Federated Data Analytics with Differentially Private Density Estimation Model

Jiayi Wang[1], Lei Cao[2], Chengliang Chai[3], Guoliang Li[1]
Tsinghua University[1], University of Arizona/ MIT CSAIL[2], Beijing Institute of Technology[3]

ABSTRACT

Federated data analytics, aimed at extracting insights from decentralized private data while preserving privacy, is crucial for organizations holding sensitive data. Existing approaches, such as output perturbation that adds noise to query results based on differential privacy, often suffer from degraded accuracy due to cumulative privacy budget consumption. In this paper, we introduce ADAPT, a novel framework that addresses this problem by training a privacy-preserving density model over decentralized data. Unlike traditional methods, ADAPT avoids accessing raw data when answering queries, thereby avoiding additional privacy leakage. We tackle the technical challenges raised by privacy-preserving federated data analytics, including parameter misalignment and distribution discrepancy, through innovative techniques of pre-alignment of network parameters and fine-tuning towards accurate data distributions. Directly using the density model, ADAPT accurately infers the results of a wide range of analytical queries. Extensive experiments demonstrate that ADAPT outperforms existing methods in terms of accuracy. Notably, for answering 8,000 analytical queries, ADAPT reduces the median relative error from over 10^3 to less than 6%. Moreover, it achieves high accuracy comparable to *centralized* differential privacy training, demonstrating its effectiveness in practical federated data analytics scenarios.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/for0nething/ADAPT.

1 INTRODUCTION

Many organizations hold valuable and sensitive data that cannot be shared with the public due to privacy concerns. This urges the need for techniques that can effectively conduct decentralized data analytics over private data owned by multiple organizations, while still preserving data privacy, so call *federated data analytics*. Federated data analytics can discover meaningful insights while each individual organization cannot if it only analyzes its own data.

EXAMPLE 1. *Consider several hospitals each possessing private patient records. These hospitals want to conduct data analytics to investigate the factors influencing a rare disease, for example, exploring the relationship between the use of surgical treatment (have undergone at least one surgery) and the cure rate for a rare disease X. Hence, they may run the following SQL query over their private data, which only contains a limited amount of patient records.*

```
SELECT AVERAGE(CureRate) FROM T
WHERE  T.DISEASE = X AND T.SURGERY ≥ 1;
```

In this example, because each hospital has no sufficient data, running analytical queries independently on their own small data might lead to misleading conclusions. Federated data analytics targets resolving this critical issue.

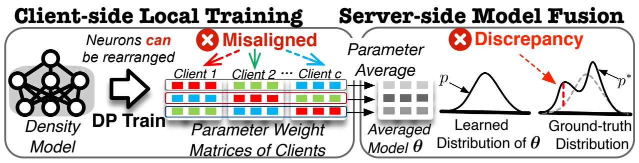
Client-side Local Training
Server-side Model Fusion
Neurons can be rearranged
Misaligned
Discrepancy
Client 1
Client 2
Client c
Density Model
DP Train
Parameter Weight Matrices of Clients
Parameter Average
Averaged Model θ
p
p*
Learned Distribution of θ
Ground-truth Distribution

Figure 1: Limitations in training density models in FL.

One intuitive way to support federated data analytics is to leverage the *output perturbation* technique [4, 5, 14, 26, 41, 54] introduced in differential privacy (DP). More specifically, it first requests each client to query its own data and injects noise to the query result to mitigate privacy leak with differential privacy guarantee; it then aggregates the noisy results reported by the clients to produce the final result. However, differential privacy allows each query to leak certain amount of private information, and moreover privacy leak accumulates. The more queries a client runs, the more private information will leak. As the goal is to preserve the privacy of the whole database, a total privacy budget [16] has to be assigned to the client. However, each query will consume certain amount of privacy budget. Therefore, a query will only get a small privacy budget when there are a large number of queries. Consequently, a large amount of noise has to be added to the result of each query, resulting in poor query accuracy. Therefore, this method fails to produce accurate results when there exist a large number of queries. For example, on a real-world dataset used in our experiments, the median relative error was over $10,000\%$ when answering only 100 queries.

Key Idea. In this work, we propose ADAPT, which for the first time makes federated data analytics practical. The key idea is to first train a *privacy-preserving density model* that accurately learns the joint data distribution across all clients, i.e., a probability density function; queries will then be directly approximated based on the density model *without accessing the private data*. This avoids additional privacy leak, thus addressing the above problem.

Technical Challenges. The success of this new federated data analytics paradigm relies on two factors: an accurate density model and an efficient inference strategy. To learn an accurate density model over the decentralized private data, a natural way is to adopt federated learning (FL) [40], where model parameters exchange iteratively between the server and clients. In the typical FL framework, clients receive model parameters, train with their local data, and send back updated parameters to the server. These updated parameters are then averaged coordinate-wise on the server to obtain a new shared global model. However, training density models in FL faces two critical challenges (Figure 1) that degrade the performance of model.

Parameter Misalignment (C1): FL commonly uses coordinate-wise averaging to aggregate the models received from the clients. Assume that each parameter has a coordinate in the client model. Coordinate-wise averaging aggregates the client models by taking an average

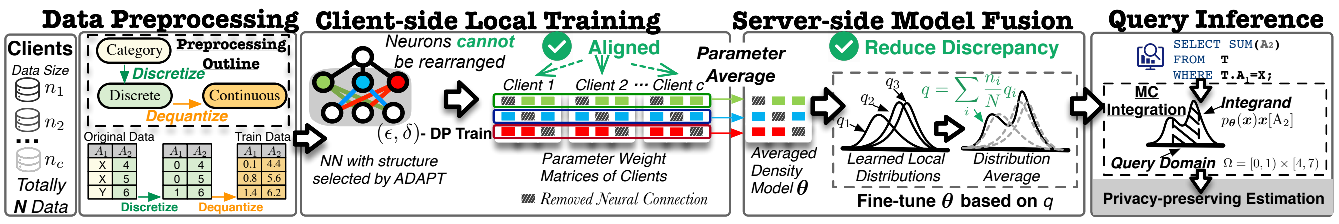
Data Preprocessing
Client-side Local Training
Server-side Model Fusion
Query Inference
Clients
Data Size
n_1
n_2
n_c
Totally N Data
Category
Preprocessing Outline
Discretize
Discrete
Continuous
Dequantize
Original Data
Train Data
Discretize
Dequantize
Neurons cannot be rearranged
Aligned
(ϵ, δ)- DP Train
NN with structure selected by ADAPT
Client 1
Client 2
Client c
Parameter Weight Matrices of Clients
Removed Neural Connection
Parameter Average
Averaged Density Model θ
Reduce Discrepancy
$q = \sum_i \frac{n_i}{N} q_i$
q_1
q_2
q_3
Learned Local Distributions
Distribution Average
Fine-tune θ based on q
SELECT SUM(A₂) FROM T WHERE T.A₁=X;
MC Integration
Integrand $p_\theta(x)x[A_2]$
Query Domain $\Omega = [0,1) \times [4,7)$
Privacy-preserving Estimation

Figure 2: Framework of ADAPT that trains a privacy-preserving density model in federated learning for analytical queries.

over the parameters at the same coordinate. However, the parameters of each model, *e.g.,* hidden neurons in fully connected layers, can be randomly permutated during training without impacting network outputs, leading to a parameter misalignment across the models. This often significantly degrades the accuracy of the aggregated model. To address this issue, existing works [36, 51, 57, 58] use an additional matching step to align neurons after each client trains its local model. However, this post-processing method is ineffective for privacy-preserving local training because the noise added to the model parameters misleads the matching. Moreover, it is also inefficient due to the additional computational overhead for matching.

Distribution Discrepancy (C2): The density models trained independently at different clients only learn the local distributions, while our ultimate goal is to learn the joint distribution across all clients. However, as illustrated in Figure 1, even if we could completely resolve the misalignment issue in coordinate-wise averaging, the probability density function (PDF) which the averaged density model estimates would still significantly deviate from the PDF learned from all client data in a centralized setting. This is because as a heuristic, the parameter average does not guarantee any error bound. Even if each client could learn a density model that perfectly fits its own data distribution, simply averaging their parameters would not produce a density model that accurately fits the joint data distribution across different clients.

Inference Inefficiency (C3): Answering analytical queries with a learned density model requires computing one or several integrals of the PDF over the value range specified by the query predicates, known as *query domain* Ω. Specifically: (1) the integrand is an expression of density corresponding to the aggregate function of the query; (2) the probability density is inferred from the density model. However, these integrations do not have analytical solutions. Existing works that use density models for cardinality estimation or approximate query processing rely on Monte Carlo integration techniques, including uniform sampling [39], progressive sampling [56] and adaptive importance sampling [19, 52, 53]. However, to accurately estimate the probability density with a model, these methods have to generate numerous sampling points within the query domain, resulting in significant computational overhead. This inefficiency issue is further exacerbated on complex aggregation queries requiring multiple integrations, like MODE.

Proposed Approach. We propose ADAPT which features novel techniques to address the above challenges in training privacy-preserving density models as well as an adaptive importance sampling-based method that effectively and efficiently approximates a broad range of analytical queries using the trained density model. In particular, our density model training method successfully overcomes the performance gap between federated learning and centralized training: ADAPT is able to produce a density model with accuracy close to a model learned with centralized privacy-preserving training.

Training Privacy-preserving Density Model. To address the *parameter misalignment* problem (C1), after careful analysis, we observe that it is the *permutation invariance* property [51, 58] of neural networks that causes this problem, where neurons in the same layer cannot be distinguished and aligned, resulting in that hidden layer neurons can be reordered without affecting the network outputs. Based on this observation, we propose a novel method that contrary to existing *post-processing*-based matching methods, *pre-aligns* the models before the clients train their local models. More specifically, we propose to alleviate the permutation invariance problem by judiciously designing the neuron structure to make neurons easily distinguished with different neural connections. This guarantees that permutating neurons would produce networks with different outputs, thus addressing the parameter misalignment. However, arbitrarily setting the neural connections may remove too many connections, resulting in an overly sparse, hence ineffective model. We thus propose a neural connection selection method to address this issue. We first formulate this problem as an optimization objective that given a hard constraint on the number of removed neural connections, minimizes the pairwise similarity between neurons. We prove that the optimal solution to this problem can be found by an efficient algorithm with a time complexity linear to the number of model parameters.

To overcome the *distribution discrepancy* problem (C2), we first propose a method called *distribution average* (DA) that can estimate the PDF of the joint distribution across all clients with *bounded error* for any data point. However, answering an analytical query by directly using DA to estimate the probability density of the corresponding domain values is not practical, because the estimation of DA has to leverage all local models, which inherently leaks more privacy [6, 13, 37] and introduces higher inference latency [36, 45]. To address these issues, we propose a method that instead leverages the theoretical conclusion of DA to guide the federated training process. It fine-tunes the parameter-averaged model to imitate the PDF computed by DA, and finally produces a single density model, thus dramatically alleviating the distribution discrepancy problem.

Query Inference Acceleration. To improve the efficiency of query inference (C3), we focus on reducing redundant sampling points based on two observations. First, not all integrations contribute equally to the accuracy of analytical queries. We thus design a strategy to identify the integrations that have negligible impact on query results and stops sampling at early stage. In this way, ADAPT avoids unnecessary computations, while not sacrificing accuracy. Second, complex queries are often decomposed into multiple *similar* queries with highly overlapping query domains. We propose to reuse samples from previous integrations in subsequent ones to reduce the total number of required samples, thereby improving efficiency.

Contributions. In summary, we make the following contributions:

(1) We propose ADAPT, a new paradigm for Approximate federated Data Analytics leveraging the Privacy-preserving densiTy model. To the best of our knowledge, this is the first work that uses the density model to solve private federated data analytics.

