Privacy-Enhanced Database Synthesis for Benchmark Publishing

Yunqing Ge
Shenzhen University
geyunqing2022@email.szu.edu.cn

Jianbin Qin*
SICS, Shenzhen University
qinjianbin@szu.edu.cn

Shuyuan Zheng*
Osaka Univeristy
zheng@ist.osaka-u.ac.jp

Yongrui Zhong
Shenzhen University
zhongyongrui2021@email.szu.edu.cn

Bo Tang
Southern University of Science and Technology
tangb3@sustech.edu.cn

Yu-Xuan Qiu
Beijing Institute of Technology
qiuyx.cs@gmail.com

Rui Mao
SICS, Shenzhen University
mao@szu.edu.cn

Ye Yuan
Beijing Institute of Technology
yuan-ye@bit.edu.cn

Makoto Onizuka
Osaka Univeristy
onizuka@ist.osaka-u.ac.jp

Chuan Xiao
Osaka Univeristy, Nagoya University
chuanx@ist.osaka-u.ac.jp

ABSTRACT

Benchmarking is crucial for evaluating a DBMS, yet existing benchmarks often fail to reflect the varied nature of user workloads. As a result, there is increasing momentum toward creating databases that incorporate real-world user data to more accurately mirror business environments. However, privacy concerns deter users from directly sharing their data, underscoring the importance of creating synthesized databases for benchmarking that also prioritize privacy protection. Differential privacy (DP)-based data synthesis has become a key method for safeguarding privacy when sharing data, but the focus has largely been on minimizing errors in aggregate queries or downstream ML tasks, with less attention given to benchmarking factors like query runtime performance. This paper delves into differentially private database synthesis specifically for benchmark publishing scenarios, aiming to produce a synthetic database whose benchmarking factors closely resemble those of the original data. Introducing *PrivBench*, an innovative synthesis framework based on sum-product networks (SPNs), we support the synthesis of high-quality benchmark databases that maintain fidelity in both data distribution and query runtime performance while preserving privacy. We validate that PrivBench can ensure database-level DP even when generating multi-relation databases with complex reference relationships. Our extensive experiments show that PrivBench efficiently synthesizes data that maintains privacy and excels in both data distribution similarity and query runtime similarity.

PVLDB Reference Format:
Yunqing Ge, Jianbin Qin, Shuyuan Zheng, Yongrui Zhong, Bo Tang, Yu-Xuan Qiu, Rui Mao, Ye Yuan, Makoto Onizuka, and Chuan Xiao. Privacy-Enhanced Database Synthesis for Benchmark Publishing. PVLDB, 18(2): 413 - 425, 2024.
doi:10.14778/3705829.3705855

*Corresponding authors.

Proceedings of the VLDB Endowment, Vol. 18, No. 2 ISSN 2150-8097.
doi:10.14778/3705829.3705855

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

1 INTRODUCTION

Benchmarking, a crucial component for evaluating the performance of DBMSs, has historically leveraged established benchmarks such as the TPC series [2]. These conventional benchmarks use fixed schemas and queries to compare the performance of different database systems standardizedly. However, they may fall short in representing the varied, specific workloads and data characteristics unique to every user. Moreover, the nuances of real-world applications, intrinsic data characteristics, and user-specific performance expectations might not be fully captured by these fixed benchmarks, showing a need for benchmarks more tailored to individual users.

For benchmark publishing, synthesizing a database is a highly challenging endeavor because it requires attention to four concerns: (1) fidelity of data distribution, (2) fidelity of query runtime performance, (3) privacy protection, and (4) synthesis efficiency. The first two concerns ensure accurate benchmarking, privacy protection accommodates compliance with data protection regulations, and synthesis efficiency facilitates rapid updates of enterprise-level benchmarks. Some database synthesis efforts, such as SAM [44], can produce high-fidelity benchmarks but overlook privacy protection. Such oversight can lead to inapplicability in contexts where data privacy is paramount, e.g., those involving user data. On the other hand, existing privacy-preserving data synthesis methods (e.g., [3, 4, 26, 28, 29, 32, 34, 46, 47]) only focus on enhancing the fidelity in data distribution while overlooking the fidelity in query runtime performance, thereby far from meeting the practical demands of benchmarking scenarios.

