Scaling Probabilistic Circuits via Data Partitioning

Jonas Seng[1] **Florian P. Busch**[1,2] **Pooja Prasad**[4] **Devendra Singh Dhami**[4] **Martin Mundt**[5] **Kristian Kersting**[1,2,3]

[1]Computer Science Department, Technical University Darmstadt
[2]Hessian Center for AI (hessian.AI)
[3]German Research Center for AI (DFKI)
[4]Department of Mathematics and Computer Science, Eindhoven University of Technology
[5]Department of Mathematics and Computer Science, University of Bremen

Abstract

Probabilistic circuits (PCs) enable us to learn joint distributions over a set of random variables and to perform various probabilistic queries in a tractable fashion. Though the tractability property allows PCs to scale beyond non-tractable models such as Bayesian Networks, scaling training and inference of PCs to larger, real-world datasets remains challenging. To remedy the situation, we show how PCs can be learned across multiple machines by recursively partitioning a distributed dataset, thereby unveiling a deep connection between PCs and federated learning (FL). This leads to federated circuits (FCs)—a novel and flexible federated learning (FL) framework that (1) allows one to scale PCs on distributed learning environments (2) train PCs faster and (3) unifies for the first time horizontal, vertical, and hybrid FL in one framework by re-framing FL as a density estimation problem over distributed datasets. We demonstrate FC's capability to scale PCs on various large-scale datasets. Also, we show FC's versatility in handling horizontal, vertical, and hybrid FL within a unified framework on multiple classification tasks.

1 INTRODUCTION

Probabilistic Circuits (PCs) are a family of models that provide tractable inference for various probabilistic queries [Poon and Domingos, 2011a, Choi et al., 2020]. This is achieved by representing a joint distribution by a computation graph on which certain structural properties are imposed. While PCs offer significant computational advantages over traditional probabilistic models such as Bayesian networks [Pearl, 1985], further performance gains can be realized by optimizing the compactness of PC representations and tailoring them to specific hardware architectures [Peharz et al., 2020a, Liu et al., 2024]. However, another natural way to scale up PCs by distributing the model over multiple machines is so far underexplored. While models like neural networks can be partitioned over multiple machines with relatively low efforts, partitioning PCs is more challenging as they come with certain structural constraints to ensure the validity of the represented joint distribution. Interestingly, we find an inherent connection between the structure of PCs and the paradigm of federated learning (FL). In PCs, sum nodes combine probability distributions over the same set of variables via a mixture. This resembles the horizontal FL setting [Konečný et al., 2016, Li et al., 2020], where all clients hold the same features but different samples. In contrast, the case of vertical FL [Yang et al., 2019, Wu et al., 2020] in which the same samples are shared, but features are split across clients, can be linked to the product nodes used in PCs, which combine distributions of a disjoint set of variables. Consequently, the hybrid FL [Zhang et al., 2020] setting, where both samples and features are separated across clients, can be represented by a combination of sum and product nodes. Thus, PCs are well positioned to connect all three FL settings in a unified way – an endeavor considered hard to achieve in the FL community [Li et al., 2023a, Wen et al., 2023]. We provide additional details on FL and its associated challenges in App. A.

As a result of this connection, we introduce *federated circuits (FCs)*, a novel FL framework that re-frames FL as a density estimation problem over a set of datasets distributed over multiple machines (subsequently called clients). FCs naturally handle all three FL settings and, therefore, provide a flexible way of scaling up PCs by learning a joint distribution over a dataset arbitrarily partitioned across a set of clients (see Fig. 1 for an illustration). Imposing the same structural properties as for PCs, FCs achieve tractable computation of probabilistic queries like marginalization and conditioning across multiple machines. Based on this, we propose a highly communication-efficient learning algorithm that leverages the semi-ring structure within the design of FCs. Our experimental evaluation shows that FCs outper-

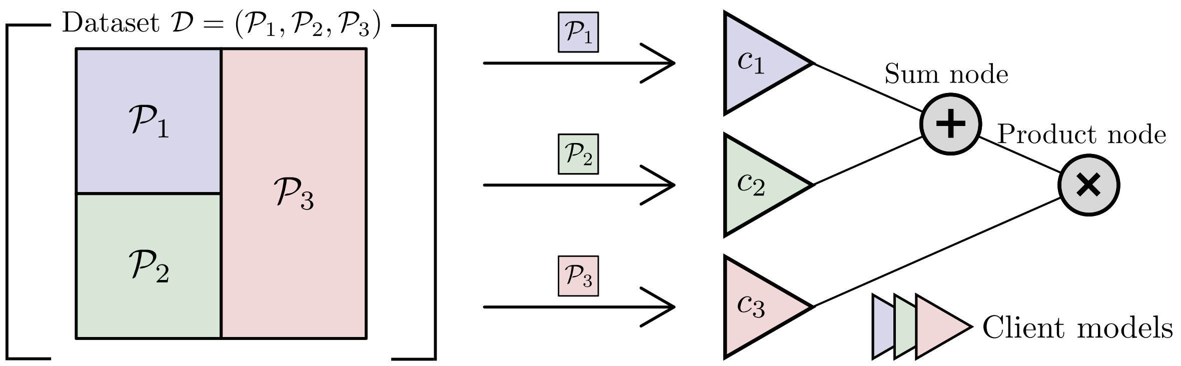

Figure 1: **Scaling PCs via Federated Circuits.** We scale PCs by splitting a dataset $\mathcal{D}$ into n subsets $\{\mathcal{P}_i\}_{i=1}^n$ s.t. $\mathcal{D} = \bigcup_{i=1}^n \mathcal{P}_i$. Each subset is assigned to a client (i.e., machine) c_j, and the resulting federated circuit (FC) is learned jointly by a set of clients. As a novel framework for federated learning (FL), FCs can perform horizontal FL (samples are split across clients), vertical FL (features are split across clients), and hybrid FL (mix of horizontal and vertical).

form EiNets [Peharz et al., 2020a] and PyJuice [Liu et al., 2024] on large-scale density estimation tasks, demonstrating the benefits of scaling up PCs. Additionally, FCs outperform or achieve competing results on various classification tasks in all federated settings compared to state-of-the-art neural network-based and tree-based methods, demonstrating its effectiveness in FL. We make the following contributions: **(1)** We introduce FCs, a communication-efficient and scalable FL framework unifying horizontal, vertical, and hybrid FL by mapping the semantics of PCs to FL. **(2)** We practically instantiate FCs to FedPCs and demonstrate how the FC framework can be leveraged to scale up PCs to large real-world datasets. **(3)** We propose a one-pass training scheme for FedPCs that is compatible with a broad set of learning algorithms. **(4)** We provide extensive experiments demonstrating the effectiveness of our approach for learning large-scale PCs and performing FL. We consider classification and density estimation on tabular and image data.

We proceed as follows: After touching upon related work, we provide the probabilistic view on FL and introduce FCs. Before concluding, we present our extensive experimental evaluation of FedPCs. Our code is publicly available at `https://github.com/J0nasSeng/federated-spn.git`.

2 PRELIMINARIES AND RELATED WORK

In the following, we briefly introduce PCs and FL and give an overview of relevant related work.

Probabilistic Circuits. PCs encode a probability distribution as a computation graph that allows for tractable inference of a wide range of queries such as conditioning and marginalization. Peharz et al. [2015b] define a PC over random variables $\mathbf{X}$ as a tuple $(\mathcal{G}, \phi)$ where $\mathcal{G} = (V, E)$ is a rooted, Directed Acyclic Graph (DAG) and $\phi : V \to 2^{\mathbf{X}}$ is the *scope* function assigning a subset of random variables to each node in $\mathcal{G}$. For each internal node N of $\mathcal{G}$, the scope is defined as $\phi(\mathsf{N}) = \cup_{\mathsf{N}' \in \mathrm{ch}(\mathsf{N})} \phi(\mathsf{N}')$. Each leaf node L computes a distribution/density over its scope. All internal nodes of $\mathcal{G}$ are either a sum node S or a product node P. Each sum node computes a convex combination of its children, and each product node computes a product of its children. With slight abuse of notation we write $\mathsf{S} = \sum_{\mathsf{N} \in \mathrm{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} \mathsf{N}$, and $\mathsf{P} = \prod_{\mathsf{N} \in \mathrm{ch}(\mathsf{P})} \mathsf{N}$. If a PC is *decomposable*, this implies tractability. Decomposability requires that for all $\mathsf{P} \in V$ it holds that $\phi(\mathsf{N}) \cap \phi(\mathsf{N}') = \emptyset$ where $\mathsf{N}, \mathsf{N}' \in \mathrm{ch}(\mathsf{P})$. Further, the *smoothness* condition implies that a PC represents a valid distribution. That is, for each sum $\mathsf{S} \in V$ it holds that $\phi(\mathsf{N}) = \phi(\mathsf{N}')$ where $\mathsf{N}, \mathsf{N}' \in \mathrm{ch}(\mathsf{S})$ [Poon and Domingos, 2011a, Peharz et al., 2015a, Sánchez-Cauce et al., 2021].