(2) We propose a network structure selection algorithm to address the parameter misalignment problem of density models, which prealigns neurons to avoid random parameter permutation in training.

(3) We solve the distribution discrepancy problem by computing an error-bounded PDF leveraging the local models and employing it to fine-tune the parameter-averaged model.

(4) We propose an inference acceleration algorithm that avoids redundant sampling in answering complex analytical queries.

(5) We conduct extensive experiments to demonstrate that ADAPT significantly outperforms existing private federated data analytics methods in terms of accuracy. In particular, for answering 8,000 analytical queries, ADAPT reduces the median relative error from 10^3 to less than 6%. Moreover, compared with SOTA FL training methods, ADAPT produces a much more accurate density model: ADAPT reduces the estimation error by over 10x, and achieves similar accuracy as centralized DP training – the *upper bound* in this setting.

2 PRELIMINARY

2.1 Problem Definition

Privacy-Preserving Federated Density Estimation. Consider c clients each holding private data with m attributes $\{A_1, A_2, \ldots, A_m\}$. Each client U_j has n_j private data records, denoted as $\boldsymbol{D}_j = \{\boldsymbol{x}_j^{(i)}\}_{i=1}^{n_j}$ and we denote the total number of data records across all clients as $N = \sum_{j=1}^{c} n_j$. The objective of privacy-preserving federated density estimation (*PrivFDE*) is to *maximize the accuracy of the density model subject to the differential privacy (DP) and federated learning constraints*. The DP constraint requires the common sample-level differential privacy [1], *i.e.,* each client satisfies (ϵ, δ)-differential privacy with respect to its local dataset. Here the parameter ϵ, known as *privacy budget*, quantifies the degree of privacy preservation. A smaller ϵ indicates stronger privacy preservation. The parameter δ is the likelihood of privacy leakage beyond the protection provided by ϵ and typically takes a very small value, *e.g.,* 10^{-5}. The FL constraint requires the raw data of each client to be stored locally and only intermediate model updates are transferred in collaborative training.

Formally, we define the optimization objective of *PrivFDE* as maximizing model accuracy using the maximum likelihood principle under FL and DP constraints:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{j=1}^{c}\sum_{i=1}^{n_j} -\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_j^{(i)}) \\ &= \arg\min_{\boldsymbol{\theta}} \sum_{j=1}^{c} \frac{n_j}{N} L(\boldsymbol{\theta}; \boldsymbol{D}_j)\end{aligned} \tag{1}$$

where $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ denotes the probability density of data point $\boldsymbol{x}$ computed by the density model parameterized by $\boldsymbol{\theta}$; and $L(\boldsymbol{\theta}; \boldsymbol{D}_j)$ denotes the loss on local dataset $\boldsymbol{D}_j$.

Supported Query Types. ADAPT directly uses the density model to support a broad range of aggregate queries without accessing the private data. In summary, ADAPT supports SQL queries of the form:

```
SELECT AGG(Y) FROM T
[WHERE      PREDICATE1]
            ...
[AND/OR     PREDICATEg]
[GROUP BY G];
```

Here AGG(Y) denotes an aggregate function applied to attribute Y over the tuples filtered by the predicates. ADAPT supports various aggregate functions, such as COUNT, SUM, AVG, VARIANCE, PERCENTILE, MODE or STDDEV (Section 5.1). ADAPT supports both *conjunctions* and *disjunctions* of single-column predicates on numerical and categorical data. Each predicate PREDICATE$_j$ for attribute A_i can either be an equality predicate (*e.g.,* $A_i = a_i$) or a range predicate (*e.g.,* $A_i \geq l_i$ or $l_i \leq A_i \leq h_i$). Without loss of generality, we focus on conjunctions in this paper, as disjunctions can be converted into conjunctions by the inclusion-exclusion principle [52]. Moreover, GROUP BY queries can be supported by decomposing the query into multiple queries that replace the GROUP BY predicate with equality predicates enumerating all distinct values of the group attribute.

2.2 Normalizing Flow

In the past decades, various generative models have been proposed for learning the joint data distribution. However, although diffusion models [22, 46], GANs [18] and VAEs [28] excel in tasks like image synthesis, they do not explicitly estimate the probability density, thus fail to meet our need. In contrast, normalizing flows (NF) [12, 15, 44] are a family of generative models known for their state-of-the-art *density estimation* capability, adopted in a range of core database problems, *e.g.,* cardinality estimation [52, 53] and learned index [55]. Naturally, in this work we adopt NF as the density estimation model, although our proposed framework is compatible to other density models as well, as confirmed in our experiments (Section 6.10).

To be specific, NF is defined by a sequence of invertible transformations $\mathbf{f}_1, \ldots, \mathbf{f}_k$, which converts a data point $\boldsymbol{x}$ to a random variable $\boldsymbol{u}$ with tractable probability density via a sequence of latent variables $\mathbf{h}_1, \ldots, \mathbf{h}_{k-1}$ following the sequence:

$$\boldsymbol{x} \overset{\mathbf{f}_1}{\longleftrightarrow} \mathbf{h}_1 \overset{\mathbf{f}_2}{\longleftrightarrow} \mathbf{h}_2 \cdots \overset{\mathbf{f}_k}{\longleftrightarrow} \boldsymbol{u}$$

Since the density change of each $\mathbf{f}_j$ can be quantified by the Jacobian matrix, the probability density of a data point $\boldsymbol{x}$ can be computed utilizing the change-of-variable formula as follows:

$$\log p(\boldsymbol{x}; \boldsymbol{\theta}) = \log p(\boldsymbol{u}) + \log\left|\frac{\partial \mathbf{f}}{\partial \boldsymbol{x}}\right| = \log p(\boldsymbol{u}) + \sum_{i=1}^{k} \log\left|\frac{\partial \mathbf{f}_i}{\partial \mathbf{h}_{i-1}}\right| \tag{2}$$

Here $\log|\frac{\partial \mathbf{f}}{\partial \boldsymbol{x}}|$ and $\log|\frac{\partial \mathbf{f}_i}{\partial \mathbf{h}_{i-1}}|$ denote the logarithm of the absolute determinant of the Jacobian matrices of $\mathbf{f}$ at $\boldsymbol{x}$ and $\mathbf{f}_i$ at $\mathbf{h}_{i-1}$. Each transformation $\mathbf{f}_i$ generally employs a coupling layer [12, 15, 42] that divides the attributes into two parts and learns the relationship between them. In training, the maximum likelihood principle is used.

2.3 Overall Framework

ADAPT learns an NF model that effectively estimates the joint distribution of all client data while ensuring differential privacy. As shown in Figure 2, the clients first jointly train an (ϵ, δ)-DP NF model $\boldsymbol{\theta}$ using federated learning. Once trained, the NF model $\boldsymbol{\theta}$ is leveraged to answer analytical queries via privacy-preserving probability density computation.

Training. ADAPT first preprocesses the original data of all clients, transforming both numerical and categorical data into a continuous format suitable for training. Numerical data, which include both continuous and discrete data, are handled differently: continuous data are already supported, while discrete data have to undergo a uniform dequantization [21, 23, 48] that adds uniform noise between 0 and the width of each discrete bin to make them continuous. Categorical data are discretized into integers as in existing works [19, 56], and then tackled as discrete data. For example, as illustrated in Figure 2, a categorical value X in column A_1 is first discretized into 0 and then dequantized into a continuous form by adding uniform noise within $[0, 1)$, *e.g.,* 0.1.

After preprocessing, to train a more accurate density model under FL and DP constraints, we first select a network structure that can achieve parameter alignment across multiple clients (Section 3). For privacy-preserving local training, we employ DP-SGD [1] that adds noise to gradients. To mitigate the distribution discrepancy problem that the distribution learned by the model of parameter averaging deviates from the target distribution, we propose a fine-tuning algorithm to adjust the averaged model (Section 4).

Inference. As the learned joint data distributions are continuous, the analytical queries can be estimated by integrations over the query domain Ω. Formally, we denote the value range satisfying the predicates for attribute A_i as R_i, thus $\Omega = R_1 \times \cdots \times R_m$. Because not all predicates are range predicates, we convert equality predicates over categorical values into ranges, similar to the preprocessing in the training phase. For example, the query in Figure 2 corresponds to $R_1 = [0, 1)$ and $R_2 = [4, 7)$, making $\Omega = [0, 1) \times [4, 7)$.

During inference, ADAPT first computes the legal range R_i for each attribute based on the query predicates to obtain Ω. Analytical queries are then expressed as integrations of the joint data distribution over Ω. For instance, the above SUM query is computed by $N \cdot \int_{\boldsymbol{x} \in \Omega} p_{\boldsymbol{\theta}}(\boldsymbol{x}) \boldsymbol{x}[A_2] \; dx$, where $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the probability density of $\boldsymbol{x}$ and $\boldsymbol{x}[A_2]$ is the value of attribute A_2 for data point $\boldsymbol{x}$. ADAPT approximates these integration results using Monte Carlo (MC) integration [32] over the Ω. However, some analytical queries require density estimation for a large number of data samples. To improve computation efficiency, we propose optimizations to reduce redundant sampling points for acceleration (Section 5).

DP Guarantee. Since the NF model $\boldsymbol{\theta}$ satisfies (ϵ, δ)-DP, answers given by $p_{\boldsymbol{\theta}}$ without accessing raw data also satisfies (ϵ, δ)-DP [17].

3 ALIGNING NETWORK PARAMETERS USING DISTINCTIVE NEURONS

Our main idea for solving the parameter misalignment problem is to make the neurons distinguishable. We first outline the overall solution in Section 3.1, where we ensure neurons at different locations are distinctive by assigning them distinct neural connections, thus their parameters can be pre-aligned. We then formalize the optimization problem of selecting appropriate neural connections

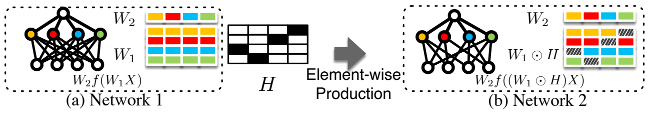

Figure 3: An Example of Neurons with Different Connections.

for each neuron in Section 3.2, derive its optimal minimum value in Section 3.3, and propose an efficient construction algorithm to achieve the optimum in Section 3.4.

3.1 Overall Solution

Our key idea is to enforce *distinctiveness among neurons prior to model training* by assigning unique roles to neurons in each hidden layer. This ensures that neurons in different positions capture specialized information, thus avoiding random parameter permutation in training and effectively aligning neurons across clients.

However, during training, we lack direct control over the weight matrix W of each layer. Therefore, we propose to enhance the orthogonality between neuron parameters by *adjusting the network structure*. Specifically, we drop some neural connections in the original fully connected layer to make neurons in the same layer possess different neural connections. In this way, the connections of neurons are not the same and thus the neurons are no longer interchangeable. Figure 3 shows an example network. After dropping some neural connections, the hidden layer neurons in Figure 3(b) become non-exchangeable because they own different neural connections.

As our goal is to maximize the difference in neural connections among neurons in the same layer, one straightforward approach is to remove an arbitrary number of connections for each neuron. However, although this is able to maximize the structural difference, it will also make the network much sparser than the original network. This affects the expressive capability of the network, thus degrading the final accuracy. Therefore, we aim to propose an approach that selectively discards some neural connections to maximize the difference between neurons, while not making the network too sparse.

