reserved as a hold-out test set, remaining unseen during training to ensure unbiased evaluation. Of the training dataset, 75% comprises shared customer records common to both the bank and the insurance company, representing the intersection of their datasets.

b) Metrics.: For evaluation, we adopted the area ratio metric [42] to assess model performance across scenarios. To provide further insight, we also report the AUC and F1-score, which are widely employed in similar classification problems.

9.5. Results

This section displays the results of completing several experiments as described in Section 9.4

Experiment I

In this experiment, a single node, centralized, and SBVFL employ the same ML model, a stacked ensemble model combining a Random Forest classifier, a Gradient Boosting classifier, and a Neural Network with an adaptive learning rate to leverage diverse learning strengths.

For standard VFL, we had to adopt a Neural Network, since other models, like the one selected for this experiment, are not supported⁶.

Figure 9 and Table 2 reveal the performance trade-offs of different models trained with single node, VFL, SBVFL, and Centralized training scenarios using the Sherpa.ai platform. Notice that the centralized case is displayed for benchmarking purposes. However, it is not possible in practice, since it violates privacy regulations.

Table 2 shows that SBVFL outperforms the single node baseline and approaches the performance of VFL and centralized models, while requiring dramatically fewer communications. On the one hand, the centralized scenario is displayed for benchmarking purposes. However, it is not possible in practice since it violates privacy regulations. On the other hand, VFL

⁶In standard VFL, we have the constraint of differentiability of the ML model with respect to its parameters

requires excessive communication, making it unfeasible in practice. SBVFL is then a real-world-ready trade-off between privacy and prediction performances. Indeed, SBVFL requires significantly less communication (6 vs. 6,756 for VFL), proving it as a more efficient methodology.

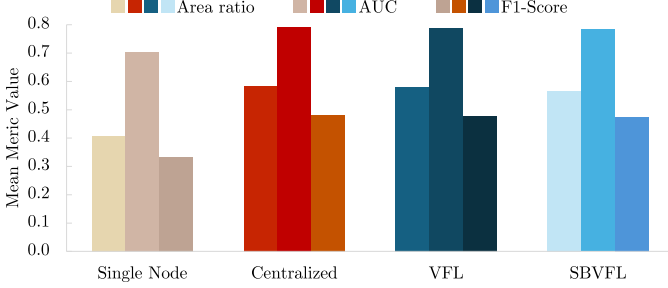


Figure 9: Mean across twenty random trials of single node ML, centralized ML, VFL, and SBVFL.

Metric	Single node (STD)	Centralized (STD)	VFL (STD)	SBVFL (STD)
Area ratio	0.407 (0.0020)	0.583 (0.0018)	0.580 (0.0015)	0.579 (0.0043)
AUC	0.703 (0.0009)	0.792 (0.0009)	0.790 (0.0008)	0.790 (0.0021)
F1-Score	0.334 (0.0041)	0.482 (0.0023)	0.478 (0.0007)	0.476 (0.0093)
Number of communications	N/A	N/A	6,756	6

Table 2: Mean and standard deviation (STD) across 20 experiments of Area Ratio, AUC, and F1-score of local ML (Single Node), Centralized ML, VFL, and SBVFL. The random baseline for the area ratio is 0, whereas for AUC is 0.5.

Experiment II

The purpose of this experiment is to offer another comparison between VFL and SBVFL. This experiment is conducted to complement Experiment I and facilitate a comparison between the two FL paradigms under the same architecture, but with the model replaced by a Neural Network (ResNet).

The average performance metrics across twenty random trials of the trained model with VFL and SBVFL under the same architecture are displayed in Figure 10 and Table 3. SBVFL requires significantly fewer communications, proving to be a more efficient technique.

Comparing the SBVFL results in this section with those in Section 8 (EMNIST Digits), we find that in this real-world setting the advantages of SBVFL are even more pronounced. In Section 8, the large number of nodes and the higher-dimensional data made federated training more challenging. By contrast, here we:

- Demonstrate SBVFL’s flexibility across a broad range of ML models.
- Observe a substantial reduction in communication rounds, which is critical in real-life distributed deployments.