Several works have tackled the goal of scaling PCs. Peharz et al. [2020b] have shown that learned PC structures can be replaced by large, random structures to scale to larger problems. Changes in the model layout, such as parallelizable layers via einsum-operations [Peharz et al., 2020a] and a reduction in IO operations [Liu et al., 2024], were also shown to reduce the speed of computation drastically. Liu et al. [2022] improved the performance of PCs by latent variable distillation using deep generative models for additional supervision during learning. Other approaches allow for negative mixture weights [Loconte et al., 2024], introduce complex weights [Wang and Van den Broeck, 2025] or use hypernetworks [Shih et al., 2021].

Federated Learning. In federated learning (FL), a set of data owners (or clients) aim to collaboratively learn an ML model without sharing their data. One distinguishes between horizontal, vertical, and hybrid FL depending on how data is partitioned. In horizontal FL, a dataset $\mathbf{D} \in \mathbb{R}^{n \times d}$ is partitioned such that each client holds the same d features but different, non-overlapping sets of samples. In vertical FL, $\mathbf{D}$ is partitioned such that each client holds the same n samples but different, non-overlapping subsets of the d

features. Hybrid FL describes a combination of horizontal and vertical FL where clients can hold both different (but possibly overlapping) sets of samples and features [Wen et al., 2023, Li et al., 2023a].

For all three FL settings, specifically tailored methods have been proposed to enable collaborative learning of models. The most common scheme in horizontal FL is to average the models of all clients regularly during training [McMahan et al., 2016, Karimireddy et al., 2020a,b, Sahu et al., 2018]. However, model averaging requires each client to share the same model structure. In vertical FL, clients hold different feature sets; thus, there is no guarantee that the model structure can be shared among clients. In these cases, tree-based and neural models are the predominant choice and are typically learned by sharing data statistics or feature representations among clients [Kourtellis et al., 2016, Cheng et al., 2021, Vepakomma et al., 2018, Ceballos et al., 2020, Chen et al., 2020, Liu et al., 2019]. Similar to tree-based vertical FL, tree-based hybrid FL approaches share data statistics (such as histograms) or model properties (such as split rules) among clients [Li et al., 2023b, 2024]. However, tree-based approaches often require complex training procedures.

3 FEDERATED CIRCUITS

This work aims to scale up PCs by splitting data and the model across multiple machines, thus harnessing the availability of compute clusters to train PCs in a federated fashion. In the following, we present an elegant and effective way to achieve that using our novel federated learning framework called federated circuits (FCs) that unifies horizontal, vertical, and hybrid FL.

3.1 PROBLEM STATEMENT & MODELING ASSUMPTIONS

Given a dataset $\mathbf{D}$ and a set of clients $\mathcal{C}$ where each $c \in \mathcal{C}$ holds a subset $\mathbf{D}_c$ of $\mathbf{D}$; we aim to learn the joint distribution $p(\mathbf{X})$ over random variables $\mathbf{X}$ (i.e., the features of $\mathbf{D}$; $\mathbf{D}$ is a $n \times |\mathbf{X}|$ matrix). The partitioning of $\mathbf{D}$ is not further specified. Hence, each client might only hold a subset of random variables $\mathbf{X}_c \subseteq \mathbf{X}$ with support $\mathcal{X}_c$. This can be interpreted as each $c \in \mathcal{C}$ holding a dataset $\mathbf{D}_c \sim p_c$ where p_c is a joint distribution over $\mathbf{X}_c$ which is related to $p(\mathbf{X})$. We introduce two critical modeling assumptions relevant for learning a joint distribution $p(\mathbf{X})$ from a dataset $\mathbf{D}$ partitioned across a set of machines.

Assumption 1 (Mixture Marginals). *There exists a joint distribution p such that the relation $\int_{\mathbf{X}\setminus\mathbf{X}_S} p(x) = \sum_{l\in L} q(L=l)\cdot p_S(x|L=l)$ holds for all $x \in \mathcal{X}$. Here, $\mathbf{X}_S \subseteq \mathbf{X}$ is a subset of the union of client random variables $\mathbf{X} = \cup_{c\in\mathcal{C}}\mathbf{X}_c$. Further, $\mathcal{X} = \bigtimes_{i\in\{1,\dots,|\mathbf{X}|\}} \mathcal{X}_i$ is the support of $\mathbf{X}$, each p_S is defined over $\mathbf{X}_S \subseteq \mathbf{X}$ and q is a prior over a latent L.*

To illustrate, consider a subset of variables $\mathbf{X}_S \subseteq \mathbf{X}$ shared among all clients and its complement $\mathbf{X}_{S^-} = \mathbf{X} \setminus \mathbf{X}_S$. Assumption 1 ensures that the marginal $\int_{\mathbf{X}_{S^-}} p(\mathbf{X})$ is representable as a mixture of all client distributions $p_c(\mathbf{X}_S)$ over $\mathbf{X}_S$. Without Assumption 1, the information stored on the clients' data subsets could not be sufficient to learn $p(\mathbf{X})$.

A key assumption in FL is that data cannot be exchanged among clients. However, dependencies among variables residing on different clients might still exist. To enable learning these “hidden" dependencies while keeping data private, we make the following assumption:

Assumption 2 (Cluster Independence). *Given disjoint sets of random variables $\mathbf{X}_1, \cdots, \mathbf{X}_n$ and a joint distribution $p(\mathbf{X}_1, \cdots, \mathbf{X}_n)$, assume that a latent L can be introduced s.t. the joint can be represented as $p(\mathbf{X}_1, \cdots, \mathbf{X}_n) = \sum_l q(L=l)\prod_{i=1}^n p(\mathbf{X}_i|L=l)$ where q is a prior distribution over the latent L.*

Note that independence is only assumed within clusters in the data. Thus, the latent variable (which can be thought of as "cluster selectors“) allows capturing dependencies among variables residing on different clients. Distributions of the form in Assumption 2 are strictly more expressive than the product distribution and allow more complex modeling.

Further, we want to emphasize that Assumptions 1 and 2 are common throughout PC literature. Assumption 1 forms the basis of the validity of marginalization, and Assumption 2 plays a crucial role in constructing structure learning algorithms for PCs. For more details, refer to App. B.

3.2 BRIDGING PROBABILISTIC CIRCUITS AND FEDERATED LEARNING

We now illustrate an inherent connection between PC semantics and FL, allowing us to scale PCs on large datasets by partitioning the data over a set of clients.

Sum Nodes and Horizontal FL. In horizontal FL, each client is assumed to hold the same set of features, i.e., $\mathbf{X}_c = \mathbf{X}_{c'}$ for all $c, c' \in \mathcal{C}$. However, each client holds different samples. Prominent horizontal FL methods aggregate the *model parameters* of locally learned models regularly during training. However, the horizontal FL setting also precisely corresponds to the interpretation of sum nodes in PCs: A sum node splits a dataset into multiple disjoint clusters. This results in a mixture distribution representing the data that is learned from the disjoint clusters. Thus, instead of aggregating model parameters, we aggregate the *distributions* learned by each client on its subset.

Definition 1 (Horizontal FL). *Assume a set of samples $\mathbf{D}_c \sim p_c$ on each client $c \in \mathcal{C}$, a joint distribution p adher-*

ing to Assumption 1 and that $\mathbf{X}_c = \mathbf{X}_{c'}$ for all $c, c' \in \mathcal{C}$ s.t. $c \neq c'$. We define horizontal FL as fitting a mixture distribution $\hat{p} = \sum_{c \in \mathcal{C}} q(c) \cdot \hat{p}_c$ such that $d(\hat{p}, p)$ and $d(p_c, \hat{p}_c)$ are minimal for all $c \in \mathcal{C}$ where d is a distance metric and $\hat{p}_c$ local distribution estimates.

This view on horizontal FL has an appealing positive side effect: Aggregating model parameters can lead to divergence during training if the client's data distributions significantly differ. Since we aggregate distributions in mixtures, we naturally can handle heterogeneous client distributions. Also, since clients can train models independently, the communication cost of the training is minimized.

Product Nodes & Vertical FL. In vertical FL, each client is assumed to hold a disjoint set of features, i.e., $\mathbf{X}_c \cap \mathbf{X}_{c'} = \emptyset$ for all $c, c' \in \mathcal{C}$. In contrast to horizontal FL, all clients hold different features belonging to the same sample instances. As in horizontal FL, there is a semantic connection between vertical FL and PCs. Product nodes in PCs compute a product distribution defined on a disjoint set of random variables. Thus, a product node separates the data along the feature dimension, corresponding to the vertical FL setting. However, a product node assumes the random variables of the child distributions to be independent of each other. Obviously, this is an unrealistic assumption for vertical FL, where features held by different clients might be statistically dependent. Assumption 2 can be exploited to capture such dependencies, and a mixture of products of independent clusters can be formed. See Sec. 3.3 for details.

Definition 2 (Vertical FL). *Assume a set of samples $\mathbf{D}_c \sim p_c$ on each data owner $c \in \mathcal{C}$, the existence of a joint distribution p adhering to Assumptions 1 and 2 and that $\mathbf{X}_c \cap \mathbf{X}_{c'} = \emptyset$ holds for all $c, c' \in \mathcal{C}$ s.t. $c \neq c'$. We define vertical FL as estimating a joint distribution $\hat{p}$ s.t. $d(p, \hat{p})$ is minimal and $\int_{\mathbf{X} \setminus \mathbf{X}_c} \hat{p}(x) = \hat{p}_c(x)$ for all $x \in \mathcal{X}$ where d is a distance metric and $\hat{p}_c$ are estimates of client distributions.*

PCs & Hybrid FL. Given Defs. 1 and 2, hybrid FL is a combination of both. In terms of PC semantics, this amounts to building a hierarchy of fusing marginals and learning mixtures. Provided with these probabilistic semantics, we can now formally bridge PCs and FL. In the following, we distinguish between clients $\mathcal{C}$ and servers $\mathcal{S}$ and define the set of machines participating in training as $\mathcal{N} = \mathcal{C} \cup \mathcal{S}$. Bringing everything together and abstracting from the probabilistic interpretation, we define **federated circuits** (FCs) as follows.