3.2 Neural Connection Selection

Consider a hidden layer in a fully connected network, where the input size is m and this layer comprises n neurons. The status of the neural connections of this layer can be denoted by an $n \times m$ binary matrix H. Let each entry H_{ij} denote whether the connection from the i-th neuron to the j-th input is present ($H_{ij} = 1$) or absent ($H_{ij} = 0$); and the i-th row in H, denoted as H_i, represents the neural connections for the i-th neuron. The original complete fully connected network corresponds to an H matrix with all ones. To avoid the network becoming too sparse, we restrict each H_i to have at most K zeros, where K is a constant, proportional to m. Therefore, our optimization goal is to maximize the differences between neurons subject to this constraint. Given two neurons, we measure their difference based on the **orthogonality** degree between their neural connections H_i and H_j, which can be computed as the dot product [9]:

$$D(H_i, H_j) = \sum_{k=1}^{m} H_{ik} \cdot H_{jk} \tag{3}$$

Here, a smaller value indicates a higher degree of orthogonality. Therefore, to maximize the neuron difference, the objective is to minimize the sum of all pair-wise $D(H_i, H_j)$, which is equivalent to the sum of all entries in HH^T except the terms on the diagonal:

$$\arg\min \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} (HH^T)_{ij} \tag{4}$$

Here $(HH^T)_{ij}$ denotes the j-th entry in the i-th row of HH^T. Now, we are ready to formally define the problem we target.

PROBLEM 1. ***Neuron Structure Selection.*** *Given a hidden layer with n neurons and m-dimensional input in a fully connected network and the sparsity constraint constant K. The task of neuron structure selection is to construct an $n \times m$ binary matrix H denoting the neural connections of the layer, where each row has at most K zeros, with the objective of minimizing $\sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} (HH^T)_{ij}$.*

3.3 Optimal Result

To guide the design of an effective selection algorithm, we first derive the optimal result of Problem 1 and establish the conditions for achieving this optimal result.

For the matrix H, let $a_1, \ldots, a_m$ denote the count of zeros in the m columns, and let $b_1, \ldots, b_m$ denote the count of ones in the m columns. The optimal value and conditions are as follows.

LEMMA 1. *The optimal result of Problem 1 is $n^2m - 2n^2K + nK - nm + \frac{n^2K^2}{m}$ and can be achieved if and only if $b_1 = \cdots = b_m = \frac{n(m-K)}{m}$.*

PROOF. Given that there are n rows, we have $a_i + b_i = n$ for $1 \le i \le m$. Additionally, since each row has at most K zeros, there are at most nK zeros in H, leading to the constraints $\sum_{i=1}^{m} a_i \le nK$ and $\sum_{i=1}^{m} b_i = nm - (\sum_{i=1}^{m} a_i) \ge n(m-K)$.

Since $(HH^T)_{ij} = D(h_i, h_j) = \sum_{k=1}^{m} H_{ik} \cdot H_{jk}$, the optimization target in Equation 4 can be transformed as follows:

$$\sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} (HH^T)_{ij} = \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \sum_{k=1}^{m} H_{ik} \cdot H_{jk} \tag{5}$$

By swapping the second and the last sum, we obtain:

$$\sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \sum_{k=1}^{m} H_{ik} \cdot H_{jk} = \sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik} \cdot (\sum_{j=1,j\neq i}^{n} H_{jk}) \tag{6}$$

Since $\sum_{j=1,j\neq i}^{n} H_{jk} = b_k - H_{ik}$, Equation 6 can be transformed:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik} \cdot (\sum_{j=1,j\neq i}^{n} H_{jk}) = \sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik} b_k - \sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik}^2 \tag{7}$$

The first term can be further transformed as:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik} b_k = \sum_{k=1}^{m} b_k \sum_{i=1}^{n} H_{ik} = \sum_{k=1}^{m} b_k \cdot b_k = \sum_{k=1}^{m} b_k^2 \tag{8}$$

Since H is a binary matrix, with entries taking values from 0 and 1, we have $H_{ik}^2 = H_{ik}$. Therefore, the second term can be transformed:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik}^2 = \sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik} = \sum_{k=1}^{m} \sum_{i=1}^{n} H_{ik} = \sum_{k=1}^{m} b_k \tag{9}$$

With Equation 8 and Equation 9, the target can be transformed:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik} b_k - \sum_{i=1}^{n} \sum_{k=1}^{m} H_{ik}^2 = \sum_{k=1}^{m} (b_k - \frac{1}{2})^2 - \frac{m}{4} \tag{10}$$

Algorithm 1 Neuron Structure Selection For Fully Connected Layer

```
Input: Number of neurons $n$, input dimension $m$, constraint $K$ on
    zeros per row
Output: $n \times m$ binary matrix $H$ with at most $K$ zeros per row,
    minimizing $\sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} (HH^T)_{ij}$
1: Init $n \times m$ binary matrix $H$ with all ones
2: Init candidate row set $P = \{1, \ldots, n\}$
3: Compute target count of ones in each column: $b_1, \ldots, b_m$
4: for $j = 1, \ldots, m$ do
5:     Random select a subset $S$ of size $b_j$ from $P$
6:     for $i \in S$ do
7:         Set $H_{ij} \leftarrow 0$
8:     /* Identify rows with less than $K$ zeros */
9:     Update $P \leftarrow \{i \mid j - (\sum_{1\le j'\le j} H_{ij'}) < K\}$
10: Return $H$
```

With Cauchy–Schwarz inequality and $\sum_{i=1}^{m} b_i \ge n(m-K)$:

$$\sum_{k=1}^{m} (b_k - \frac{1}{2})^2 \ge m \cdot (\frac{\sum_{k=1}^{m} (b_k - \frac{1}{2})}{m})^2 \ge m \cdot (\frac{n(m-K) - \frac{m}{2}}{m})^2 \tag{11}$$

With Equation 10 and Equation 11, we come to the lower bound:

$$\sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} (HH^T)_{ij} \ge m \cdot (\frac{n(m-K) - \frac{m}{2}}{m})^2 - \frac{m}{4} \tag{12}$$

This minimum equals $n^2m - 2n^2K + nK - nm + \frac{n^2K^2}{m}$ and can be derived when all b_i are equal (by Cauchy–Schwarz inequality) and their sum $\sum_{i=1}^{m} b_i$ equals $n(m-K)$, *i.e.,* each row has exactly K zeros. Thus, the condition for optimality is $b_1 = \cdots = b_m = \frac{n(m-K)}{m}$. □

The intuition is to evenly distribute inputs among neurons to minimize overlap in their connections. Note that m sometimes does not divide $n(m-K)$. In such cases, $n(m-K)\%m$ of the $b_1, \ldots, b_m$ should be $\lfloor \frac{n(m-K)}{m} \rfloor + 1$, while the remaining should be $\lfloor \frac{n(m-K)}{m} \rfloor$.

3.4 Neural Connection Selection Algorithm

Guided by the above optimality condition, we design an efficient neuron structure selection algorithm that achieves the optimal minimum value. As shown in Algorithm 1, it first initializes all elements in the $n \times m$ matrix H to one (Line 1). Then a set of rows P for keeping track of the rows with less than K zeros in H is established (Line 2). Initially, P contains all rows, *i.e.,* $P = \{1, \ldots, n\}$, since no rows contain zeros. Next, the target count of ones for each column, denoted as $b_1, \ldots, b_m$ is computed using the conclusions drawn in Section 3.3 (Line 3). Subsequently, the algorithm determines the values of the matrix H column by column. For the j-th column, a subset S of rows, with a size of b_j is selected randomly from the candidate row set P (Line 5). For each row H_i within the selected subset, the corresponding entry H_{ij} is set to 0 (Line 7). After that, as some rows now contain more zeros, the candidate row set P is updated based on the current state of H (Line 9). The above process iterates through all columns, ultimately resulting in a binary matrix H that obtains the optimal solution (Line 10).

Time Complexity. The overall time complexity is only $O(nm)$, linear to the size of H. Therefore, Algorithm 1 can efficiently determine the neuron structures for each layer, and finally determine the structure of the whole network.

Construction Algorithm For Small Input Sizes. We further optimize the above approach based on the unique property of density models. Many density models have very small input dimensions in many layers, *e.g.,* the input dimension of each coupling layer in NF is only half of the original data dimension. This causes the problem that only using neural connections is not sufficient to distinguish neurons. Consider a scenario where the input size is only 6 while the hidden layer has 256 neurons. The coupling layer of NF will take three of these dimensions as input and model the correlations between the data of input dimensions and the data of other dimensions. Since 3 neural connections have a total of 2^3 different states, while $2^3 \ll 256$, it is impossible to distinguish the 256 hidden neurons based on the 3 neural connections.

To address this issue, we extend the values for H from $\{0, 1\}$ to $\{0, \frac{1}{p}, \frac{2}{p}, \ldots, 1\}$, denoting the weight of each neural connection, where p is an integer, *e.g.,* 10. In this way, suppose the input size is r, the number of possible neural connection states is augmented from 2^r to p^r such that even a small input size is sufficient to distinguish hundreds of neurons. This extension requires only a minor change to Algorithm 1. Specifically, we just need to replace the 0 in Line 7 with uniform sampling from $\{0, \frac{1}{p}, \ldots, \frac{p-1}{p}\}$, *i.e.,* setting H_{ij} as a random value drawn from $\{0, \frac{1}{p}, \ldots, \frac{p-1}{p}\}$ instead of being fixed at 0. With such an extension, Algorithm 1 remains efficient with the same $O(nm)$ time complexity while able to support the small input size in the NF models for providing parameter alignment.

4 FEDERATED MODEL TRAINING

To overcome the *distribution discrepancy* problem, we propose a *Distribution Average* method, which approximates the target joint distribution of all client data with a bounded error. Guided by this more accurate approximation, ADAPT fine-tunes the model obtained by parameter averaging to eliminate the distribution discrepancy. In Section 4.1, we first introduce the Distribution Average approach. Then in Section 4.2, we theoretically prove its error bound. Finally in Section 4.3, we introduce the training algorithm.

4.1 Distribution Average

Note our ultimate objective is to accurately learn the joint distribution of data from all clients. We observe that this target distribution constitutes a *mixture distribution*, *i.e.,* a weighted combination of the local distributions of the clients, denoted mathematically as $p^* = \sum_i w_i p_i$. Here, p^* denotes the target distribution and each local distribution p_i of client U_i serves as a component distribution, with a mixture weight $w_i = \frac{n_i}{N}$. Here, the mixture weight can be interpreted as the probability that a data sample is drawn from this mixture component. Hence, client U_i with n_i data samples has a weight w_i of $\frac{n_i}{N}$. Therefore, the density in the target distribution is a mixture density that can be computed as the weighted average of local distribution densities, *i.e.,* $p^*(x) = \sum_i w_i p_i(x)$. Since each local model is trained on local data for many iterations to fit the local distribution p_i by the learned distribution q_i, the mixture of learned distributions, *i.e.,* $\sum_i w_i q_i(x)$ can form an accurate approximation of p^*, which we call *Distribution Average (DA)*.

4.2 Error Bound of Distribution Average

DA approximates p^* by $q = \sum_i w_i q_i$. Next, we show that the approximation error of q to p^*, measured by their KL-divergence $D_{KL}(p^* \| q)$ is bounded.

Since employing neural networks inevitably incurs certain approximation errors, we introduce γ_i to denote the error when fitting p_i by q_i, which is defined as the maximum ratio between $p_i(x)$ and $q_i(x)$, *i.e.,* $\gamma_i = \max\{\frac{p_i(x)}{q_i(x)}\}$. Thus, we have $\frac{p_i(x)}{q_i(x)} \leq \gamma_i$ for all x.