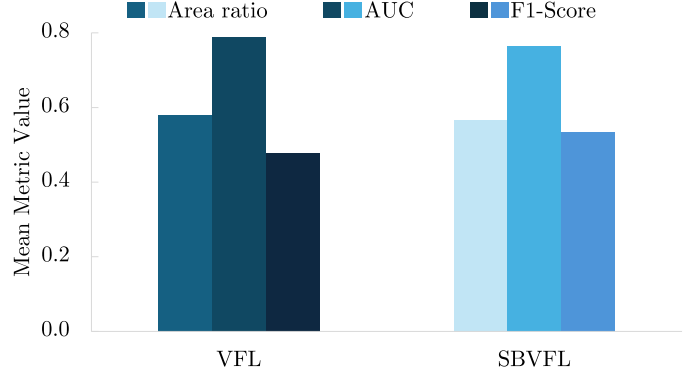


Figure 10: Mean across twenty random trials of VFL and SBVFL.

Metric	VFL (STD)	SBVFL (STD)
Area Ratio	0.580 (0.0015)	0.532 (0.0116)
AUC	0.790 (0.0008)	0.766 (0.0058)
F1 Score	0.478 (0.0007)	0.534 (0.0079)
Number of communications	6,756	6

Table 3: The mean across twenty random trials of the performances of the trained model under VFL and SBVFL.

10. LIMITATIONS

One limitation of applying SBVFL is a slight loss in predictive performance (see Sections 8 and 9). This may be due to the artificially separated manifolds induced by the matrix-valued operator Λ_k defined in Equation 4.1. Specifically, if Q denotes the privacy multiplier, there are $Q - 1$ such manifolds. Another limitation arises from the choice of Q : while higher values of Q strengthen privacy guarantees, they may also slow convergence or increase variance in training dynamics. This trade-off underscores the need for a balance between preserving privacy and enhancing learning efficiency.

11. CONCLUSION

In this paper, we presented SBVFL, a novel vertical FL method that outperforms existing approaches. The term “Blind” reflects that clients train locally without continuous coordination with the server. This is enabled by server-issued synthetic labels, privacy-preserving proxy targets constructed so that clients cannot infer the true labels. During server-side aggregation, these proxies are mapped back to the original labels, preserving utility while substantially strengthening privacy and security.

Our experimental results in Section 8, grounded on the solid theoretical foundations of Theorem 6.1, demonstrate the effectiveness of SBVFL. The method achieves accuracy levels comparable to both centralized and standard VFL, while dramatically reducing node-server communications during training by $\sim 99\%$. SBVFL therefore represents a significant leap forward in preserving security and privacy for distributed heterogeneous training, as discussed in Section 7. Moreover,

this approach unlocks new use cases, as the large number of communications required by the standard VFL approach makes it computationally infeasible.

CONTRIBUTIONS AND ACKNOWLEDGMENTS

Alex Acero
Daniel M. Jimenez-Gutierrez
Dario Pighin
Enrique Zuazua
Joaquin Del Rio
Xabi Uribe-Etxebarria

The authors are presented in alphabetical order by first name.