Definition 3 (Federated Circuits). *A **federated circuit** (FC) is a tuple $(\mathcal{G}, \psi_{\mathcal{G}}, \omega)$ where $\mathcal{G} = (V, E)$ is a rooted, Directed Acyclic Graph (DAG), $\psi_{\mathcal{G}} : V \rightarrow \mathcal{N}$ assigns each $\mathsf{N} \in V$ to a client/server $n \in \mathcal{N}$ based on the structure of $\mathcal{G}$ and $\omega : V \rightarrow O$ assigns an operation $o \in O$ to each node $\mathsf{N} \in V$ where $o : dom(\mathrm{ch}(\mathsf{N})) \rightarrow dom(\mathsf{N})$ computes the value of N given the values of the children of N.*

FCs extend the definition of PCs in the sense that FCs represent a computational graph $\mathcal{G} = (V, E)$ distributed over multiple machines where arbitrary operations can be performed in each node $\mathsf{N} \in V$. Note that the FC's graph structure is not only a mere computation graph, but also defines the information flow among server(s) and clients, which we refer to as the communication network. Also, FCs are not restricted to the probabilistic interpretation presented above. For example, parameterizing leaves by decision trees and introducing a node N that performs averaging yields a bagging model.

3.3 FEDERATED PROBABILISTIC CIRCUITS

Let us now dive deeper into the probabilistic interpretation of FCs. To that end, we present a concrete instantiation of FCs leveraging Probabilistic Circuits (PCs) as leaf models, resulting in federated PCs (FedPCs). Following the probabilistic interpretation from Sec. 3.2, we align the PC structure with the communication network structure to form a federated PC.

Definition 4 (Federated PC). *A Federated PC (FedPC) is a FC where each leaf node C is a density estimator and each node N s.t. $\mathrm{ch}(\mathsf{N}) \neq \emptyset$ is either a sum node (S) or a product node (P).*

Note that only the client nodes C hold a dataset and we only demand the clients to be parameterized by a density estimator. In order for FedPCs to be computationally efficient, these density estimators should be tractable. In the following, we parameterize the leaf nodes C as PCs.

The assignment function ψ transforms the PC's computation graph into a distributed computation graph, thus inducing a communication network. This establishes a direct correspondence between PC semantics (computation graph) and the communication network structure in FedPCs. Inference is performed as usual in PCs by propagating likelihood values from the leaf nodes to the root node. The only difference is that the result of a node N has to be sent to its parent(s) $\mathbf{pa}(\mathsf{N})$ over the communication network if $\psi(\mathsf{N}) \neq \psi(\mathsf{N}')$ holds for $\mathsf{N}' \in \mathbf{pa}(\mathsf{N})$.

Training FedPCs requires adapting the regular training procedure for PCs because in FL, clients cannot access other clients' data. For example, training with Expectation Maximization (EM) requires access to the same samples for all clients, which is incompatible with horizontal and hybrid FL. Similarly, LearnSPN Gens and Domingos [2013] requires access to all features due to independence tests performed during training. To solve this, we propose a *one-pass* training procedure for FedPCs.

One-Pass Training. Our one-pass learning algorithm learns the structure and parameters of FedPCs such that local models can be trained independently (Algo. 1, Fig. 2). Before

Algorithm 1: One-Pass Training

```
Data: Clients $\mathcal{C}$, features $\mathbf{X}$, cluster size $K$, FedPC
Result: Trained fedPC
1  Set $\mathbf{M} = \mathbf{0}^{|\mathcal{C}|\times|\mathbf{X}|}$ and map = [];
   /* mark features per client */
2  $\mathbf{M}_{i,j} = 1$ if $X^{(j)}$ on client $i$;
3  for $j, \mathbf{u}$ in enum. of distinct columns $\mathcal{U}$ do
4      $\mathbf{S}^{(j)} = \{i : i \in \{1, \dots, |\mathbf{X}| \wedge \text{all}(\mathbf{u} == \mathbf{M}_{:,i})\}\}$;
       /* Group clients per feature */
5      $O_{\mathbf{S}^{(j)}} = \text{argwhere}(\mathbf{u} == 1)$;
6      map.append($\mathbf{S}^{(j)}$, $O_{\mathbf{S}^{(j)}}$);
7  sums = [];
8  for $\mathbf{S}^{(j)}$, $O_{\mathbf{S}^{(j)}}$ in map do
9      if $|O_{\mathbf{S}^{(j)}}| > 1$ then
10         s = fedPC.add_sum($\mathbf{S}^{(j)}$, $O_{\mathbf{S}^{(j)}}$);
11         sums.add(s)
12     else
13         client_clusters = cluster_local_data($O_{\mathbf{S}^{(j)}}$, $K$);
14 products = fedPC.add_products($P$);
   /* Construct random structure over
      non-shared features */
15 for prod in products do
16     prod.children.add(sums);
17     for client, clusters in client_clusters do
18         prod.children.add_rand_subset(clusters);
19 fedPC.add_mixture_over_products(products);
20 fedPC.train_clients();
21 fedPC.infer_weights();
22 return fedPC
```

training, all clients $c \in \mathcal{C}$ share their set of uniquely identifiable features/random variables $\mathbf{X}_c$ with a server, resulting in the feature set indicator matrix $\mathbf{M}^{|\mathcal{C}|\times|\mathbf{X}|}$ **(Lines 1-2)**. Feature identifiers can be names of features such as "account balance" and must correspond to the same random variable on all clients (thus uniquely identifiable). Then, the server divides the joint feature space $\mathbf{X}$ into disjoint subspaces $\mathbf{S}^{(j)}$. For this, we consider the set of distinct column vectors $\mathcal{U}$ of $\mathbf{M}$ where we denote distinct vectors as $\mathbf{u}$. Since each column of $\mathbf{M}$ indicates the set of clients a feature resides on, we can use each $\mathbf{u} \in \mathcal{U}$ to compute a set of features that are shared across the same set of clients. This results in $|\mathcal{U}|$ distinct feature sets, denoted $\{\mathbf{S}^{(1)}, \dots, \mathbf{S}^{(|\mathcal{U}|)}\}$. Each $O_{\mathbf{S}^{(j)}}$ denotes the set of clients that hold the features in $\mathbf{S}^{(j)}$. **(Lines 3-7)**. This procedure is illustrated in Fig. 2 (top).

Afterward, the FedPC structure is constructed as shown in Fig. 2 (bottom): First, we build a mixture (sum node) for each subspace $\mathbf{S}^{(j)}$ where $|O_{\mathbf{S}^{(j)}}| > 1$, i.e., more than one client holds $\mathbf{S}^{(j)}$ **(Lines 9-12)**. This enables each client to learn a PC over $\mathbf{S}^{(j)}$ independently. After that,$|O_{\mathbf{S}^{(j)}}| = 1$ holds for all remaining $\mathbf{S}^{(j)}$. Also, the scope of the sums nodes introduced in the FedPC share no features with any of the remaining $\mathbf{S}^{(j)}$ since the server divided the feature space into disjoint subspaces. Therefore, we introduce P product nodes to construct the remaining part of the FedPC. To this end, we divide the data of all subspaces $\mathbf{S}^{(j)}$ where $|O_{\mathbf{S}^{(j)}}| = 1$ holds into K clusters **(Line 14)**. Each client learns a dedicated PC for each cluster. To ensure that the FedPC spans the entire feature space of the clients, the children of product nodes are set as follows: Each sum node introduced in the FedPC becomes a child of each product node. Additionally, for each $\mathbf{S}^{(j)}$ where $|O_{\mathbf{S}^{(j)}}| = 1$ holds, we randomly select a PC learned over one of the K clusters s.t. the scope of each product node spans $\mathbf{X}$, and each PC representing a cluster is the child of at least one product node. Then, we build a mixture over all product nodes using a sum node **(Lines 15-20)**. Note that we seek to construct product nodes over independent clusters, which aligns with the maximum entropy principle (see App. C.1 for details). Once the FedPC is constructed, all client-sided PCs are learned. Since clients learn their PCs independently, each client can use an arbitrary learning algorithm (even different ones). As a last step, the network-sided parameters, i.e., the weights of network-sided sum nodes, of the FedPC are inferred **(Line 21-22)**. For each sum node S, the weight $\mathbf{w}_{\mathsf{S}}^{(i)}$ associated with the i-th child (i.e., distribution) of S is set to $\frac{\rho(\mathsf{N}_i)}{\sum_i \rho(\mathsf{N}_i)}$. Here, $\rho(\mathsf{N}_i) = \sum_{\mathsf{C}\in\text{ch}(\mathsf{N}_i)} |\mathbf{D}_{\mathsf{C}}|$ where $\mathbf{D}_{\mathsf{C}}$ is the dataset used to train the leaf C. Hence, the network-sided weights can be inferred without any forward or backward pass. Note that this approach reduces horizontal FL to learning a mixture of the client's data distributions and vertical FL to learning a mixture over P product nodes.