The KL-divergence between p^* and q is defined as:

$$D_{KL}(p^* \| q) = \int p^*(x) \log \frac{p^*(x)}{q(x)} \, dx \tag{13}$$

Leveraging $p^* = \sum_i w_i p_i$, $q = \sum_i w_i q_i$ and $\frac{p_i(x)}{q_i(x)} \leq \gamma_i$, we have:

$$\frac{p^*(x)}{q(x)} = \frac{\sum_i w_i p_i(x)}{\sum_i w_i q_i(x)} \leq \frac{\sum_i \gamma_i w_i q_i(x)}{\sum_i w_i q_i(x)} \tag{14}$$

Taking $\gamma_{max} = \max\{\gamma_i\}$, we have:

$$\frac{\sum_i \gamma_i w_i q_i(x)}{\sum_i w_i q_i(x)} \leq \frac{\sum_i \gamma_{max} w_i q_i(x)}{\sum_i w_i q_i(x)} = \gamma_{max} \tag{15}$$

Therefore, we have $\frac{p^*(x)}{q(x)} \leq \gamma_{max}$. Since p^* is a probability density function satisfying $\int p^*(x) dx = 1$, we can obtain the upper bound of the KL-divergence as:

$$\begin{aligned} D_{KL}(p^* \| q) &= \int p^*(x) \log \frac{p^*(x)}{q(x)} \, dx \\ &\leq \int p^*(x) \log(\gamma_{max}) \, dx = \log(\gamma_{max}) \end{aligned} \tag{16}$$

By Equation 16, the approximation error of DA is upper bounded by the approximation error of the most inaccurate local model, which is generally small in practice.

4.3 Federated Flow Training Algorithm

ADAPT leverages *DA* to fine-tune the density model learned by the classic parameter averaging in FL.

Finetune via KL Divergence. In the t-th round, the central server receives updated local models $\boldsymbol{\theta}_1^t, \ldots, \boldsymbol{\theta}_c^t$ from clients and aggregates them using parameter average, obtaining the averaged model $\boldsymbol{\theta}^t$. We use q' to denote the distribution represented by $\boldsymbol{\theta}^t$ and q to denote the distribution obtained by DA of the learned distributions of $\boldsymbol{\theta}_1^t, \ldots, \boldsymbol{\theta}_c^t$.

To narrow the gap between q' and q is equivalent to adjusting $\boldsymbol{\theta}^t$ to reduce the KL divergence $D_{KL}(q \| q')$, defined as:

$$D_{KL}(q \| q') = \int q(x) \log \frac{q(x)}{q'(x)} \, \mathrm{d}x \tag{17}$$

Unfortunately, the exact computation of $D_{KL}(q \| q')$ is infeasible due to the absence of closed-form expressions for $q(x)$ and $q'(x)$. However, based on the law of large numbers, the KL divergence can be approximated using a set of data points $\boldsymbol{D}$ by [25]:

$$D_{KL}(q \| q') = \lim_{|\boldsymbol{D}| \to \infty} \sum_{x_i \in \boldsymbol{D}} \frac{1}{|\boldsymbol{D}|} \log \frac{q(x_i)}{q'(x_i)} \tag{18}$$

However, relying on samples to approximate the KL divergence requires additional data. Fortunately, normalizing flow models, as a class of generative models, inherently possess the capability to generate synthetic data points. Therefore, we first utilize the NF

Algorithm 2 Federated Learning with Refinement of NF Model

```
Input: c clients, each client $U_j$ possesses $n_j$ pieces of private data
    $D_j = \{x_j^{(i)}\}_{i=1}^{n_j}$, in total $N$ data, the number of communication
    rounds $T$, the number of local epochs $E$, the learning rate of
    local training $\eta$, the learning rate of fine-tuning the NF model $\eta'$
Output: Global NF model $\theta$ that learns the joint distribution of all
    client data
1:  Server Executes:
2:  Init $\theta^0$
3:  for $t = 1, \ldots, T$ do
4:      for $i = 1, \ldots, c$ in parallel do
5:          Send global model $\theta^{t-1}$ to client $U_i$
6:          $\theta_i^t \leftarrow$ ClientUpdate($i, \theta^{t-1}$)
7:      /* Model obtained by parameter average */
8:      $\theta^t \leftarrow \sum_{i=1}^{c} \frac{n_i}{N} \theta_i^t$
9:      /* Fine-tuning the fused NF model*/
10:     $D^t \leftarrow$ Generate a set of data points using $\theta^t$
11:     $q^t = \sum_{i=1}^{c} \frac{n_i}{N} p_{\theta_i^t}(D^t)$
12:     for each batch $B$ of $D^t$ do
13:         $\theta^t \leftarrow \theta^t - \eta' \nabla \mathcal{L}(\theta^t; B)$
14: Return $\theta^T$
15: function ClientUpdate($i, \theta^{t-1}$)
16:     $\theta_i^t = \theta^{t-1}$
17:     for epoch $k = 1, 2, \ldots, E$ do
18:         for each batch $b$ of $\mathcal{D}_i$ do
19:             $\theta_i^t \leftarrow \theta_i^t - \eta \nabla L(\theta_i^t; b)$
20:     Return $\theta_i^t$ to the sever
```

model parameterized by θ^t to generate a small set of data points D^t, *e.g.,* $0.1N$ data. Subsequently, we compute the probability densities of these points in q and q' separately. The density in q' can be computed using the NF model parameterized by θ^t, while the density in q can be computed using all the client models based on DA. Finally, we use D^t and their probability densities computed using DA as training data to refine θ^t. The objective is to minimize the KL divergence loss $\mathcal{L}(\theta^t; D^t)$, where the loss function $\mathcal{L}$ is defined as:

$$\mathcal{L}(\theta; D) = \sum_{x_i \in D} \frac{1}{|D|} \log \frac{q(x_i)}{p_\theta(x_i)} \tag{19}$$

Through iterative adjustment of θ^t to reduce the loss over D^t, the gap between the learned distribution and the error-bounded distribution q is significantly narrowed. This iterative process results in more accurate model fusion in each federated round, ultimately yielding a more accurate trained model.

Training Algorithm. Algorithm 2 shows the federated learning algorithm augmented with the distribution average-based fine-tuning. The server starts by initializing the network structure using the neural connection selection algorithm proposed in Section 3.4 for aligning network parameters (Line 2). The server and clients then train the model in T rounds. In the t-th round (Line 4-Line 13), the server sends the global model θ^{t-1} to the clients (Line 5), and all the clients perform E epochs of privacy-preserving local training to learn the local distributions in parallel (Line 6). Then each client U_i uploads its updated model θ_i^t back to the server (Line 20). The server first conducts a basic coordinate-wise parameter average of these models to obtain an initial global model θ^t (Line 8). Then the server further fine-tunes θ^t (Line 10-Line 13) based on Distribution Average. This refinement produces a more accurate global model θ^t which is then used for subsequent federated training. After T rounds of federated training, ADAPT produces the global model θ that accurately captures the joint distribution of all client data.

5 INFERENCE

After training the privacy-preserving NF model θ, the next step is to utilize the model to estimate the results of analytical queries.

Given a query Q with an aggregate function AGG(Y) over attribute Y, we first identify the query domain Ω based on predicates of Q. We then transform the query into an expression of one or more integrals, where the integrands correspond to the aggregate function AGG and Ω denotes the integral interval.

Next, we will first introduce how to convert Q into integrals for different aggregate functions in Section 5.1. Subsequently, we will introduce how to utilize Monte Carlo (MC) integration to compute the integrals for answering queries in Section 5.2. We then present the acceleration optimizations in Section 5.3.

5.1 Supporting Different Aggregate Functions

For simplicity, we use $p(x)$ to denote $p_\theta(x)$ and use y to denote x[Y], *i.e.,* the value of attribute Y for a data point x.

COUNT. The COUNT aggregation can be computed as:

$$N \cdot \int_\Omega p(x)\, \mathrm{d}x$$

Here, the integral of $p(x)$ over Ω computes the proportion of data points within Ω, also known as the *selectivity* of the query. Therefore, multiplying it by the total data size N leads to the total number of data records that meet the query predicates, *i.e.,* the COUNT result.

AVG. The AVG aggregation can be computed as:

$$\mathbb{E}[\mathrm{Y}] = \frac{\int_\Omega p(x) y\, \mathrm{d}x}{\int_\Omega p(x)\, \mathrm{d}x}$$

Since the average of Y is equivalent to the mathematical expectation of Y, we can compute it as $\mathbb{E}[\mathrm{Y}]$.

SUM. The SUM aggregation can be computed as:

$$COUNT \cdot AVG = N \cdot \int_\Omega p(x) \cdot y \mathrm{d}x$$

Since SUM is the product of COUNT and AVG, we can multiply the expressions of COUNT and AVG to get the SUM result.

VARIANCE. The VARIANCE aggregation can be computed as:

$$\mathbb{E}[\mathrm{Y}^2] - \mathbb{E}[\mathrm{Y}]^2 = \left[\frac{\int_\Omega p(x) y^2\, \mathrm{d}x}{\int_\Omega p(x)\, \mathrm{d}x}\right] - \left[\frac{\int_\Omega p(x) y\, \mathrm{d}x}{\int_\Omega p(x)\, \mathrm{d}x}\right]^2$$

The computation is based on the definition of variance. The first term $\mathbb{E}[\mathrm{Y}^2]$ corresponds to the expectation of Y^2, while the second term is the square of AVG.

STDDEV. The STDDEV aggregation can be computed as:

$$\sqrt{VARIANCE} = \sqrt{\left[\frac{\int_\Omega p(x) y^2\, \mathrm{d}x}{\int_\Omega p(x)\, \mathrm{d}x}\right] - \left[\frac{\int_\Omega p(x) y\, \mathrm{d}x}{\int_\Omega p(x)\, \mathrm{d}x}\right]^2}$$

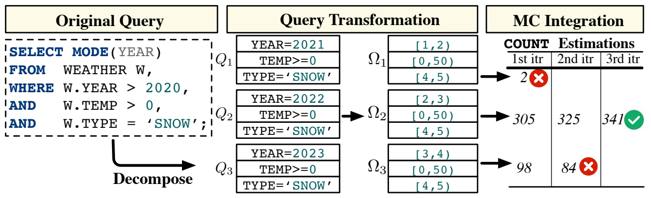

Figure 4: An Example of Early Termination for a `MODE` query.

`STDDEV` is the squared root of `VARIANCE`. Therefore, it can be computed in a similar way to variance.

MODE. `MODE` identifies the most frequently occurring value in the dataset. It can be computed by: (1) decompose the original query into multiple queries by replacing the `MODE` predicate with equality predicates for all possible attribute values; (2) estimate the cardinality of these queries, *i.e.,* compute their `COUNT` results; (3) select the attribute value with the largest estimated `COUNT` as the final result. Figure 4 shows an example for the above process.

PERCENTILE. `PERCENTILE`(Y, p) computes the value below which a percentage (p) of the values in column Y falls. That is, it computes the value α such that $P(\mathrm{Y} < \alpha) = p$. Setting p to 0, 0.5, and 1, `PERCENTILE` is able to estimate `MIN`, `MEDIAN`, and `MAX`. To compute α, the bisection method is employed [39, 50], which involves a binary search of α using multiple `COUNT` queries.

RANGE. `RANGE` computes the difference between the maximum and minimum values in column Y, *i.e.,* `MAX−MIN`. It can be estimated by separately computing `MAX` and `MIN` using the estimation method for `PERCENTILE`.

5.2 Monte Carlo Integration

Although we derive the integral expression, unfortunately there is no way to answer an analytical query by directly computing the integration, due to the absence of a closed-form solution. Therefore, we employ Monte Carlo (MC) integration [31, 32] for approximation. The key idea of MC is to first sample a set of data points from the integration interval Ω, compute their integrand values, and finally integrate the results to get the estimation. Among these steps, *sampling* is crucial to the accuracy and efficiency of MC.

Adaptive Importance Sampling. A straightforward sampling approach is to uniformly sample data records from Ω. However, it tends to degrade accuracy since the integrand within Ω may not exhibit a uniform distribution.