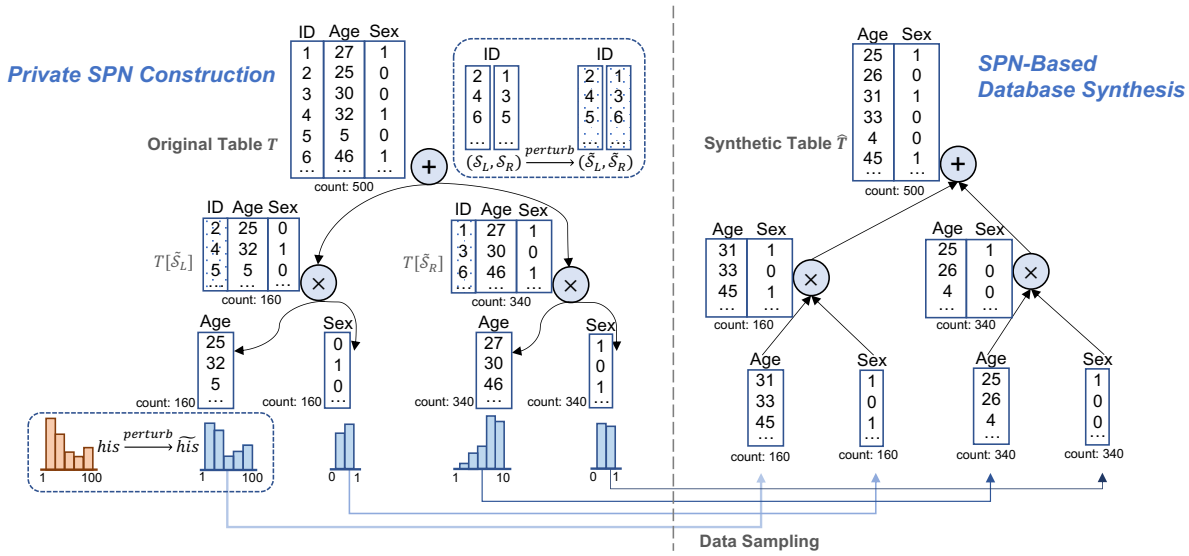


Figure 1: Learning a single-relation SPN and generating data based on the SPN.

In this paper, we propose PrivBench, a database synthesis framework for benchmark publishing that addresses all four of the aforementioned concerns. The design of PrivBench is based on our novel observation: *sum-product networks* [33], when used as data synthesis models, can simultaneously ensure the fidelity of data distribution and query runtime while offering linear-time efficiency for data sampling. However, SPNs do not provide privacy protection. Therefore, PrivBench combines SPNs with differential privacy (DP) [8], a widely-adopted privacy technique, to construct privacy-preserving SPNs. Using DP-based SPNs, PrivBench efficiently synthesizes high-fidelity databases, protecting user privacy and reducing the compromise on quality in benchmarking scenarios. In particular, PrivBench constructs SPNs on the input database instance, which is equivalent to partitioning the records of each relational table into disjoint blocks and building histograms within each block. Additionally, we connect the per-table SPNs based on reference relationships to form a multi-SPN model. We inject noise into all partitioning operations, histograms, and SPN connections to guarantee DP for data synthesis through the multi-SPN.

Example 1. Figure 1 illustrates the process of constructing an SPN for a single table and synthesizing data from it. Initially, the original table comprises two attributes and 500 rows. The data is partitioned into two groups of rows through clustering, which introduces a *sum node*. Subsequently, the columns within each group are separated by a *product node*, with each column forming a leaf node. Each leaf node stores a histogram representing its corresponding single-column subtable. To ensure DP, the table partition decision $(\mathcal{S}_L, \mathcal{S}_R)$ for the sum node, which consists of two sets of row indices or IDs, and the histogram H for each leaf node are perturbed. During the database synthesis phase, synthetic data are sampled from these perturbed histograms and progressively merged up the hierarchy. A product node performs a horizontal concatenation of columns, while a sum node executes a vertical concatenation of rows. This recursive procedure is repeated until it reaches the root node, resulting in the generation of a complete synthetic database.