3.4 ANALYSIS OF COMMUNICATION EFFICIENCY

As a key requirement for efficient training when learning models at scale on partitioned data, we now analyze the communication efficiency of FedPCs.

Horizontal FL. Assume a client set $\mathcal{C}$, each holding a model with M parameters. Further, assume models are aggregated K times during training (K communication rounds). Then, model aggregation-based algorithms like FedAvg, commonly used in horizontal FL, send $\mathcal{O}(M \cdot |\mathcal{C}| \cdot K)$ messages over the network. K usually ranges between hundreds to even thousands, depending on the task and model complexity. Training FedPCs with one-pass training, in contrast, only requires $\mathcal{O}(|\mathcal{C}| \cdot (M + 1))$ messages over the network as models are learned locally and independently, followed by setting the parameters ($\mathcal{O}(|\mathcal{C}|)$ messages) of the sum nodes and aggregating the model on the server ($\mathcal{O}(M|\mathcal{C}|)$ messages).

Vertical FL. In vertical settings, SplitNN-like architectures are commonly used. Assume training a SplitNN architecture for E epochs that output a feature vector of size F for each sample of a dataset with S samples, vertically distributed

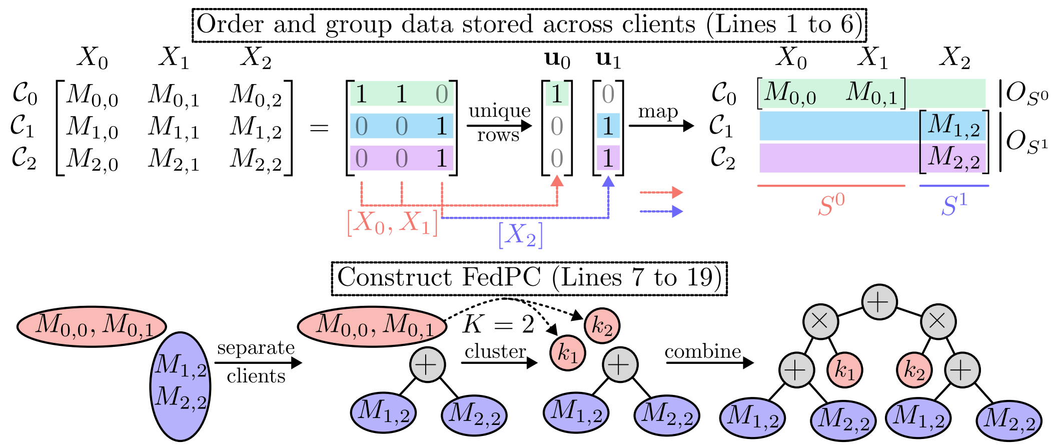

Figure 2: **One-Pass Training Visualized.** (Top) First, the matrix $\mathbf{M}$ is initialized, representing which features are held by which client. Feature subsets are constructed by considering distinct column vectors $\mathbf{u}$ of $\mathbf{M}$ that represent the same set of clients. This forms a mapping indicating which features are modeled as a mixture over clients. (Bottom) This mapping is utilized by forming mixtures over different clients sharing the same feature set via sum nodes. Features that are not shared over multiple clients will be clustered into K clusters (here $K = 2$). The FedPC is formed by creating product nodes containing all sum nodes from the previous steps and at least one of the K clusters. Lastly, the root node is inserted.

over clients $\mathcal{C}$. The training requires sending $\mathcal{O}(E \cdot |\mathcal{C}| \cdot F \cdot S)$ messages over the network. In contrast, with one-pass training of FedPCs, each client learns a dedicated PC with M parameters for each of the K clusters that are learned. The last layer of the FedPC is a mixture of P products of clusters. The mixture parameters are set after training each client's model. Aggregating the learned models and setting the network-sided mixture parameters requires $\mathcal{O}(K \cdot M \cdot |\mathcal{C}| + P)$ messages to be sent. If $(K \cdot M + \frac{P}{|\mathcal{C}|}) < (E \cdot F \cdot S)$ holds, training FedPCs is more communication efficient than training SplitNN-like architectures. In practice, this is likely to hold: The number of clusters is usually smaller than 100 while feature vectors can have hundreds of dimensions (i.e., $F > 100$). Further, models should have fewer parameters than samples in the dataset to ensure generalization (i.e., $M < S$). P can be set to an arbitrary value, depending on $|\mathcal{C}|$ and the data. App. D provides more details and an intuition on communication costs.

Hybrid FL. In hybrid FL, FedPCs are trained on several subspaces: Some exist on all or a subset of clients (denoted as R_s) and some are only available on one client (denoted as R_d). Further denote communication costs of FedPCs in horizontal FL and vertical FL as C_h and C_v, respectively. Since the training procedure in hybrid cases essentially performs horizontal FL on shared feature spaces and vertical FL on disjoint feature spaces, $\mathcal{O}(|R_s| \cdot C_h + |R_v| \cdot C_v)$ messages are sent over the network during training.

4 EXPERIMENTS

Our empirical evaluation corroborates that FedPCs can be leveraged to scale up PCs effectively via data and model partitioning. By performing horizontal, vertical and hybrid FL in one unified framework, we obtain high-performing models with the same or improved performance compared to prominent FL baselines.

We aim to answer the following questions: **(Q1)** Can FedPCs decrease the required training time and successfully learn a joint distribution over distributed data? **(Q2)** Do FedPCs effectively scale up PCs, thus yielding more expressive models? **(Q3)** How do FCs with different parameterizations perform on classification tasks compared to existing FL methods? **(Q4)** How does our one-pass learning algorithm compare to training with the EM algorithm?

Experimental Setup. To see if FedPCs, an instantiation of FCs, successfully scale up PCs, we follow Liu et al. [2024] and perform density estimation on three large-scale, high-resolution image datasets: Imagenet, Imagenet32 (both 1.2M samples), and CelebA (200K samples). The datasets were partitioned over 2-16 clients horizontally. We compare FedPCs against Einets [Peharz et al., 2020a], PyJuice [Liu et al., 2024], Probabilistic Integral Circuits (PICs) [Gala et al., 2024], Inception PCs (IncPCs) [Wang and Van den Broeck, 2025], and Latent Variable Distillation (LVD) [Liu et al., 2023] on 5 different seeds.

To evaluate FCs in FL scenarios, we selected three tabular datasets that cover various application domains and data regimes present in the real world: one credit fraud dataset ($\sim$ 300K samples), a medical dataset (breast cancer detection; $<$ 1000 samples), and the popular Income dataset ($>$ 1M samples). The selected datasets for FL cover low-data, medium-data, and large-data regimes (see App. E for more details). Both balanced (breast cancer) and imbalanced (income, credit) datasets are included in our evaluation. We

	Log-Likelihood				Relative Runtime			
	cent.	horizontal	vertical	hybrid	cent.	horizontal	vertical	hybrid
MNIST	3352±3.5	3350±3.2	3351±3.8	3349±3.7	1.0	**0.07±0.01**	0.13±0.01	0.13±0.02
Income	−11.5±0.1	−11.4±3.5	−11.9±3.3	−12.0±1.5	1.0	**0.17±0.02**	0.236±0.01	0.21±0.02
Cancer	−38.9±0.3	−38.5±1.1	−38.6±0.5	−38.7±1.5	1.0	**0.21±0.07**	0.35±0.05	0.35±0.1
Credit	−12.8±1.0	−13.1±0.5	−12.5±2.3	−12.5±1.3	1.0	0.42±0.05	**0.31±0.09**	0.40±0.13

Table 1: **FedPCs speed up training while retaining model performance.** We trained PCs in a centralized setting (cent.) and in all FL settings (using FedPCs) on different datasets and the same structure learning algorithm. We find that FedPCs tremendously speed up training while there is no reduction in log-likelihood. This demonstrates that PCs can be learned in federated settings (for MNIST, log densities are reported). We report relative runtime where centralized runtime is 1.0.

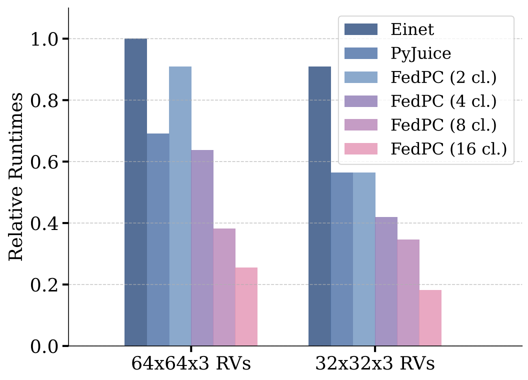

Figure 3: **FedPCs speed up training** on large-scale image data (64x64 and 32x32 RGB images) due to parallel training on separate data subsets.