To address this issue, we employ the adaptive importance sampling (AIS) algorithm [31, 32], which refines the sampling process by dividing it into multiple steps. The goal is to sample according to the integrand distribution, denoted as l. To this end, AIS leverages samples from earlier steps to gradually approximate l, thus guiding subsequent sampling steps. To speed up the approximation, AIS divides each attribute of l into a sequence of successive buckets and uniform samples from each bucket. The samples w.r.t. each attribute will then be joined together to form the complete data records.

To accurately approximate l, AIS dynamically adjusts the width of each bucket such that the buckets divide the total integral uniformly. In this way, uniform sampling from the buckets approximately follows the distribution of l. Such partition produces narrower buckets that contain data objects with higher integrand values, *i.e., more important*. Thus, there will be more buckets at the high spikes in the integrand. Consequently, AIS will sample more data records from such *important* areas. AIS iteratively adjusts the buckets until convergence, *i.e.,* almost no further changes in bucket width.

Overall Inference Process. Given the analytical query, `ADAPT` first obtains the query domain of the query and determines the corresponding integrand function based on the aggregate function. At the beginning of MC integration, AIS initializes equi-width buckets for the query domain, as there is no knowledge about l. Then AIS iteratively samples data points from the buckets, computes their integrand values using the NF model, and updates the buckets until convergence. Finally, AIS conducts a weighted aggregation of all the sampled data points across various steps to derive the final integration result for answering the analytical queries.

5.3 Inference Acceleration

Supporting complex aggregate functions typically involves multiple integrations. Independently computing these integrations often results in inefficiency due to a large number of redundant samples. To this end, we propose to leverage inherent correlations between these integrations to reduce redundant sampling in AIS.

Our optimization is based on two key principles: (1) **Early Termination of Low-Impact Integrations**: Answering analytical queries does not always require accurately estimating all integrations. By identifying and terminating low-impact integrations early in the iterative process of AIS, we can avoid unnecessary computations without affecting accuracy. (2) **Sample Reuse Across Similar Integrations**: A complex query is typically decomposed into multiple *similar* queries: have similar query domains and tend to generate similar samples. Therefore, by reusing samples drawn during previous integrations in subsequent integrations, we can reduce the total number of samples, thus improving overall efficiency.

Next, we illustrate these principles using `MODE` and `PERCENTILE` queries as examples.

Early Termination for `MODE` Queries. Estimating a `MODE` query involves decomposing it into multiple `COUNT` queries. Because the goal is to identify the value with the highest cardinality, rather than computing the exact cardinalities for all values, using numerous samples to accurately estimate each `COUNT` is inefficient. The key idea here is to identify the values with small cardinality and stop the corresponding computation early in the iterative AIS process, as these values are unlikely to be the `MODE` result. Specifically, in each iteration, `ADAPT` concurrently conducts AIS for all `COUNT` queries and estimates the result of each query using current sampled data points. In the next iteration, it aborts the queries that have low cardinalities. This significantly reduces the total number of samples, thus improving estimation efficiency.

Figure 4 shows an example. Contrary to the naive AIS that conducts all three iterations for each `COUNT` query, our optimized approach early terminates sampling for Q_1 and Q_3 after the first and second iterations, thus reducing the total number of samples without sacrificing accuracy.

Reuse Samples for `PERCENTILE` Queries. When answering a `PERCENTILE` query, a sequence of `COUNT` queries that differ only in the attribute used by the `PERCENTILE` function will be estimated during the binary search process. Given that queries with similar query domains generate similar samples, this offers opportunities to reuse data points sampled for earlier integrations. Specifically, we initialize the buckets of AIS for each integration using samples from the *most recent* integration that exceeds the target percentile. This is

Table 1: Statistics of datasets.

Dataset	# Rows	# Columns	# Numerical Columns
Power	2,075,259	6	6
IMDB	3,688,889	7	4
BJAQ	382,168	5	5
Flights	5,819,079	8	4

effective in that: (1) the query domain of this query contains the current query, requiring no additional samples for bucket initialization; (2) its query domain has the largest overlap with the current query, ensuring that most samples can be reused. Note this reuse does not sacrifice accuracy, as subsequent sampling iterations still lead AIS to convergence.

$$\Omega_0 = [0,8] \xrightarrow[\text{Converge in 5 iterations}]{\text{Sample } D_0} N_0 = 100$$
$$\Rightarrow \Omega_1 = [0,4] \xrightarrow[\text{Converge in 2 iterations}]{\text{Reuse } D_0\text{; Sample } D_1} N_1 = 60$$
$$\Rightarrow \Omega_2 = [0,2] \xrightarrow[\text{Converge in 2 iterations}]{\text{Reuse } D_1\text{; Sample } D_2} N_2 = 40$$
$$\Rightarrow \Omega_3 = [0,3] \xrightarrow[\text{Converge in 2 iterations}]{\text{Reuse } D_1\text{; Sample } D_3} N_3 = 50$$

Consider above one-dimensional example where the query is to find the median ($p = 0.5$) in $\Omega_0 = [0,8]$. First, we compute the cardinality of the entire domain Ω_0, which converges in 5 iterations using samples D_0. With an estimated N_0 of 100, the target is $N_0 \times p = 50$. The binary search then begins with the estimation of $\Omega_1 = [0,4]$. Given the similarity between Ω_1 and Ω_0, we can reuse the data points in D_0 that fall within Ω_1 to initialize the buckets for AIS. In this way, the number of iterations for Ω_1 is reduced to two. Similarly, both Ω_2 and Ω_3 reuse D_1 and obtain faster convergence in AIS.

6 EXPERIMENTS

Our experiments aim to answer the following questions:

- How does ADAPT perform in answering various aggregate queries compared to other federated data analytics paradigms in accuracy and efficiency? **(Section 6.2)**
- How does ADAPT perform in training density models compared to other FL methods in accuracy and training time? **(Section 6.2)**
- How do the acceleration optimizations impact the efficiency and accuracy of inference? **(Section 6.3)**
- How does ADAPT perform under different DP budgets? **(Section 6.4)**
- How scalable is ADAPT to the number of clients? **(Section 6.5)**
- How does the accuracy of ADAPT change when different proportions of neural connections are removed? **(Section 6.7)**
- How does ADAPT perform when the data amount of clients is skewed? **(Section 6.8)**
- Does ADAPT easily adapt to data changes? **(Section 6.9)**
- Is ADAPT compatible to other density models? **(Section 6.10)**
- How effective are the proposed network structure selection and fine-tuning methods in training process? **(Section 6.11)**

6.1 Experimental Settings

Dataset. The experiments are conducted on four real-world datasets which have been extensively adopted by previous works [15, 20, 43, 52]. The sizes of these datasets vary from 300K to 5M. Table 1 presents the statistical details of these datasets.

(1) Power [24] is a household electric power consumption data, with approximately 2 million numerical data points.
(2) IMDB [30] is a movie dataset commonly used for evaluation of query optimization methods. Following previous works [52, 53], we use the join result of Company_name, Movie_companies, Title, movie_info_idx to evaluate the methods.
(3) BJAQ [7] includes hourly air pollutant data of Beijing, with approximately 300K tuples.
(4) Flights [11] includes the on-time performance of domestic flights operated by large air carriers in the U.S. in 2015.

We distribute the datasets to clients in the same way as the classic FL methods [40, 51]. We first split each dataset into 80% train set and 20% test set. The train set is then uniformly partitioned and assigned to clients as local data. We use the train set for computing the ground truth of analytical queries for evaluation. We use the test set to evaluate the density estimation accuracy of the density models.

Baselines. To validate the effectiveness of our proposed new paradigm, we compare it with the following four categories of baselines:

- **Output Perturbation** (1-2): Add noise to query results to ensure privacy in data analytics.
- **Data Synthesis** (3-4): Learn the local data distribution and then synthesize data for data analytics.
- **Federated Training** (5-8): Within our new federated data analytics paradigm, we employ various FL methods to validate the superiority of ADAPT in training density models.
- **Centralized Training** (9-10): Aggregate all client data to train on one server without protecting privacy. The results serve as a reference to evaluate the performance gap between models trained using FL and those trained in a centralized manner.

(1) DPSQL [54] protects data privacy by adding noise to query results. A privacy budget is evenly allocated across all queries.
(2) FLEX [26] also adds noise to query results for privacy protection. However, FLEX only supports COUNT queries. For a fair comparison, we only compare with FLEX on COUNT queries, and on these queries, assign the same privacy budgets as DPSQL.
(3) PrivMRF [10] is the SOTA data synthesis method with DP guarantee. It generates synthetic data for each client using Markov Random Fields and then collects the data together. Over the collected data, the queries are executed and the results are used as estimations.
(4) DP-WGAN [2] also synthesizes data but uses DP-WGAN.
(5) FedAVG [40] is the classic federated learning algorithm that adopts coordinate-wise averaging to fuse local models.
(6) FedMA [51] matches the neurons of models before parameter average. It formulates neuron matching as a linear assignment problem and solves the problem using the Hungarian algorithm [29].
(7) GAMF [36] improves linear assignment methods for neuron matching [45, 51] by incorporating second-order weight similarity. It achieves the *SOTA accuracy* in matching-based fusion methods.
(8) FedPAN [35] pre-aligns neurons by perturbing the neuron outputs with a location-based periodic function.
(9) CentralDP collects all client data and trains the model with differential privacy (DP) guarantee in a centralized way. However, in reality, collecting all client data to one server is not allowed. Therefore, it only serves as an *upper bound for FL with DP.*
(10) Similarly, Central conducts centralized training but not using DP. Its results serve as an *upper bound* for all the other methods.

Hyper-parameter Setting. We use 4 clients by default, each holding a privacy budget of $\epsilon = 10$ and $\delta = 10^{-6}$. The NF model consists of 10 coupling layers [15], each containing two hidden layers with 256 units. The local training in FL uses the SGD optimizer with a momentum of 0.9 and a batch size of 512. The learning rate is selected by grid search for all methods with cosine learning rate decay. The FL training consists of 40 rounds, with the server aggregating client models after each epoch of local training, *i.e.,* $E = 1$. ADAPT employs a mask selection parameter $K = 13$, masking approximately 5% of all network connections for aligning neurons. ADAPT generates 10% of the original dataset, *i.e.,* $0.1N$, for fine-tuning. During inference, adaptive importance sampling is executed with 100 buckets, and the sampling process continues until convergence.

Evaluation Metrics. We measure both the query accuracy and latency. To measure accuracy, we adopt the relative error (RE) [20, 39] over the query workloads of different aggregate queries. Defined as RE = $\frac{|Estimation-GroudTruth|}{GroundTruth}$, it compares the estimated value of the aggregate query (*Estimation*) against the actual value (*GroundTruth*). To thoroughly evaluate accuracy, we report different quantiles of the relative errors, including 50% (Median), 95%, 99% and 100% (Max). Additionally, for NF training, we also evaluate the log-likelihood (LL) of the trained models over the test set, which is a common metric for evaluating the accuracy of density models [12, 15, 44]. For latency, we report the average query latency as well as the end-to-end training time for NF model-based methods.

Workloads for Testing. For each dataset, we generate 2,000 queries for each aggregate function, including COUNT, SUM, AVERAGE, and VARIANCE. The multidimensional aggregate queries incorporate both range and equality predicates. Following existing works [52, 56], the queries are generated by: (1) Randomly select a numerical column to place the aggregate function. (2) Randomly select the number of predicates, denoted as f. (3) Randomly select f distinct columns to place the predicates. For numerical columns, predicates are uniformly selected from $\{=, \leq, \geq\}$. For categorical columns, only equality predicates are generated because range predicates on categorical attributes have no practical meaning. (4) Randomly select a tuple from the dataset and use its attributes as the literals for the predicates. In addition, we randomly select 10% of the queries that have categorical predicates and change one of the categorical predicates to GROUP BY predicate to also evaluate the GROUP BY queries. The selectivity distributions of queries for different datasets are shown in Figure 5.