Table 1: Comparison with existing differentially private data synthesis methods.

Method	Multi-relation	Similarity		Efficiency
		Data distr.	Query runtime	
AIM [29]		low	low	low
PrivSyn [47]		low	med.	med.
DataSynthesizer [32]		low	med.	high
PrivBayes [46]		low	med.	high
Exponential-PreFair [34]		med.	low	low
MST [28]		med.	low	med.
DPGAN [26]		med.	med.	low
PrivMRF [3]		med.	high	low
Greedy-PreFair [34]		med.	high	med.
PrivLava [4]	✓	high	med.	low
PrivBench	✓	**high**	**high**	**high**

In essence, our contribution is a novel exploration into the balance between synthesizing enterprise-level databases that maintain a high fidelity to original user data for DBMS benchmarking, while preserving user privacy. Through the strategic incorporation of SPNs and DP, PrivBench aligns closely with benchmarking scenarios, protecting user privacy and reducing the compromise on the quality or authenticity of the synthesized database. Table 1 provides a summary of comparisons with alternative methods.

Our experimental results on several publicly accessible datasets demonstrate that, compared to PrivLava [4], the current state-of-the-art (SOTA) in privacy-preserving data synthesis, PrivBench consistently achieves superior performance in both data distribution similarity and query runtime similarity, while significantly enhancing synthesis efficiency. Notably, PrivBench reduces KL divergence (KLD) by up to 62.4% and Q-error by up to 97.8% relative to PrivLava, which respectively measure the differences in data distribution and query runtime performance between the original and synthetic data. Our contributions are summarized as follows.

- We study the problem of privacy-preserving database synthesis, which transforms a database in a differentially private manner while aligning the performance of query workloads on the synthetic database with the original one.

H-ID	Rooms	…
1	2	…
2	5	…
3	3	…
…	…	…
Range	[1, 10]	…

(a) Primary private table.

ID	Sex	Age	…	H-ID
1	1	27	…	1
2	0	25	…	1
3	0	30	…	2
4	1	32	…	2
5	0	5	…	2
6	1	46	…	3
…	…	…	…	…
Range	[0, 1]	[1, 100]	…	

(b) Secondary private table.

Figure 2: A database consisting of two private tables.

- To solve the problem, we propose PrivBench, which synthesizes databases by leveraging differentially private SPNs. These SPNs not only address the four key concerns in benchmark publishing scenarios but also support multi-relation database synthesis by constructing differentially private fanout tables for primary-foreign key references, adeptly managing complex dependencies among the relations. To the best of our knowledge, our work is the first to facilitate differentially private SPNs and the first to apply them to database synthesis.
- We conduct a rigorous analysis of privacy and time complexity for PrivBench. Our analysis shows how PrivBench ensures that its data synthesis, through the complex multi-SPN construction process, achieves database-level DP [4], and that the synthesis time is polynomial in the number of records and the number of attributes in each table.
- Our extensive experimental results illustrate that PrivBench consistently achieves better fidelity in both data distribution and query runtime performance while being significantly more efficient than alternative methods.

2 PROBLEM DEFINITION

We consider a scenario where a benchmark publisher, such as an enterprise or organization, synthesizes a database benchmark using private data (e.g., user data) and releases it to other parties, referred to as benchmark consumers, for testing purposes. The benchmark publisher is a trusted party who is granted access to the original database by the data entities (e.g., users). In contrast, the benchmark consumers may be malicious and may try to infer private information from the released benchmark. Therefore, data anonymization techniques, such as differential privacy [9], are required to safeguard data privacy and ensure compliance with data protection regulations like the GDPR.