	CelebA	Imagenet32	Imagenet64
EiNet	4.93 ± 0.09	5.35 ± 0.06	5.38 ± 0.06
PyJuice	4.30 ± 0.03	5.19 ± 0.01	4.95 ± 0.03
PICs	4.11 ± 0.03	4.46 ± 0.02	4.42 ± 0.03
IncPC*	-	4.19	3.90
LVD	4.02 ± 0.02	4.39 ± 0.04	4.12 ± 0.03
FedPC (2 cl.)	4.14 ± 0.07	4.50 ± 0.03	3.84 ± 0.03
FedPC (4 cl.)	4.10 ± 0.07	4.34 ± 0.04	3.69 ± 0.04
FedPC (8 cl.)	3.98 ± 0.06	4.29 ± 0.03	3.61 ± 0.04
FedPC (16 cl.)	**3.87 ± 0.04**	**4.18 ± 0.04**	**3.54 ± 0.06**

Table 2: **FedPCs outperform EiNets and PyJuice on density estimation tasks.** FedPCs achieve better results on density estimation tasks on CelebA, Imagenet32, and Imagenet because they can learn large models distributed across multiple machines. Results reported in bits per dimension (bpd) (lower is better). Values for baselines marked with * were taken from the respective papers (code not available). Best in **bold**, 2nd best underlined.

selected tabular datasets as they are well suited to investigate FCs in horizontal, vertical, and hybrid settings and represent various real-world applications. We compare FCs to multiple strong and widely used baselines. As a neural network architecture parameterization, we use TabNet [Arik and Pfister, 2020] which is tailored to tabular datasets. We train the networks with the widely used FedAvg (horizontal FL) and SplitNN (vertical FL) frameworks. Additionally, we compare FCs to FedTree [Li et al., 2023b] since tree models excel at tabular datasets. For details, see App. E.

(Q1) FedPCs learn joint distributions over partitioned data in less time. First, we validate that FedPCs correctly and efficiently perform density estimation on partitioned datasets distributed over multiple clients. To this end, multiple datasets were distributed over a set of clients corresponding to horizontal (5 clients), vertical (2 clients), and hybrid FL (2 clients). To demonstrate that FedPCs are also robust against label shifts, a common regime in FL, each client received data from only a subset of classes in the horizontal case, and local PCs were learned over the client samples. In the vertical case, we split data s.t. feature spaces of clients are disjoint, but each client holds the same samples. In hybrid settings, data was distributed s.t. both feature- and sample-spaces among clients have overlaps (but no full overlap). For all tabular datasets, the leaves of the FedPC were parameterized with MSPNs [Molina et al., 2018], a member of the PC model family capable of performing density estimation on mixed data domains (i.e., continuous and discrete random variables). We chose MSPNs as the centralized models, which were learned using LEARNSPN, a recursive greedy structure learning algorithm for SPNs [Gens and Domingos, 2013]. For MNIST, EiNets with Gaussian densities were used as PC instantiations in all settings. Note that FedPCs were chosen to approximately match the size of centralized models, i.e., no model upscaling was performed.

Tab. 1 compares log-likelihoods and relative runtime of centralized PC training on the full datasets with log-likelihood scores and relative runtimes achieved by FedPC in different FL settings. FedPCs achieve the same log-likelihoods as centralized PCs on tabular datasets while being tremendously faster in training. Hence, we answer **(Q1)** affirmatively.

(Q2) FedPCs effectively scale up PCs. To examine whether FedPCs can be leveraged to scale up PCs effectively, we compared FedPCs to EiNets, PyJuice, PICs, IncPCs, and LVD on CelebA, Imagenet32, and Imagenet. Einets, PyJuice, and FedPCs were parameterized with the Poon-Domingos (PD) architecture. PICs define the architecture based on quadrature integration points, while IncPC and

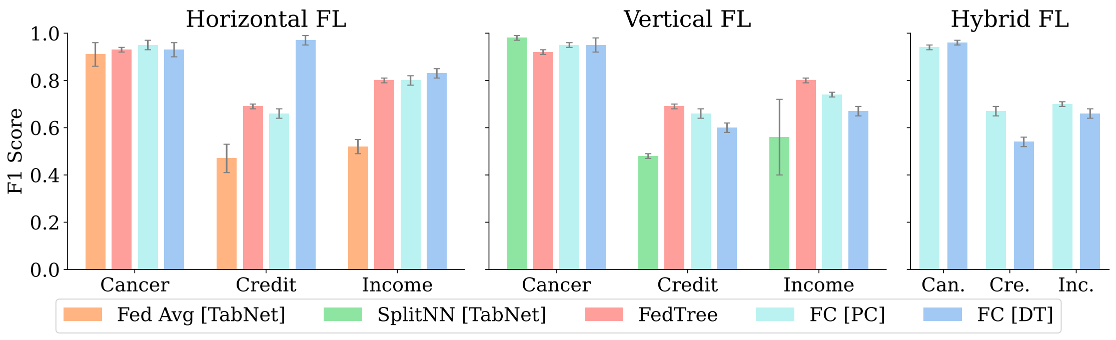

Figure 4: **FCs are competitive to prominent FL methods in all settings.** FCs achieve competitive performance on various classification tasks compared to prominent horizontal/vertical FL baselines. FCs also handle the more challenging setting of hybrid FL without performance drops. We reported the F1 score (higher is better).

LVD use the HCTL structure. FedPCs used EiNets in the clients, and data was distributed among 2, 4, 8, and 16 clients. The FedPC model and baseline models were selected to ensure that each fits within a single GPU (see App. E for system details). Einets and FedPCs were parameterized with Gaussian leave distributions, while the other baselines were parameterized with Categorical distributions. The parameterizations were chosen based on empirical observations; for Einets and FedPCs, Gaussians worked better than categorical distributions.

For FedPC training, the images were distributed horizontally at random, s.t. each client holds approximately equally large subsets. The leaves and all baselines were trained with EM. In Tab. 2, we show bits per dimension (bpd) achieved by FedPCs and our baselines on the test set. It can be seen that with an increasing number of participating clients and, thus, a larger model, the density estimation performance also increases on all three datasets. We posit that this is because larger models exhibit higher expressivity, allowing them to capture statistical characteristics of the data better than smaller models. Also, lower bpd scores achieved on the test set by larger models indicate that no overfitting appeared due to more model parameters. However, note that more exhaustive scaling will likely lead to overfitting. Finding the optimal model size/number of clients in a principled way is beyond the scope of this work and is left for future endeavors. Besides better modeling performance, a larger number of clients reduces training time significantly (see Fig. 3). FedPCs thus efficiently scale tractable probabilistic models to large datasets.

(Q3) FCs achieve state of the art classification results in FL. FCs can be parameterized with different models in the leaves. We examine two parameterizations to solve a federated classification task on three tabular datasets. First, we use the FedPC (FC [PC]) from **(Q1)**, which can be used to solve discriminative tasks leveraging tractable computation of conditionals in PCs. The second FC parameterization we examine is decision trees (FC [DT]), representing an instantiation of a bagging model. To see how FCs perform in federated classification tasks, we compare FCs to well-known methods for horizontal FL and vertical FL. The experiments were conducted on tabular datasets covering various real-world application domains and distribution properties. We employ TabNet and FedTree as strong baselines. In the horizontal FL setting, TabNet was trained using FedAvg; in the vertical FL setting, it was trained in a SplitNN fashion [Ceballos et al., 2020]. The results were compared against our one-pass training. FCs yield comparable or better results than our baselines on all datasets (see Fig 4; App. F) while being significantly more flexible compared to the baselines.

	EM	one-pass
Synth. Data	-53.6 ± 1.3	-53.2 ± 1.2
Income	-18.5 ± 0.1	-18.0 ± 0.5
Breast-Cancer	-52.3 ± 0.2	-55.7 ± 0.2
Credit	-26.7 ± 1.2	-28.3 ± 0.4

Table 3: **One-pass training retains performance.** We trained the same FedPC architecture on various datasets using EM and one-pass training in a vertical setting. The average log-likelihood value of the hold-out test set across 10 runs is reported.

(Q4) One-pass training retains performance. To see how the proposed one-pass training compares to training PCs with standard optimization algorithms such as EM, we define an FL setup where data exchange is allowed. This is necessary as we have to train the PC and FedPC architecture with EM to compare to our one-pass procedure. We used RAT-SPNs [Peharz et al., 2020b] as leaf parameterizations of the FedPC. Then, we trained a FedPC using standard EM (i.e., data exchange was allowed) and another FedPC with the same FedPC architecture on a vertically split dataset using our one-pass procedure. We report the final average log-likelihood of the test dataset, both for EM training and one-pass training (see Tab. 3). It can be seen that there is no significant decrease in log-likelihood in any case. Hence, our results indicate that one-pass training is preferable since it yields comparable model performance while being more communication efficient.

5 CONCLUSION

In this work, we introduced federated circuits that hinge on an inherent connection between PCs and FL. We demonstrated that both the training speed and expressivity of PCs can be increased by learning PCs on scale across partitioned data. Since our framework allows for the integration of various types of density estimators, other models and advances of PCs and other fields can be integrated seamlessly and can therefore be scaled up straightforwardly.

Limitations and Future Work. While our experiments showed that scaling PCs can considerably improve training speed and performance, scaling to such large-scale models requires sufficient computational resources. For future work, investigating other parametrizations for FCs beyond PCs is promising. Additionally, it is interesting how the probabilistic framework for hybrid FL could also benefit more traditional FL applications, apart from scaling PCs.

ACKNOWLEDGEMENTS

This work is supported by the Hessian Ministry of Higher Education, Research, Science and the Arts (HMWK; "The Third Wave of AI"). Further, this work benefited from the National High-Performance Computing project for Computational Engineering Sciences (NHR4CES).

The Eindhoven University of Technology authors received support from their Department of Mathematics and Computer Science and the Eindhoven Artificial Intelligence Systems Institute.