REFERENCES

- [1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
- [2] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: challenges and applications," *International journal of machine learning and cybernetics*, vol. 14, no. 2, pp. 513–535, 2023.
- [3] S. Li, D. Yao, and J. Liu, "Fedvs: Straggler-resilient and privacy-preserving vertical federated learning for split models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 20 296–20 311.
- [4] Z. Ren, L. Yang, and K. Chen, "Improving availability of vertical federated learning: Relaxing inference on non-overlapping data," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1–20, 2022.
- [5] T. Chen, X. Jin, Y. Sun, and W. Yin, "Vaf: a method of vertical asynchronous federated learning," *arXiv preprint:2007.06081*, 2020.
- [6] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A. X. Liu, and T. Wang, "Label inference attacks against vertical federated learning," in *31st USENIX security symposium (USENIX Security 22)*, 2022, pp. 1397–1414.
- [7] X. Qiu, T. Parcollet, D. J. Beutel, T. Topal, A. Mathur, and N. D. Lane, "Can federated learning save the planet?" *arXiv preprint arXiv:2010.06537*, 2020.
- [8] 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 preprint:1711.10677*, 2017.
- [9] 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.
- [10] T. Chen, G. Giannakis, T. Sun, and W. Yin, "Lag: Lazily aggregated gradient for communication-efficient distributed learning," *Advances in neural information processing systems*, vol. 31, 2018.
- [11] K. Mishchenko, G. Malinovsky, S. Stich, and P. Richtárik, "Proxskip: Yes! local gradient steps probably lead to communication acceleration! finally!" *arXiv preprint arXiv:2202.09357*, 2022.
- [12] C. Wang, J. Liang, M. Huang, B. Bai, K. Bai, and H. Li, "Hybrid differentially private federated learning on vertically partitioned data," *arXiv preprint:2009.02763*, 2020.
- [13] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, J. Joshi, and H. Ludwig, "Fedv: Privacy-preserving federated learning over vertically partitioned data," in *Proceedings of the 14th ACM workshop on artificial intelligence and security*, 2021, pp. 181–192.
- [14] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "Blindfl: Vertical federated machine learning without peeking into your data," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 1316–1330.
- [15] J. L. Salmeron and I. Arévalo, "Blind federated learning without initial model," *Journal of Big Data*, vol. 11, no. 1, p. 56, 2024.
- [16] S. Razavikia, J. M. B. D. S. Júnior, and C. Fischione, "Blind federated learning via over-the-air q-qam," *IEEE Transactions on Wireless Communications*, 2024.
- [17] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances, and challenges," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 3615–3634, 2024.
- [18] A. Khan, M. ten Thij, and A. Wilbik, "Vertical federated learning: A structured literature review," *Knowledge and Information Systems*, pp. 1–39, 2025.
- [19] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.
- [20] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "Ffd: A federated learning based method for credit card fraud detection," in *International conference on big data*. Springer, 2019, pp. 18–32.
- [21] E. Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," April 2016.
- [22] N. A. I. O. (NAHO), "Usa national ai initiative," January 2021, available at: <https://www.ai.gov/>.
- [23] UK, "Uk data protection act," 2018.
- [24] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
- [25] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data *et al.*, "A field guide to federated optimization," *arXiv preprint arXiv:2107.06917*, 2021.
- [26] F. Folino, G. Folino, F. S. Pisani, L. Pontieri, and P. Sabatino, "Efficiently approaching vertical federated learning by combining data reduction and conditional computation techniques," *Journal of Big Data*, vol. 11, no. 1, p. 77, 2024.
- [27] P. Billingsley, *Probability and Measure*, ser. Wiley Series in Probability and Statistics. Wiley, 2012. [Online]. Available: https://books.google.es/books?id=Xe_5TmWV44YC
- [28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [29] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
- [30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
- [31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [32] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.
- [33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [34] D. Ruiz-Balet and E. Zuazua, "Neural ode control for classification, approximation, and transport," *SIAM Review*, vol. 65, no. 3, pp. 735–773, 2023.
- [35] A. Agrachev and A. Sarychev, "Control on the manifolds of mappings as a setting for deep learning," *arXiv preprint:2008.12702*, 2020.
- [36] C. Cuchiero, M. Larsson, and J. Teichmann, "Deep neural networks, generic universal interpolation, and controlled odes," *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 3, pp. 901–919, 2020.
- [37] P. Tabuada and B. Ghahserifard, "Universal approximation power of deep residual neural networks via nonlinear control theory," *arXiv preprint:2007.06007*, 2020.
- [38] C. Esteve, B. Geshkovski, D. Pighin, and E. Zuazua, "Large-time asymptotics in deep learning," *arXiv preprint:2008.02491*, 2020.
- [39] B. Dacorogna, *Direct methods in the calculus of variations*. Springer Science & Business Media, 2007, vol. 78.
- [40] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.
- [41] D. Romanini, A. J. Hall, P. Papadopoulos, T. Titcombe, A. Ismail, T. Cebere, R. Sandmann, R. Roehm, and M. A. Hoeh, "Pyvertical: A vertical federated learning framework for multi-headed splitnn," *arXiv preprint:2104.00489*, 2021.
- [42] I.-C. Yeh and C. hui Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, vol. 36, no. 2, Part 1, pp. 2473–2480, 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417407006719>