6.2 Overall Evaluation

6.2.1 Comparison of Accuracy. Table 2 shows the relative errors of different methods on different aggregate queries and the test log-likelihoods for methods using NF models. Methods are grouped by category. The results could be ranked as centralized training > federated training > data synthesis > output perturbation, where the accuracy of centralized training serves as a reference of the upper bound for FL methods.

Next, we explain the results. The accuracy of ADAPT is very high on all aggregate queries over all datasets with different characteristics. As Table 2 shows, ADAPT outperforms all other federated training methods on both test LL and the entire distribution of relative errors on all datasets. For example, the medians of COUNT, SUM, AVERAGE and VARIANCE of ADAPT on Power (0.05, 0.06, 0.04, 0.06) are close to the optimal CentralDP that trains in a centralized manner with DP (0.04, 0.06, 0.04, 0.05). Especially, ADAPT also performs well on errors at the tail (99th, Max). For example, at the Max-quantile of COUNT queries on IMDB dataset, ADAPT outperforms other federated training methods by up to 251×. As a consensus [56], errors at the tail indicate the stability of estimators and are harder to optimize than the medians. Therefore, the results further demonstrate that ADAPT is a *well-performing yet stable* estimator. This is because our framework could train a more accurate density model compared to other FL methods (ADAPT obtains the highest test LL on all datasets compared to other FL-based methods).

ADAPT performs better than FedAVG because ADAPT uses an optimized parameter-aligned model structure and a customized model fine-tuning algorithm during model fusion, which effectively improves the training accuracy in FL. ADAPT outperforms the matching-based model fusion methods including FedMA and GAMF to a large extent. For example, on BJAQ dataset, the model trained by ADAPT achieves a test LL of -3.38 and a maximum relative error of 0.49 for COUNT queries, while FedMA and GAMF only achieve -4.38, 210 and -4.42, 195 respectively. The reason is that the noise added to gradients generally misleads the matching process in FedMA and GAMF, thereby degrading the accuracy of their trained models.

ADAPT outperforms FedPAN. On Power, the max relative error for AVERAGE queries of ADAPT is 0.58, while that of FedPAN is 35.3. This is because FedPAN only perturbs the neuron outputs with a periodic function, which is not effective in aligning neurons.

ADAPT also outperforms methods based on output perturbation (DPSQL and FLEX) in accuracy by up to 4 magnitudes. The reason is that these methods have to assign a tiny privacy budget to each of the queries. It thus has to add a large amount of noise to each query result, inevitably leading to poor accuracy. ADAPT obtains higher accuracy compared with PrivMRF and DP-WGAN, because synthesized data is only an inaccurate approximation of the learned distribution, and can lead to catastrophic errors when none of the synthesized data satisfies the query, *i.e.,* the 0-tuple problem [47].

6.2.2 Comparison of Efficiency. We evaluate the training time of the methods that train density models as well as the inference latency of different types of approaches.

Training Time. We report the training time of different federated training methods on Power and Flights in Figure 6(a). We could see that the training time of ADAPT (about 1 hour) is almost the same as the conventional FedAVG and applicable in practice. ADAPT is faster than FedMA and GAMF (2-5 hours), because FedMA and GAMF need to iteratively match the network parameters before each model fusion, which is time-consuming. In addition, ADAPT is only a little slower than FedAVG and FedPAN. The reason is that though ADAPT introduces the element-wise mask multiplication to the network and introduces the fine-tuning process, the additional computation could be computed efficiently and almost brings no overheads.

Inference Latency. We report the average inference latency of different methods on Power and Flights of COUNT queries in Figure 6(b). The methods are grouped into 4 categories: Density (ADAPT) denoting all methods that estimate query results with density models, Synthetic denoting all data synthesis methods (PrivMRF and DP-WGAN) and the conventional output perturbation methods DPSQL and FLEX. We could observe that Density has

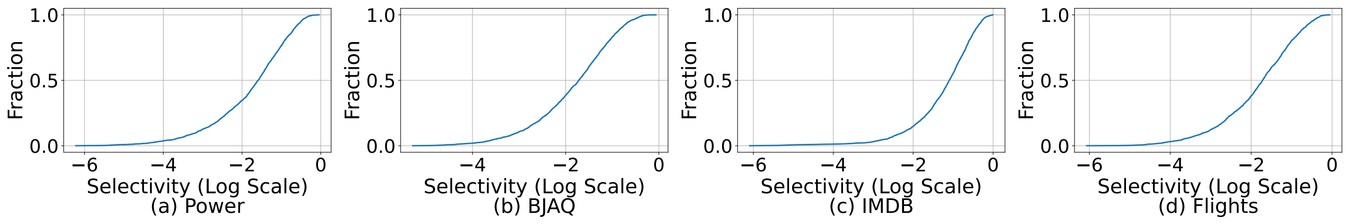

Figure 5: Distribution of Query Selectivity

Table 2: Test log-likelihood (the higher the better), relative errors (the lower the better) on different aggregate queries of different methods on 4 datasets. The results of `CentralDP` and `Central` are in gray since they are infeasible to protect privacy and their results only serve as a reference to evaluate the gap between models trained in FL and in a centralized way. "-" denotes unsupported metrics or queries.

Dataset	Method	Test LL	COUNT				SUM				AVERAGE				VARIANCE			
			50th	95th	99th	Max	50th	95th	99th	Max	50th	95th	99th	Max	50th	95th	99th	Max
Power	DPSQL	-	183	$5.1e^3$	$1.0e^4$	$1.4e^4$	$1.3e^3$	$2.1e^4$	$4.8e^4$	$6.0e^4$	89.1	301	$1.2e^3$	$2.3e^3$	658	$4.2e^3$	$9.8e^3$	$5.9e^4$
	FLEX	-	617	$8.9e^3$	$1.5e^4$	$2.1e^4$	-	-	-	-	-	-	-	-	-	-	-	-
	PrivMRF	-	1.91	9.45	141	573	1.53	11.2	82.1	290	1.12	5.91	20.9	103	1.89	13.4	105	674
	DP-WGAN	-	2.88	29.1	510	$1.1e^3$	2.34	15.1	160	401	1.33	9.25	31.5	126	2.01	19.5	$1.0e^3$	$1.2e^3$
	FedAVG	0.13	0.10	1.39	3.58	134	0.12	2.46	5.13	275	0.11	2.49	4.95	34.7	0.13	8.56	13.3	48.2
	FedMA	-0.08	0.97	9.04	17.5	274	0.82	9.66	30.2	471	0.69	3.00	5.93	60.6	1.83	14.8	121	591
	GAMF	-0.07	0.94	8.65	14.9	205	0.80	8.18	31.1	381	0.77	4.71	7.38	50.1	1.81	15.1	119	480
	FedPAN	0.14	0.10	1.35	3.46	105	0.12	3.43	4.75	255	0.10	2.44	5.04	35.3	0.13	8.97	11.5	39.8
	ADAPT	**0.21**	**0.05**	**0.34**	**0.71**	**0.88**	**0.06**	**0.37**	**0.84**	**1.09**	**0.04**	**0.40**	**0.53**	**0.58**	**0.06**	**0.43**	**1.38**	**3.29**
	CentralDP	0.24	0.04	0.31	0.68	0.84	0.06	0.35	0.81	1.05	0.04	0.38	0.47	0.50	0.05	0.39	1.29	3.15
	Central	0.64	0.02	0.07	0.16	0.34	0.03	0.23	0.41	0.64	0.01	0.12	0.21	0.24	0.03	0.24	0.50	0.76
IMDB	DPSQL	-	12.7	$1.1e^3$	$2.3e^3$	$1.4e^4$	24.7	$3.1e^4$	$2.0e^5$	$8.6e^5$	9.10	39.1	384	$1.2e^3$	9.81	$1.2e^3$	$7.3e^3$	$1.5e^4$
	FLEX	-	20.1	$1.3e^3$	$4.7e^3$	$2.0e^4$	-	-	-	-	-	-	-	-	-	-	-	-
	PrivMRF	-	0.31	1.66	876	$5.7e^3$	1.09	6.34	816	$8.5e^3$	0.94	4.26	6.91	57.5	1.32	6.49	30.1	375
	DP-WGAN	-	0.45	2.49	$1.2e^3$	$8.1e^3$	1.42	18.0	$1.2e^3$	$3.8e^4$	1.56	10.3	20.5	98.2	2.01	104	851	$3.8e^3$
	FedAVG	-4.02	0.10	0.96	1.80	110	0.12	0.54	1.89	37.5	0.05	0.47	0.82	1.19	0.27	0.87	18.8	59.4
	FedMA	-4.57	0.49	2.05	83.1	239	0.44	5.18	33.1	185	0.31	0.93	8.83	84.1	1.44	8.39	41.5	391
	GAMF	-4.28	0.44	3.92	69.0	198	0.37	4.85	32.8	147	0.29	0.88	7.61	52.5	1.39	7.41	39.9	402
	FedPAN	-4.01	0.10	0.85	1.59	79.5	0.12	0.52	1.63	74.0	0.05	0.38	0.85	1.15	0.26	0.74	8.22	30.6
	ADAPT	**-3.94**	**0.05**	**0.39**	**0.81**	**0.95**	**0.09**	**0.26**	**0.97**	**1.25**	**0.03**	**0.12**	**0.26**	**0.36**	**0.22**	**0.54**	**0.88**	**5.82**
	CentralDP	-3.67	0.04	0.31	0.72	0.95	0.08	0.24	0.88	1.11	0.03	0.10	0.23	0.34	0.21	0.48	0.81	5.77
	Central	-2.98	0.03	0.18	0.32	0.53	0.04	0.17	0.45	0.74	0.02	0.05	0.13	0.19	0.11	0.21	0.47	2.89
BJAQ	DPSQL	-	208	656	$1.5e^3$	$2.9e^3$	374	$3.7e^3$	$4.6e^3$	$1.1e^4$	67.2	174	245	258	581	974	$3.0e^3$	$5.4e^3$
	FLEX	-	491	851	$1.8e^3$	$3.6e^3$	-	-	-	-	-	-	-	-	-	-	-	-
	PrivMRF	-	0.25	0.71	0.86	10.3	0.29	1.32	2.82	6.91	0.09	1.31	3.10	5.75	0.23	4.11	42.4	$1.2e^3$
	DP-WGAN	-	0.33	1.05	3.41	38.2	0.45	2.56	8.74	38.9	0.35	2.85	5.39	9.74	0.32	5.00	64.1	$1.5e^3$
	FedAVG	-3.51	0.13	0.29	0.86	4.49	0.14	0.45	0.67	2.34	0.07	0.28	0.80	1.39	0.15	2.41	34.6	863
	FedMA	-4.38	0.49	3.05	18.2	210	0.51	4.97	28.0	209	0.38	0.91	1.92	63.1	1.64	5.81	48.2	$2.6e^3$
	GAMF	-4.42	0.53	3.18	16.9	195	0.50	4.38	30.1	391	0.34	0.81	1.47	57.0	1.39	5.96	41.3	$2.5e^3$
	FedPAN	-3.43	0.13	0.24	0.79	4.63	0.14	0.41	0.59	1.82	0.06	0.24	0.61	1.11	0.15	1.21	38.1	$1.3e^3$
	ADAPT	**-3.38**	**0.09**	**0.19**	**0.39**	**0.49**	**0.10**	**0.24**	**0.30**	**0.35**	**0.04**	**0.15**	**0.24**	**0.33**	**0.13**	**0.41**	**1.44**	**5.22**
	CentralDP	-3.30	0.08	0.17	0.36	0.47	0.09	0.22	0.29	0.30	0.03	0.15	0.20	0.29	0.12	0.36	1.22	4.81
	Central	-2.77	0.02	0.07	0.09	0.28	0.02	0.07	0.12	0.19	0.01	0.07	0.12	0.15	0.07	0.30	0.79	2.57
Flights	DPSQL	-	5.26	385	$1.5e^3$	$4.8e^4$	6.28	778	$9.4e^3$	$8.9e^4$	3.46	49.5	482	$1.0e^3$	4.79	840	$3.8e^3$	$4.9e^3$
	FLEX	-	6.80	809	$2.5e^3$	$8.4e^4$	-	-	-	-	-	-	-	-	-	-	-	-
	PrivMRF	-	0.32	1.75	6.70	73.7	0.33	1.84	6.98	44.4	0.12	0.57	0.83	8.47	0.51	4.75	34.8	249
	DP-WGAN	-	0.37	2.08	8.74	102	0.40	3.85	10.4	80.6	0.28	2.51	7.47	37.7	0.74	7.85	49.5	$1.4e^3$
	FedAVG	-4.79	0.14	0.65	0.85	17.1	0.24	0.67	0.97	5.99	0.10	0.38	0.84	9.67	0.33	3.31	14.7	146
	FedMA	-6.01	0.48	9.85	15.9	103	0.68	5.49	9.44	84.2	0.47	4.89	8.41	38.4	1.04	8.50	51.0	$2.1e^3$
	GAMF	-5.82	0.52	3.81	3.57	95.4	0.62	2.34	7.57	70.9	0.49	2.23	6.33	20.9	1.32	5.50	27.4	$1.0e^3$
	FedPAN	-4.75	0.14	0.62	0.81	13.3	0.24	0.65	0.87	16.3	0.14	0.43	0.85	13.4	0.31	2.21	56.1	186
	ADAPT	**-4.69**	**0.11**	**0.42**	**0.65**	**1.77**	**0.16**	**0.42**	**0.75**	**2.40**	**0.07**	**0.31**	**0.54**	**0.97**	**0.23**	**1.87**	**8.87**	**9.85**
	CentralDP	-4.61	0.10	0.42	0.65	1.60	0.15	0.41	0.61	2.18	0.07	0.28	0.53	0.91	0.22	1.69	8.22	9.47
	Central	-4.00	0.07	0.27	0.40	0.81	0.10	0.33	0.48	1.29	0.03	0.20	0.35	0.53	0.10	1.01	1.80	3.10