Consider a benchmark that consists of a database instance $D = \{T_1, T_2, \ldots, T_n\}$ with its associated schema, each $T_i \in D$ being a private relation table. For any $T_i, T_j \in D$, a record $r_i \in T_i$ refers to a record $r_j \in T_j$, denoted as $r_i \rightarrow r_j$, if the foreign key of r_i refers to the primary key of r_j and the values of the two keys match. A record r_i depends on r_j, denoted as $r_i \rightsquigarrow r_j$, if either $r_i \rightarrow r_j$ or r_i refers to another record $r_k \in T_k$ ($k \neq i, j$) that depends on r_j. We assume $r \rightsquigarrow r$ never occurs for any record r.

Without loss of generality, following previous work [4], we assume that T_1 is a *primary* private table (e.g., the table in Figure 2a) that contains sensitive information, and any other table $T_i, i \in [2, n]$, is a *secondary* private table (e.g., the table in Figure 2b) in which the records depend on the records in T_1. We study the case of synthesizing a database $\widehat{D}$ similar to D while preserving the privacy of all private tables. We employ DP [9], a celebrated approach for protecting database privacy.

Definition 1 (Table-Level DP [9]). *A randomized algorithm $\mathcal{M}$ that takes as input a table T satisfies table-level ϵ-DP ($\epsilon \geq 0$) if for any possible output o, and for any pair of neighboring tables T, T' that differ in the value of only one record, we have*

$$\mathbf{Pr}[\mathcal{M}(T) = o] \leq \exp(\epsilon) \cdot \mathbf{Pr}[\mathcal{M}(T') = o].$$

Definition 2 (Database-Level DP [4]). *A randomized algorithm $\mathcal{M}$ that takes as input a database D satisfies database-level ϵ-DP ($\epsilon \geq 0$) if for any possible output o and for any pair of neighboring databases D, D', we have*

$$\mathbf{Pr}[\mathcal{M}(D) = o] \leq \exp(\epsilon) \cdot \mathbf{Pr}[\mathcal{M}(D') = o].$$

D and D' neighbor if for the primary table T_1, they differ in the value of only one record $r_1 \in T_1$, and for the other tables $T_i, i \in [2, n]$, they differ in the values of all records $r_i \rightsquigarrow r_1$ that depend on r_1.

Table-level DP provides a strong guarantee that the output distributions remain indistinguishable before and after any change to a single record $r \in T$. This guarantee should hold even when accounting for dependencies among records induced by foreign keys in the relational schema. For example, in Figure 2, if we aim to protect a household in the primary table with multiple household members in the secondary table, we need to ensure that any holistic changes to the information of each household and its members do not significantly affect the output. Therefore, we adopt *database-level* DP [4], which further ensures that any changes in other records resulting from modifying a record $r_1 \in T_1$ also have a limited impact on the output. To ensure a bounded multiplicity for neighboring databases, we follow prior work [4, 7, 19, 38] to assume that each record in T_1 is referred to by at most τ_i records for each secondary table T_i. The privacy budget ϵ controls the level of indistinguishability. The smaller the privacy budget, the greater the level of privacy protection. With the definition of DP, we target the following privacy-preserving database synthesis problem.

Problem 1 (Privacy-Preserving Database Synthesis). *Given a database D with its schema, the task is to generate a database $\widehat{D}$ with the same schema as D while ensuring database-level ϵ-DP.*

Whereas many differentially private data synthesizers are available [3, 4, 26, 28, 29, 32, 34, 46, 47], their aims are mainly reducing the errors of aggregate queries or optimizing downstream ML tasks, rather than publishing a database for benchmarking. Considering that the benchmark is used to test the performance of a DBMS, we aim to synthesize a database $\widehat{D}$ with the following concerns.

1. Data distribution similarity. $\widehat{D}$ should be statistically similar to D. Given a table $T \in D$ and its counterpart $\widehat{T} \in \widehat{D}$, we measure their statistical similarity by the KL divergence (KLD) of T from $\widehat{T}$:

$$\mathsf{KLD}(T \parallel \widehat{T}) = \sum\nolimits_{x \in \mathcal{X}(T \cup \widehat{T})} \mathbf{Pr}[x|T] \log \left(\frac{\mathbf{Pr}[x|T]}{\mathbf{Pr}[x|\widehat{T}]} \right)$$

where $\mathbf{Pr}[x|T]$ denotes the probability of row value x in T, and $\mathcal{X}(T)$ denotes the set of row values in T. Then, the KLD of D from $\widehat{D}$ is defined as the average KLD over all the tables of D.