References

Sercan O. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning, 2020.

Ethan Caballero, Kshitij Gupta, Irina Rish, and David Krueger. Broken neural scaling laws. In *The Eleventh International Conference on Learning Representations*, 2023.

Iker Ceballos, Vivek Sharma, Eduardo Mugica, Abhishek Singh, Alberto Roman, Praneeth Vepakomma, and Ramesh Raskar. Splitnn-driven vertical partitioning. *CoRR*, abs/2008.04137, 2020.

Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. VAFL: a method of vertical asynchronous federated learning. *CoRR*, abs/2007.06081, 2020.

K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36:87–98, 2021.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020.

Gennaro Gala, Cassio de Campos, Antonio Vergari, and Erik Quaeghebeur. Scaling continuous latent variable models as probabilistic integral circuits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 873–880. PMLR, 2013.

Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Mime: Mimicking centralized stochastic algorithms in federated learning. *arXiv preprint arXiv:2008.03606*, 2020a.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 2020b.

Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Nicolas Kourtellis, Gianmarco De Francisci Morales, Albert Bifet, and Arinto Murdopo. Vht: Vertical hoeffding tree. In *2016 IEEE International Conference on Big Data (Big Data)*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35:3347–3366, 2023a.

Qinbin Li, Zhaomin Wu, Yanzheng Cai, Ching Man Yung, Tianyuan Fu, Bingsheng He, et al. Fedtree: A federated learning system for trees. *Proceedings of Machine Learning and Systems*, 5, 2023b.

Qinbin Li, Chulin Xie, Xiaojun Xu, Xiaoyuan Liu, Ce Zhang, Bo Li, Bingsheng He, and Dawn Song. Effective and efficient federated tree learning on hybrid data. In *The Twelfth International Conference on Learning Representations*, 2024.

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

Tianbo Li, Tianze Luo, Yiping Ke, and Sinno Jialin Pan. Mitigating performance saturation in neural marked point processes: Architectures and loss functions. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 986–994. Association for Computing Machinery, 2021.

Anji Liu, Honghua Zhang, and Guy Van den Broeck. Scaling up probabilistic circuits by latent variable distillation. In *The Eleventh International Conference on Learning Representations*, 2022.

Anji Liu, Honghua Zhang, and Guy Van den Broeck. Scaling up probabilistic circuits by latent variable distillation. In *The Eleventh International Conference on Learning Representations*, 2023.

Anji Liu, Kareem Ahmed, and Guy Van den Broeck. Scaling tractable probabilistic circuits: A systems perspective, 2024.

Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. A communication efficient vertical federated learning framework. *CoRR*, abs/1912.11187, 2019.

Lorenzo Loconte, Aleksanteri Mikulus Sladek, Stefan Mengel, Martin Trapp, Arno Solin, Nicolas Gillis, and Antonio Vergari. Subtractive mixture models via squaring: Representation and learning. In *The Twelfth International Conference on Learning Representations*, 2024.

H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pages 15–17, 1985.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752. PMLR, 2015a.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On Theoretical Properties of Sum-Product Networks. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38, pages 744–752. PMLR, 2015b.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van Den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 7563–7574. PMLR, 2020a.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020b.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. page 337–346. AUAI Press, 2011a.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011b.

Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127, 2018.

Raquel Sánchez-Cauce, Iago París, and Francisco Javier Díez. Sum-product networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44 (7):3821–3839, 2021.

Andy Shih, Dorsa Sadigh, and Stefano Ermon. Hyperspns: Compact and expressive probabilistic circuits. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, december 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

Benjie Wang and Guy Van den Broeck. On the relationship between monotone and squared probabilistic circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 21026–21034, 2025.

Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. A survey on federated learning: challenges and applications. *Int. J. Mach. Learn. & Cyber.*, pages 513—-535, 2023.

Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. Privacy preserving vertical federated learning for tree-based models. *arXiv preprint arXiv:2008.06170*, 2020.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Xinwei Zhang, Wotao Yin, Mingyi Hong, and Tianyi Chen. Hybrid federated learning: Algorithms and implementation. *CoRR*, abs/2012.12420, 2020.

Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed variational inference for sum-product networks. In *International Conference on Machine Learning (ICML)*, volume 48, pages 1310–1318, 2016.

A FEDERATED LEARNING: GOALS AND APPROACHES

Federated Learning (FL) aims to collaboratively learn a machine learning model across a set of clients, each holding a different dataset without sharing the data with a central server (i.e., keeping data private) [Li et al., 2020]. The server only orchestrates the training. In FL, one distinguishes horizontal, vertical, and hybrid settings. In horizontal FL, all clients share the same features but hold different samples; in vertical FL, all clients hold the same samples but different features; and in hybrid FL, a mix of horizontal and vertical FL is assumed. Most FL approaches focus on horizontal scenarios with neural networks as models. The majority of approaches tackle horizontal FL by averaging the model on a central server regularly, thereby integrating distribution information via model merging [Li et al., 2020, McMahan et al., 2016, Sahu et al., 2018]. This averaging, however, requires that each client trains the exact same neural architecture. For example, if two clients would like to train an image classifier and one client uses a convolutional neural network (CNN; [LeCun et al., 1998]) while the other uses a Transformer model [Vaswani et al., 2017], it is not possible to average the model parameters due to the different semantics and structure of CNNs and Transformers. In vertical and hybrid FL, the most prominent approaches use neural networks by sharing learned representations instead of the data directly, or by employing tree models that are synchronized by merging learned decision rules [Shih et al., 2021, Li et al., 2023b]. Sharing learned representations, however, can get costly quickly with respect to communication efficiency, leading to long training. Merging tree models, on the other hand, is more communication efficient, but often requires additional model assumptions or complex learning algorithms, restricting the expressivity of learned models.

In contrast, Federated Circuits (FC) neither require merging of model parameters, nor do they require representations being shared or complex learning algorithms being employed. By exploiting the structural properties of probabilistic circuits (PCs) and mapping PC semantics to FL, we obtain an elegant learning framework that can handle horizontal, vertical, and hybrid FL in one framework effectively while being communication efficient.

B DISCUSSION ON ASSUMPTIONS

As a preliminary to FCs, we introduced two assumptions that allowed us to construct the FC framework. Here, we provide some more background on these assumptions. For clarity, let us state the assumptions again.

Assumption 1 (Mixture Marginals). There exists a joint distribution p such that the relation $\int_{\mathbf{X}\setminus\mathbf{X}_S} p(x) = \sum_{l\in L} q(L = l) \cdot p_S(x|L = l)$ holds for all $x \in \mathcal{X}$. Here, $\mathbf{X}_S \subseteq \mathbf{X}$ is a subset of the union of client random variables $\mathbf{X} = \cup_{c\in\mathcal{C}}\mathbf{X}_c$. Further, $\mathcal{X} = \bigtimes_{i\in\{1,\dots,|\mathbf{X}|\}} \mathcal{X}_i$ is the support of $\mathbf{X}$, each p_S is defined over $\mathbf{X}_S \subseteq \mathbf{X}$ and q is a prior over a latent L.

Assumption 2 (Cluster Independence). Given disjoint sets of random variables $\mathbf{X}_1, \cdots, \mathbf{X}_n$ and a joint distribution $p(\mathbf{X}_1, \cdots, \mathbf{X}_n)$, assume that a latent L can be introduced s.t. the joint can be represented as $p(\mathbf{X}_1, \cdots, \mathbf{X}_n) = \sum_l q(L = l) \prod_{i=1}^n p(\mathbf{X}_i|L = l)$ where q is a prior distribution over the latent L.

As discussed in the main paper, Assumption 1 ensures that the data that resides on all participating clients is sufficient to learn $p(\mathbf{X})$, at least in the limit of infinite samples available. However, this only covers the federated learning perspective of this assumption. There is also a PC perspective on this assumption. For this, let us introduce the induced tree representation of PCs from [Zhao et al., 2016]:

Definition 5. *Induced Trees [Zhao et al., 2016]. Given a complete and decomposable PC s over $\mathbf{X} = \{X_1, \dots, X_n\}$, $\mathcal{T} = (\mathcal{T}_V, \mathcal{T}_E)$ is called an induced tree PC from s if*

1. $\mathsf{N} \in \mathcal{T}_V$ *where* N *is the root of* s.
2. *for all sum nodes* $\mathsf{S} \in \mathcal{T}_V$*, exactly one child of* S *in* s *is in* $\mathcal{T}_V$*, and the corresponding edge is in* $\mathcal{T}_E$.
3. *for all product node* $\mathsf{P} \in \mathcal{T}_V$*, all children of* P *in* s *are in* $\mathcal{T}_V$*, and the corresponding edges in* $\mathcal{T}_E$.

We can use Def. 5 to represent decomposable and complete PCs as mixtures [Zhao et al., 2016].

Proposition 1 (Induced Tree Representation). *Let τ_s be the total number of induced trees in s. Then the output at the root of s can be written as $\sum_{t=1}^{\tau_s} \prod_{(k,j)\in\mathcal{T}_{tE}} w_{kj} \prod_{i=1}^n p_t(X_i = x_i)$, where $\mathcal{T}_t$ is the t-th unique induced tree of s and $p_t(X_i)$ is a univariate distribution over X_i in $\mathcal{T}_t$ as a leaf node.*

Using Prop. 1, we see that any decomposable and smooth PC can be represented as a mixture without any hierarchy, i.e., we can collapse the PC structure into a structure of depth one. Since marginalizing over a decomposable and smooth PC yields another decomposable and smooth PC again, and since the marginalized PC can be represented as an induced tree, Assumption 1 is a standard assumption in the PC literature.