the smallest inference latency, and performs similarly for datasets of different sizes. For example, on `Flights`, `Density` has an inference latency of 19 ms, much smaller than other methods (about 145

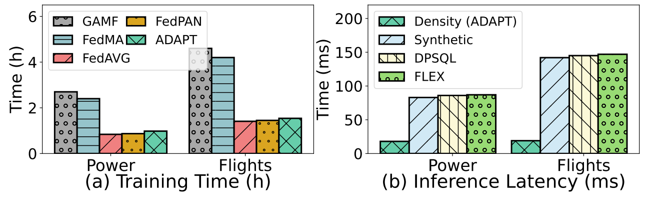

Figure 6: Efficiency evaluation of different methods.

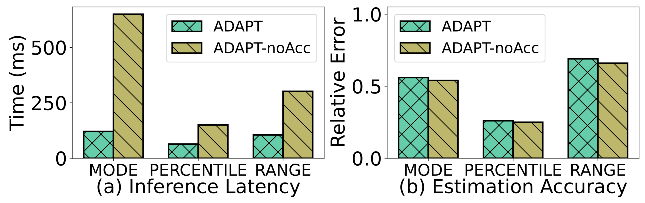

Figure 7: Evaluation of inference acceleration.

ms). The reason is that `Density` could answer queries by directly accessing the density model, while other methods need to execute the queries on the original or synthesized data.

6.3 Evaluation of Inference Acceleration

To evaluate the acceleration algorithms proposed in Section 5.3, we measure the latency of `MODE` queries (for Early Stop), `PERCENTILE` and `RANGE` queries (for Reuse Samples). We conduct the experiments over `IMDB` using the same model as the experiments in Section 6.2 and 1,000 queries for each aggregate function. The accuracy (95%-quantile relative error) is also reported for a thorough evaluation. The results are shown in Figure 7, where `ADAPT-noAcc` denotes `ADAPT` without the acceleration optimizations. In comparison to the non-optimized counterpart, our optimizations reduce the latency by 2-5 times, without sacrificing accuracy at all. This is because `ADAPT` avoids redundant samples that do not influence estimation accuracy.

6.4 Varying Privacy Budget

We vary the privacy budget ϵ for each client from 1 to 100 to evaluate its influence. We compare `ADAPT` with the two most well-performing FL-based methods, `FedAVG` and `FedPAN`, and the centralized DP method `CentralDP`. The results are in Figures 8(a)-(b). On `Power`, as the privacy budget increases, the 95%-quantile relative error of `ADAPT` decreases from about 0.59 to 0.28, consistently close to the optimal `CentralDP`. This indicates that `ADAPT` performs well under various privacy budgets, and is accurate even in scenarios with limited privacy budgets (*e.g.,* only 1). This is because even in the case of local private training, the aligned parameters and the fine-tuning in `ADAPT` effectively improve the accuracy of model fusion, thus finally obtaining a more accurate model.

6.5 Varying Number of Clients

We vary the number of clients from 2 to 128 to evaluate its influence on estimation accuracy. As shown in Figures 8(c)-(d), as the number of clients increases, the 95%-quantile relative error of `ADAPT` increases from about 0.35 to 1.09 on `IMDB`. However, the relative errors of `FedAVG` and `FedPAN` increase from about 0.6 to 2.8. `ADAPT` performs well with various numbers of clients since its effective parameter alignment is independent of the number of clients.

6.6 Varying Number of Queries

As explained in Section 1, the accuracy of output perturbation methods is influenced by the number of queries due to the cumulative consumption of privacy budget. To investigate its influence, we compare `ADAPT` with `DPSQL` and `FLEX` varying the number of queries. Figures 8(e)-(f) show that when the number of queries is small, all methods obtain accurate results. However, as the number of queries increases, output perturbation methods exhibit a significant increase in errors. In contrast, `ADAPT` maintains high accuracy, which indicates that `ADAPT` can estimate accurately regardless of the number of queries by employing the privacy-preserving density model.

6.7 Varying the Proportion of Removed Neural Connections

We vary the proportion (ρ) of removed neural connections of the NF model and report the 95%-quantile relative errors of `ADAPT` for answering `SUM` queries in Figure 9. As ρ increases, the relative error first decreases and then increases. The reason is that, initially, as more connections are removed, the neurons in the same layer gradually become distinguishable. Therefore, `ADAPT` achieves better parameter alignment, and the removal of a small number of connections has negligible impact on the capability of the network. However, when too many connections are removed, the accuracy decreases due to the decreased expressive capability of the network. We can also observe that, in general, 2% to 15% is a reasonable range for ρ that can achieve a balance between parameter alignment and network expressive ability.

6.8 Varying Client Data Size Imbalance

We also evaluate the effect of client data size imbalance. Specifically, we use Dirichlet distribution $Dir_c(\beta)$ to allocate different amounts of data points to the c clients following existing works [33, 61]. The parameter β controls the imbalance level and a smaller β denotes a higher imbalance level. We sample a vector $\boldsymbol{B} \sim Dir_c(\beta)$ and allocate client E_j with a $\boldsymbol{B}_j$ proportion of the total data. Figure 10 shows the 99%-quantile relative errors for `SUM` queries with different β. The results show that data size imbalance generally has a small effect on methods based on density models. In addition, `ADAPT` outperforms other methods with all β with stable performance. The reason is that the weighted factors in the parameter average of FL [33] and `DA` already consider the client data size, which makes `ADAPT` able to handle imbalanced client data sizes accurately.

6.9 Data Update Evaluation

The local dataset on the client side may undergo updates, such as adding new data. A straightforward solution is to train a new model from scratch. However, this significantly increases privacy leakage, as more privacy budgets are needed for retraining. To mitigate this issue, we propose an incremental training method to update the model with minimal additional privacy budgets. In the evaluation, we partition `Power` into two parts based on a time attribute. The first part is used as the original data while the second part is treated as new data. We compare three different training strategies: `ADAPT-Retrain`: train a new model with an additional privacy budget $\epsilon = 10$ for each client. `ADAPT-Inc`: incrementally train the stale model on the updated data with an additional privacy budget of $\epsilon = 1$. `ADAPT-Stale`: use the stale model without any updates. We evaluate these models on the updated dataset. As shown in Figure 11(a),

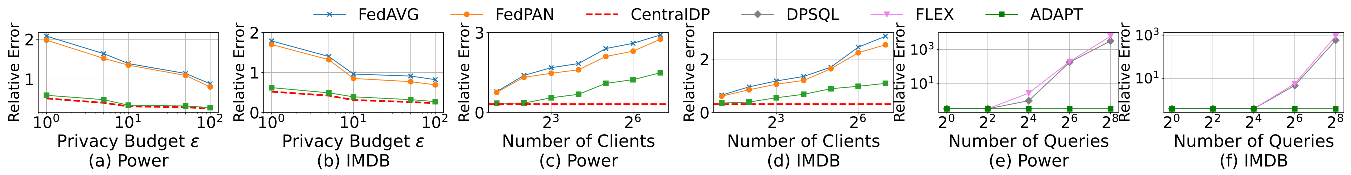
Figure 8: Evaluation of varying (a, b): privacy budget; (c, d): number of clients; (e, f): number of queries with COUNT queries.

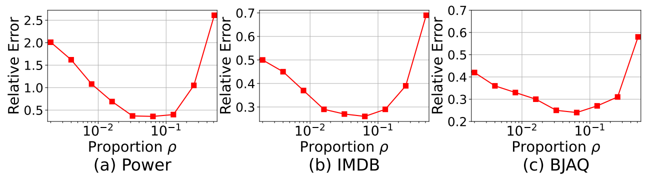
Figure 9: Evaluation of varying removed connection proportion.

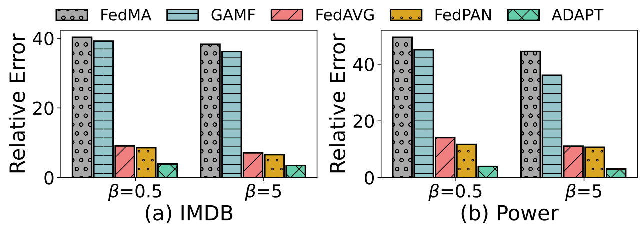
Figure 10: Evaluation of varying client data size imbalance level.

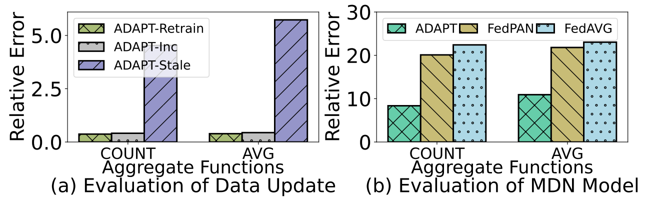
Figure 11: Evaluation of: (a) data update; (b) MDN model.

ADAPT-Inc obtains almost the same accuracy as ADAPT-Retrain while consuming much fewer privacy budgets.