2. Query runtime similarity. For the query workloads to be executed on D, we expect they report the same runtime performance on $\widehat{D}$. Given a query, we can compare the execution times on D and $\widehat{D}$. Since the execution time may vary across DBMSs, we can also compare the cardinality, i.e., the number of records in the query result, which is often used in a query optimizer for estimating query performance [22]. In particular, Q-error [31] is a widely used measure for comparing cardinalities:

$$\mathsf{QE}(q, D, \widehat{D}) = \max\left(\frac{Card(q,D)}{Card(q,\widehat{D})}, \frac{Card(q,\widehat{D})}{Card(q,D)}\right)$$

where q denotes a query in the workload, and $Card(q, D)$ denotes the cardinality of executing q on D. Then, we compute the mean Q-error of the queries in the workload.

3. Synthesis efficiency. $\widehat{D}$ should be synthesized efficiently. In practice, databases often contain a large number of records and attributes. As these databases are frequently updated with daily activities, the synthetic benchmark also needs to be updated accordingly, resulting in significant time costs. Therefore, we should ensure that the synthesis algorithm completes in polynomial time w.r.t. both the number of records and the number of attributes.

3 PRIVBENCH

In this section, we propose PrivBench, an SPN-based differentially private database synthesis method.

3.1 Sum-Product Network

An SPN [33] is a rooted acyclic-directed graph that represents the data distribution of a dataset. PrivBench utilizes SPNs to address the three concerns outlined in Section 2 due to the following merits.

(1) SPNs have a strong ability to capture data distributions. Current SOTAs [3, 4] for privacy-preserving data synthesis have employed graphical models (GMs) other than SPNs to learn data distributions. However, it has been proven that all tractable GMs can be transformed into equivalent SPNs and that SPNs are even strictly more general, implying that SPNs can learn more accurate data distributions than other GMs [33].
(2) SPNs have demonstrated excellent performance in cardinality estimation [12, 13]. Our insight is that, since computing the cardinalities of queries on synthetic data can be approximately viewed as estimating the cardinalities on the original data, we believe that SPNs should perform well in optimizing the Q-error metric, resulting in better query runtime similarity.
(3) Inference in SPNs finishes in time linear to the number of nodes [33], making sampling data from SPNs notably efficient.

In a nutshell, the use of SPNs for database synthesis aligns with the major concerns of benchmark publishing except that SPNs do not provide privacy protection.

Following prior work on cardinality estimation [13], we employ binary tree-structured SPNs with sum and product nodes as internal nodes and leaves. Formally, given a table T with attributes $X_1, \ldots, X_M$, a sum (resp. product) node v_{sum} (resp. v_{prod}) splits the rows (resp. columns) into two subtables $T[\mathcal{S}_L], T[\mathcal{S}_R]$, where $\mathcal{S}_L, \mathcal{S}_R$ are two subsets of row (resp. column) indices. A leaf node v_{leaf} represents the marginal distribution of an attribute $X_m, m \in [M]$ w.r.t. a subset $\mathcal{S}$ of rows using a histogram, i.e., $v_{\mathsf{leaf}} = \mathbf{Pr}[X_m|\mathcal{S}]$. Then, the value of a sum node is the weighted sum of its children v_L, v_R, i.e., $v_{\mathsf{sum}} = w_L \cdot v_L + w_R \cdot v_R$, where $w_L = \frac{|\mathcal{S}_L|}{|\mathcal{S}_L|+|\mathcal{S}_R|}, w_R = \frac{|\mathcal{S}_R|}{|\mathcal{S}_L|+|\mathcal{S}_R|}$ are weights proportional to the sizes of the split clusters, while the value of a product node is the product of its children, i.e., $v_{\mathsf{prod}} = v_L \cdot v_R$. Consequently, the value of the root node approximates the joint distribution $\mathbf{Pr}[X_1, ..., X_M]$ for a given table by summing and multiplying the marginal distributions represented by the leaves from bottom to top.