Also, Assumption 2 can be viewed from a PC perspective. In popular structure learning algorithms such as LearnSPN Gens and Domingos [2013], a PC is learned by alternating data clustering with testing for independent subsets of features. Thus, the ultimate goal of algorithms like LearnSPN is to find clusters in which subsets of random variables are considered independent in order to maximize log-likelihood. Therefore, Assumption 2 is closely related to LearnSPN and, thus, a common assumption in PC modeling.

C PROOFS

In this section we give full proofs for our propositions in the paper.

C.1 FEDPCS AND PRINCIPLE OF MAXIMUM ENTROPY

Assumption 2 aligns with the principle of maximum entropy: we aim to find the joint distribution with maximum entropy *within* clusters while allowing for dependencies among clients' random variables and ensuring the marginals for each client are preserved. Although multiple joint distributions can preserve the marginals, non-maximal entropy solutions introduce additional assumptions or prior knowledge, limiting flexibility. By assuming independence of all variables within a cluster, we efficiently construct the maximum entropy distribution via a mixture of product distributions. For independent variables, the product distribution maximizes entropy, as can be shown by leveraging the joint and conditional differential entropy. Given random variables $\mathbf{X} = X_1, \ldots, X_n$ and a density p defined over support $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, the joint differential entropy is defined as:

$$h(\mathbf{X}) = \int_{\mathcal{X}} p(x_1, \ldots, x_n) \, \mathrm{log} p(x_1, \ldots, x_n) \tag{1}$$

The conditional differential entropy for two sets of random variables $\mathbf{X}$ and $\mathbf{Y}$ and a joint distribution $p(\mathbf{X}, \mathbf{Y})$ defined over support $\mathcal{X} \times \mathcal{Y}$ is defined analogously:

$$h(\mathbf{X}|\mathbf{Y}) = \int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \, \mathrm{log} p(\mathbf{x}|\mathbf{y}) \tag{2}$$

Given two sets of random variables $\mathbf{X}, \mathbf{Y}$ with densities $p(\mathbf{X})$ and $p(\mathbf{Y})$ and support $\mathcal{X}, \mathcal{Y}$ respectively, the joint $p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{X}) \cdot p(\mathbf{Y})$ is the maximum entropy distribution if $\mathbf{X}$ and $\mathbf{Y}$ are mutually independent.

Proof. We consider the two cases that $\mathbf{X}$ and $\mathbf{Y}$ are mutually independent and that they are not mutually independent. The joint entropy can be written as $h(\mathbf{X}, \mathbf{Y}) = h(\mathbf{X}|\mathbf{Y}) + h(\mathbf{Y})$. In the case of mutual independence, this reduces to $h(\mathbf{X}, \mathbf{Y}) = h(\mathbf{X}) + h(\mathbf{Y})$. Hence it has to be shown that $h(\mathbf{X}|\mathbf{Y}) < h(\mathbf{X})$ holds if $\mathbf{X}$ and $\mathbf{Y}$ are not mutually independent:

$$\begin{aligned}
&h(\mathbf{X}|\mathbf{Y}) < h(\mathbf{X}) \\
&\equiv -\int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} p(\mathbf{x}|\mathbf{y}) < -\int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} p(\mathbf{x}) \\
&\equiv -\left(\int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} p(\mathbf{x}|\mathbf{y}) - \int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} p(\mathbf{x})\right) < 0 \\
&\equiv -\left(\int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} \frac{p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})}\right) < 0
\end{aligned}$$

Since $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}$ holds where $\perp\!\!\!\perp$ means mutual independence, $\frac{p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})} \neq 1$ at least for some $\mathbf{x}, \mathbf{y}$. Since the mutual independence $I(\mathbf{X}, \mathbf{Y}) = \int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x}) \cdot p(\mathbf{y})}$ can be represented as $I(\mathbf{X}, \mathbf{Y}) = h(\mathbf{X}) - h(\mathbf{X}|\mathbf{Y})$, $I(\mathbf{X}, \mathbf{Y}) \geq 0$ holds and $-\left(\int_{\mathcal{X},\mathcal{Y}} p(\mathbf{x}, \mathbf{y}) \mathrm{log} \frac{p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})}\right) = h(\mathbf{X}|\mathbf{Y}) - h(\mathbf{X})$ it follows that $h(\mathbf{X}) > h(\mathbf{X}|\mathbf{Y})$.

$\square$

D COMMUNICATION EFFICIENCY

Communication efficiency is a critical property when it comes to learning models across multiple machines, as it is done in FL. Here, in addition to our theoretical results, we more intuitively provide further details on the communication efficiency of FCs. For that, we plot the communication cost in Megabytes (MB) required to train a FedPC vs. FedAvg/SplitNN in horizontal/vertical FL settings with datasets of different sizes (1M and 100M samples). Regardless of the number of samples in the dataset, FedPCs are more communication efficient compared to our baselines in both horizontal and vertical settings (see Fig. 5).

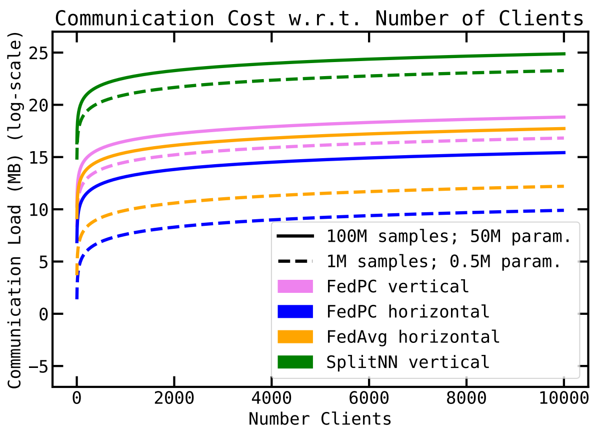

Figure 5: **FedPCs are communication-efficient.** We compare communication cost in Megabytes (MB) sent over the network during one full training of a model (0.5M/50M parameters) on a dataset (1M/100M samples) using results from Section 3.4. Results are shown on log-scale. It can be seen that FedPCs significantly reduce communication cost of training.

E EXPERIMENTAL DETAILS

In this section, we provide a detailed overview of our experimental setup. In all experiments, we consider a cross-silo FL setup with one orchestrating server. Note that in the case of FedPCs, the server does not perform model averaging. Instead, it orchestrates the local model training and construction of the FedPC model.

E.1 DATASETS

The following describes the datasets used in our experiments. If not stated differently, the datasets were distributed across clients as follows:

In horizontal cases, we either split samples randomly across clients (done for all binary classification tasks) or we distribute a subset of the dataset corresponding to a certain label (e.g. the 0 in MNIST) to one client.

In vertical cases, we split tabular datasets randomly along the feature-dimension, i.e. each client gets all samples but a random subset of features assigned. For image data, we split the images into non-overlapping patches which were then distributed to the clients.

In hybrid cases, we split tabular datasets along both, the feature and the sample-dimension. We do this s.t. at least two clients have at least one randomly chosen feature in commeon (but hold different samples thereof). For image data, we split images into overlapping patches, sample a subset of the dataset and assign the resulting subsets to clients.

Income Dataset. We used the Income dataset from `https://www.kaggle.com/datasets/wenruliu/adult-income-dataset`. This dataset represents a binary classification problem with 14 features and approximate 450K samples in the train and 900 samples in the test set. We encoded discrete variables to numerical values using TargetEncoder from sklearn. Additionally, missing values were imputed using the median of the corresponding feature. Further we standardized all features.

Breast Cancer Dataset. We used the Breast Cancer dataset from `https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data`. It represents a binary classification problem with 31 features and 570 samples. We split the dataset into 450 training samples and 120 test samples. We standardized all features for training.

Credit Dataset. We used the Give Me Some Credit dataset from `https://www.kaggle.com/c/GiveMeSomeCredit`. The dataset represents a binary classification task with 10 features, 1.5M training samples and 100K test samples. We encoded discrete variables to numerical values using TargetEncoder from sklearn. Additionally, missing values were imputed using the median of the corresponding feature. Further we standardized all features.

MNIST. We used the MNIST dataset provided by pytorch. It contains 70K hand-written digits between 0 and 9 as 28x28 images (60K train, 10K test). We standardized all features as preprocessing.

Imagenet/Imagenet32. We used the Imagenet dataset provided by pytorch. It consists of about 1.2M images showing objects of 1000 classes. The images come in different resolutions; we resized each image to 64x64 (Imagenet) and 32x32 (Imagenet32) pixels, applied center cropping, and standardized all features as preprocessing. We distributed samples randomly across clients as a simple dataset partioning.

E.2 DISCRETIZATION

In our experimental setup, FCs and Einets were parameterized with Gaussian leaves and fitted on RGB image data. Since image data is discrete (takes integer values from 0-255) and Gaussians are defined over a continuous domain and thus define a probability *density* rather than a probability *mass* function, we have to discretize the Gaussian leaves to obtain the probability for a given image $\mathbf{x}$. Therefore, we construct 255 buckets, discretizing a Gaussian with parameters μ and σ by computing the probability mass as $p(x) = \Phi(\frac{x-\mu+\frac{1}{255}}{\sigma}) - \Phi(\frac{x-\mu}{\sigma})$. The computation graph will remain fixed since the probabilistic semantics of PCs hold for densities and probability mass functions.