6.10 Other Density Models

The optimizations proposed in this paper are applicable to other density models. To show this, we conduct experiments using Mixture Density Networks (MDN), which are widely adopted in distribution modeling [8, 38]. The output of MDN models is a mixture of Gaussians, described as $p(\boldsymbol{x}) = \sum_{i=1}^{m} w_i \cdot \mathcal{N}(\mu_i, \sigma_i)$, where neural networks learn the parameters w, μ and σ. We train MDN models with ten Gaussians, parameterized by five-layer fully connected networks with 256 hidden neurons and ReLU activations. The models are evaluated using BJAQ on COUNT and AVG queries. Figure 11(b) shows the 95%-quantile relative error. The results show that training with our ADAPT framework significantly reduces errors. This confirms the effectiveness of ADAPT in training other density models.

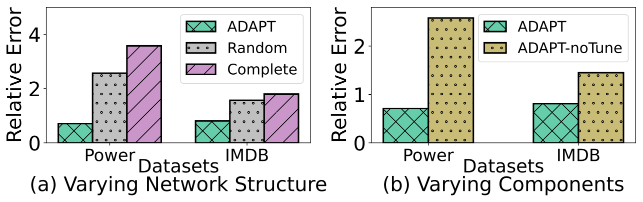
Figure 12: Effectiveness evaluation of proposed techniques.

6.11 Ablation Study

We evaluate the proposed optimization techniques including network structure selection and fine-tuning. We evaluate by COUNT queries on Power and IMDB and report the 99%-quantile relative errors.

6.11.1 Network Structure Selection. We evaluate the performance of different methods for selecting network structures, including the one in Algorithm 1 (ADAPT), random selection (Random), and not removing connections (Complete). Both ADAPT and Random remove $\rho = 5\%$ of connections. As shown in Figure 12(a), ADAPT obtains more accurate results compared with Random and Complete. This is because ADAPT judiciously selects optimal neural connections, thus obtaining a better alignment and higher accuracy.

6.11.2 Fine-tuning. We evaluate our fine-tuning method (Section 4) that refines the NF model. ADAPT-noTune denotes the method without fine-tuning, *i.e.*, simply using coordinate-wise averaging. As shown in Figure 12(b), fine-tuning improves the estimation accuracy. The reason is that the error-bounded DA can adjust the averaged model towards a more accurate distribution.

7 RELATED WORK

Differential Privacy. Differential Privacy (DP) [16] offers a mathematical framework to quantify privacy loss when analyzing sensitive data. The key idea is to ensure that any data analysis outcome does not change significantly, regardless of whether individual data is included. To train deep neural networks with DP guarantee, differentially private stochastic gradient descent (DP-SGD) [1] is commonly employed by perturbing the gradients with noise. In FL, to protect the privacy of client data, *sample-level* [1] DP is generally adopted. It enables each client to perturb its local gradients using DP-SGD based on its own DP budget and transfer noised model parameters to the server. Sample-level DP guarantees that both intermediate and final model parameters are indistinguishable regardless of the presence or absence of any individual data record at any client.

Federated Learning. Federated learning (FL) [40] enables multiple clients to collaboratively train a global model, without sharing raw data. FL significantly enhances model accuracy through iterative model fusion compared to one-shot fusion [36, 45], where models trained separately for clients are merged. However, model accuracy still declines compared to centralized training that aggregates data from all clients [34]. Recent studies [36, 51, 57] suggest this decline is mainly due to *parameter misalignment* during server-side model fusion. In FL, different clients may have updated model parameters that are not aligned by coordinates due to the *permutation invariance* [51, 57, 58] property of neural networks. This misalignment leads to conflicts in parameter averaging and decreases accuracy.

Extensive research [36, 51, 57] has tried to address this issue, employing graph matching algorithms to align network neurons before averaging. However, the effectiveness of these methods degrades significantly when using DP-SGD in local training because the added noise for privacy preservation tends to mislead the matching.

Data Synthesis. Some existing methods use data synthesis to approximate query results with protected data privacy. These methods employ a model (*e.g.,* Markov Random Fields [10], VAE [49], GAN [2, 3, 27] and Bayesian Networks [59, 60]) to learn the original data distribution and generate synthesized data. In this way, we can directly query the synthesized data, rather than querying the raw private data. However, these methods are not accurate enough because the synthesized data is likely to have a large variance with the learned distribution, which always degrades the accuracy. For example, it may face the 0-tuple problem [47] that no synthesized data satisfies the query, which is likely to cause catastrophic errors.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose ADAPT, which enables federated data analytics with high accuracy regardless of query number while preserving data privacy. ADAPT leverages a privacy-preserving density model trained using a specially designed neural network structure for parameter alignment and a specialized training algorithm. This model is then used to answer queries directly without requiring access to raw data. ADAPT significantly improves the accuracy of federated data analytics compared to existing methods, especially when a large number of queries are involved. In the future, we plan to enhance ADAPT to support multi-table scenarios. This is challenging, as it has to capture the correlations among multiple tables.

REFERENCES

[1] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 308–318. ACM, 2016.

[2] M. Alzantot and M. Srivastava. Differential Privacy Synthetic Data Generation using WGANs, 2019.

[3] S. Augenstein, H. B. McMahan, D. Ramage, S. Ramaswamy, P. Kairouz, M. Chen, R. Mathews, and B. A. y Arcas. Generative models for effective ML on private, decentralized datasets. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[4] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers. Shrinkwrap: Efficient SQL query processing in differentially private data federations. *Proc. VLDB Endow.*, 12(3):307–320, 2018.

[5] J. Bater, Y. Park, X. He, X. Wang, and J. Rogers. SAQE: practical privacy-preserving approximate query processing for data federations. *Proc. VLDB Endow.*, 13(11):2691–2705, 2020.

[6] S. Z. Béguelin, L. Wutschitz, S. Tople, V. Rühle, A. Paverd, O. Ohrimenko, B. Köpf, and M. Brockschmidt. Analyzing information leakage of updates to natural language models. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 363–375. ACM, 2020.

[7] Beijing Multi-Site Air-Quality Data Data Set. https://archive.ics.uci.edu/dataset/501/beijing+multi+site+air+quality+data, 2024. Last accessed: 2024-06-23.

[8] C. M. Bishop. Mixture density networks. 1994.

[9] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[10] K. Cai, X. Lei, J. Wei, and X. Xiao. Data synthesis via differentially private markov random fields. *Proceedings of the VLDB Endowment*, 14(11):2190–2202, 2021.

[11] F. Delays and C. D. Set. https://www.kaggle.com/usdot/flight-delays, 2024. Last accessed: 2024-06-23.

[12] L. Dinh, D. Krueger, and Y. Bengio. NICE: non-linear independent components estimation. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[13] I. Dinur and K. Nissim. Revealing information while preserving privacy. In F. Neven, C. Beeri, and T. Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 202–210. ACM, 2003.

[14] W. Dong, J. Fang, K. Yi, Y. Tao, and A. Machanavajjhala. R2T: instance-optimal truncation for differentially private query evaluation with foreign keys. In Z. G. Ives, A. Bonifati, and A. E. Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 759–772. ACM, 2022.

[15] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7509–7520, 2019.

[16] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[17] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[18] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[19] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep learning models for selectivity estimation of multi-attribute queries. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1035–1050. ACM, 2020.

[20] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! *Proc. VLDB Endow.*, 13(7):992–1005, 2020.

[21] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2722–2730. PMLR, 2019.

[22] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[23] E. Hoogeboom, T. S. Cohen, and J. M. Tomczak. Learning discrete distributions by dequantization. *CoRR*, abs/2001.11235, 2020.

[24] I. household electric power consumption data set. https://github.com/gpapamak/maf, 2024. Last accessed: 2024-06-24.

[25] B. Jiang, J. Pei, Y. Tao, and X. Lin. Clustering uncertain data based on probability distribution similarity. *IEEE Trans. Knowl. Data Eng.*, 25(4):751–763, 2013.

[26] N. M. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for SQL queries. *Proc. VLDB Endow.*, 11(5):526–539, 2018.

[27] J. Jordon, J. Yoon, and M. van der Schaar. PATE-GAN: generating synthetic data with differential privacy guarantees. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[28] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[29] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[30] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015.

[31] G. P. Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192–203, 1978.

[32] G. P. Lepage. Adaptive multidimensional integration: vegas enhanced. *J. Comput. Phys.*, 439:110386, 2021.

[33] Q. Li, Y. Diao, Q. Chen, and B. He. Federated learning on non-iid data silos: An experimental study. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 965–978. IEEE, 2022.

[34] X. Li, F. Tramèr, P. Liang, and T. Hashimoto. Large language models can be strong differentially private learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[35] X. Li, Y. Xu, S. Song, B. Li, Y. Li, Y. Shao, and D. Zhan. Federated learning with position-aware neurons. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10072–10081. IEEE, 2022.

[36] C. Liu, C. Lou, R. Wang, A. Y. Xi, L. Shen, and J. Yan. Deep neural network fusion via graph matching with applications to model ensemble and federated learning. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 13857–13869. PMLR, 2022.

[37] T. Luo, M. Pan, P. Tholoniat, A. Cidon, R. Geambasu, and M. Lécuyer. Privacy budget scheduling. In A. D. Brown and J. R. Lorch, editors, *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*, pages 55–74. USENIX Association, 2021.

[38] Q. Ma, A. M. Shanghooshabad, M. Almasi, M. Kurmanji, and P. Triantafillou. Learned approximate query processing: Make it light, accurate and fast. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org, 2021.

[39] Q. Ma and P. Triantafillou. Dbest: Revisiting approximate query processing engines with machine learning models. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1553–1570. ACM, 2019.

[40] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In A. Singh and X. J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[41] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 19–30. ACM, 2009.

[42] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5):145:1–145:19, 2019.

[43] G. Papamakarios, I. Murray, and T. Pavlakou. Masked autoregressive flow for density estimation. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2338–2347, 2017.

[44] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1530–1538. JMLR.org, 2015.

[45] S. P. Singh and M. Jaggi. Model fusion via optimal transport. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[46] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2256–2265. JMLR.org, 2015.

[47] J. Sun and G. Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, 2019.

[48] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[49] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing for data exploration using deep generative models. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1309–1320. IEEE, 2020.

[50] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[51] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, and Y. Khazaeni. Federated learning with matched averaging. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[52] J. Wang, C. Chai, J. Liu, and G. Li. FACE: A normalizing flow based cardinality estimator. *Proc. VLDB Endow.*, 15(1):72–84, 2021.

[53] J. Wang, C. Chai, J. Liu, and G. Li. Cardinality estimation using normalizing flow. *VLDB J.*, 33(2):323–348, 2024.

[54] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. Differentially private SQL with bounded user contribution. *Proc. Priv. Enhancing Technol.*, 2020(2):230–250, 2020.

[55] S. Wu, Y. Cui, J. Yu, X. Sun, T. Kuo, and C. J. Xue. NFL: robust learned index via distribution transformation. *Proc. VLDB Endow.*, 15(10):2188–2200, 2022.

[56] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep unsupervised cardinality estimation. *Proc. VLDB Endow.*, 13(3):279–292, 2019.

[57] F. Yu, W. Zhang, Z. Qin, Z. Xu, D. Wang, C. Liu, Z. Tian, and X. Chen. Fed2: Feature-aligned federated learning. In F. Zhu, B. C. Ooi, and C. Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 2066–2074. ACM, 2021.

[58] M. Yurochkin, M. Agarwal, S. Ghosh, K. H. Greenewald, T. N. Hoang, and Y. Khazaeni. Bayesian nonparametric federated learning of neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7252–7261. PMLR, 2019.

[59] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: private data release via bayesian networks. In C. E. Dyreson, F. Li, and M. T. Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1423–1434. ACM, 2014.

[60] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Trans. Database Syst.*, 42(4):25:1–25:41, 2017.

[61] H. Zhu, J. Xu, S. Liu, and Y. Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 2021.