E.3 TRAINING & HYPERPARAMETERS

PyJuice. For PyJuice we follow the training setup of [Liu et al., 2024] and train the models on randomly cropped 32x32 patches. For Imagenet32, this corresponds to the full image, while for CelebA and Imagenet, this corresponds to a random 32x32 block of the image. We used EM as an optimization procedure.

FCs and Einets. For FCs and Einets, we followed the setup from [Peharz et al., 2020a] and trained all models on the full image (i.e., 32x32 for Imagenet32 and 64x64 for Celebi/Imagenet). We used EM as an optimization procedure.

For PICs and LVD, we followed the experimental setup described in the corresponding papers. The code of IncPCs was not publicly available.

We ran all experiments with 5 different seeds (0-4). The following tables show the settings of all relevant hyperparameters for each dataset and FL setting.

FL-Setting	Dataset	Structure	Threshold	min_num_instances	glueing
horizontal	Income	learned	0.3	200	-
	Credit	learned	0.5	200	-
	Cancer	learned	0.4	300	-
vertical	Income	learned	0.4	100	combinatorial
	Credit	learned	0.5	50	combinatorial
	Cancer	learned	0.4	300	combinatorial
hybrid	Income	learned	0.4	100	combinatorial
	Credit	learned	0.5	50	combinatorial
	Cancer	learned	0.4	300	combinatorial

Table 4: Hyperparameters used in our experiments for all tabular datasets.

	MNIST	Imagenet32	Imagenet	CelebA
num_epochs	5	10	10	10
batch_size	64	64	64	64
online_em_frequency	5	10	50	10
online_em_stepsize	0.1	0.25	0.5	0.25
Structure	poon-domingos	poon-domingos	poon-domingos	poon-domingos
pd_num_pieces	8	8	8	8
K	10	40	40	40
Leaf Distribution	Gaussian	Gaussian	Gaussian	Gaussian
min_var	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
max_var	0.25	0.25	0.25	0.25

Table 5: Hyperparameters used in our experiments for image datasets.

E.4 MODEL DETAILS

For our density estimation tasks on image data, we employ the PD architecture [Poon and Domingos, 2011b]. In horizontal settings, the region graph will be the same for all clients since its construction is deterministic and only based on the number of features to be modelled. In vertical and hybrid settings, the region graphs can vary among clients, depending on how the features are distributed among clients.

Further, the model size varies with the number of clients: The more clients participate in the training, the larger the model can be. The model sizes used in our experiments can be seen in Tab. 6.

	Imagenet32	Imagenet	CelebA
cent.	152M	387M	387M
2 clients	304M	775M	775M
4 clients	608B	1.5B	1.5B
8 clients	1.2B	3.1B	3.1B
16 clients	2.4B	6.2B	6.2B

Table 6: Number of parameters of our models, depending on the dataset and the number of clients.

E.5 HARDWARE

All experiments were conducted on Nvidia DGX machines with Nvidia A100 (40GB) GPUs, AMD EPYC 7742 64-Core Processor and 2TiB of RAM.

F FURTHER RESULTS

Here, we provide further experimental details on FCs.

Further Discussion of Density Estimation Results. There are two possible ways of improving the modeling performance of PCs. One way is to improve the modeling itself (e.g., by defining different architectures, allowing for negative or complex mixture weights, etc.). Since PCs are a special kind of neural network, another way of improving modeling performance is to exploit neural scaling laws [Caballero et al., 2023, Li et al., 2021].

Recent works on neural scaling laws suggest that increasing the model size from a small to a mid-sized model yields significant gains in terms of model performance while adding parameters to an already large model has minor effects [Caballero et al., 2023, Li et al., 2021]. This means that a saturation effect occurs when increasing the number of model parameters. Thus, in our setting, observing a larger gain when scaling from 1 to 2 clients and less improvement when going from e.g., 8 to 16 clients is well aligned with recent findings.

FL Classification Results. We compare FCs to several baselines in horizontal, vertical, and hybrid FL. In horizontal FL, we compare against FedAvg (using TabNet [Arik and Pfister, 2020]) and FedTree [Li et al., 2023b]; in vertical FL, we compare against SplitNN (also using TabNet) and FedTree. In hybrid FL, we compare different parameterizations of FCs (FedPCs and FCs parameterized with decision trees). We find that FCs are competitive or outperforming the selected baselines in all FL settings (see Tab. 7). This makes them a very flexible FL framework that still yields high-performing models.

		Cancer		Credit		Income	
		Acc.	F1	Acc.	F1	Acc.	F1
Horizontal FL	FedAvg [TabNet] (5 cl.)	0.92 ± 0.03	0.92 ± 0.03	0.71 ± 0.11	0.48 ± 0.04	0.68 ± 0.06	0.51 ± 0.03
	FedAvg [TabNet] (10 cl.)	0.92 ± 0.04	0.91 ± 0.05	0.56 ± 0.12	0.47 ± 0.06	0.64 ± 0.06	0.52 ± 0.03
	FedTree (5 cl.)	0.93 ± 0.01	0.92 ± 0.01	0.91 ± 0.01	0.63 ± 0.01	0.88 ± 0.01	0.82 ± 0.02
	FedTree (10 cl.)	0.94 ± 0.01	0.93 ± 0.01	0.92 ± 0.01	0.69 ± 0.01	0.87 ± 0.01	0.80 ± 0.01
	FC [PC] (5 cl.)	0.98 ± 0.01	0.98 ± 0.01	0.93 ± 0.02	0.68 ± 0.02	0.87 ± 0.02	0.80 ± 0.01
	FC [PC] (10 cl.)	0.95 ± 0.02	0.95 ± 0.02	0.93 ± 0.01	0.66 ± 0.02	0.87 ± 0.01	0.80 ± 0.02
	FC [DT] (5 cl.)	0.95 ± 0.03	0.93 ± 0.02	0.92 ± 0.01	0.67 ± 0.01	0.89 ± 0.01	0.83 ± 0.01
	FC [DT] (10 cl.)	0.95 ± 0.02	0.93 ± 0.03	0.92 ± 0.01	0.97 ± 0.02	0.89 ± 0.01	0.83 ± 0.02
	SplitNN [TabNet]	-	-	-	-	-	-
Vertical FL	SplitNN [TabNet] (2 cl.)	0.98 ± 0.01	0.98 ± 0.01	0.93 ± 0.01	0.48 ± 0.01	0.56 ± 0.25	0.42 ± 0.17
	SplitNN [TabNet] (3 cl.)	0.98 ± 0.01	0.98 ± 0.01	0.93 ± 0.01	0.48 ± 0.01	0.62 ± 0.20	0.56 ± 0.16
	FedTree (2 cl.)	0.94 ± 0.01	0.93 ± 0.01	0.92 ± 0.01	0.69 ± 0.02	0.87 ± 0.01	0.80 ± 0.01
	FedTree (3 cl.)	0.93 ± 0.01	0.92 ± 0.01	0.92 ± 0.01	0.69 ± 0.01	0.87 ± 0.01	0.80 ± 0.01
	FC [PC] (2 cl.)	0.96 ± 0.01	0.96 ± 0.01	0.92 ± 0.01	0.67 ± 0.01	0.84 ± 0.02	0.74 ± 0.01
	FC [PC] (3 cl.)	0.95 ± 0.01	0.95 ± 0.01	0.92 ± 0.01	0.66 ± 0.02	0.84 ± 0.01	0.74 ± 0.01
	FC [DT] (2 cl.)	0.96 ± 0.01	0.96 ± 0.02	0.93 ± 0.01	0.60 ± 0.02	0.83 ± 0.02	0.67 ± 0.02
	FC [DT] (3 cl.)	0.95 ± 0.01	0.95 ± 0.03	0.93 ± 0.01	0.60 ± 0.02	0.82 ± 0.02	0.67 ± 0.02
	FedAvg [TabNet]	-	-	-	-	-	-
Hybrid FL	FC [PC] (2 cl.)	0.94 ± 0.01	0.94 ± 0.01	0.92 ± 0.01	0.67 ± 0.01	0.82 ± 0.02	0.71 ± 0.01
	FC [PC] (3 cl.)	0.94 ± 0.01	0.94 ± 0.01	0.92 ± 0.01	0.67 ± 0.02	0.80 ± 0.01	0.70 ± 0.01
	FC [DT] (2 cl.)	0.96 ± 0.01	0.96 ± 0.02	0.93 ± 0.01	0.60 ± 0.02	0.82 ± 0.02	0.66 ± 0.02
	FC [DT] (3 cl.)	0.96 ± 0.01	0.96 ± 0.01	0.93 ± 0.01	0.54 ± 0.02	0.82 ± 0.02	0.66 ± 0.02
	FedAvg [TabNet]	-	-	-	-	-	-
	SplitNN [TabNet]	-	-	-	-	-	-
	FedTree	-	-	-	-	-	-

Table 7: **All Classification results of FL experiments.** Here, we show the detailed performances of FC, FedAvg, and SplitNN in all three FL settings. It can be seen that FCs, while being much more flexible than our baselines, still achieve competitive or better results on various classification tasks.