Algorithm 1: PrivBench

Input : Database D
Output : Synthetic database $\widehat{D}$
Param: Privacy budgets ϵ_i^s (for SPN construction) and ϵ_i^f (for fanout construction), $\forall i \in [n]$

1 **for** *each private table* T_i *of* D **do**
2 $t_i \leftarrow \mathsf{PrivSPN}(T_i, \epsilon_i^s)$;
3 **for** *each pair of tables* (T_i, T_j) *where* T_i *refers to* T_j **do**
4 $t_i' \leftarrow \mathsf{PrivFanout}(T_i, t_i, FK_{i,j}, \epsilon_i^f)$;
5 **for** *each modified SPN* t_i' **do**
6 $\widehat{T}_i \leftarrow \mathsf{SampleDataFromSPN}(t_i')$;
7 **return** $\widehat{D} = \{\widehat{T}_1, \ldots, \widehat{T}_n\}$;

3.2 Overview of PrivBench

As shown in Algorithm 1, PrivBench involves a three-phase process for database synthesis.

- *Private SPN Construction (Lines 1–2).* Firstly, we construct an SPN t_i for each table T_i in the input database D. Each SPN t_i is created by a differentially private algorithm PrivSPN, where each leaf represents the marginal of an attribute for some rows.
- *Private Fanout Construction (Lines 3–4).* Secondly, we complement each SPN t_i with some leaf nodes that model primary-foreign key references to obtain a modified SPN t_i'. We use a differentially private algorithm PrivFanout to calculate *fanout frequencies* for each foreign key and store them in the complemented leaf nodes.
- *SPN-Based Database Synthesis (Lines 5–6)* Thirdly, we sample a table $\widehat{T}_i$ from each modified SPN t_i' to synthesize a database $\widehat{D}$.

3.3 Private SPN Construction

3.3.1 Overview of PrivSPN. Algorithm 2 outlines the process for constructing an SPN on a single table, which calls Algorithm 5 to optimize the SPN's structure. The PrivSPN algorithm recursively generates a binary tree $t = (\mathsf{parent}, \mathsf{left}, \mathsf{right})$, where parent is a node, and left, right are the two child subtrees. We divide PrivSPN into the following three procedures.

- Planning *(Operation Planning):* We decide an operation op for the parent node and the privacy budget ϵ_{op} allocated to op.
- ParentGen *(Parent Generation):* According to the operation op and the privacy budget ϵ_{op}, we generate the parent node.
- ChildrenGen *(Children Generation):* We further generate the children left, right through recursive calls to PrivSPN.

For ease of presentation, we first introduce how the parent and children are generated, and then we delve into the Planning procedure.

Algorithm 2: Private SPN Construction PrivSPN(T, ϵ)

Input : table T, total privacy budget ϵ.
Output : A tree of SPN t = (parent, left, right)

op, $\epsilon_{\text{op}}, \bar{\epsilon} \leftarrow$ Planning(T, ϵ);
parent, $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R) \leftarrow$ ParentGen(T, op, ϵ_{op});
left, right $\leftarrow$ ChildrenGen(T, op, $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R, \bar{\epsilon}$);
return (parent, left, right);
procedure ParentGen(T, op, ϵ_{op})
  if op = OP.LEAF **then**
    $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R \leftarrow \emptyset, \emptyset$;
    $\widetilde{his} \leftarrow \text{his}(T) + \text{Lap}(\frac{\Delta(\text{his})}{\epsilon_{\text{op}}})$; parent $\leftarrow$ LeafNode($\widetilde{his}$);
  else if op = OP.SUM **then**
    $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R \leftarrow$ RowSplit(T, ϵ_{op}); parent $\leftarrow$ SumNode($\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R$);
  else if op = OP.PRODUCT **then**
    $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R \leftarrow$ ColSplit(T, ϵ_{op}); parent $\leftarrow$ ProdNode($\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R$);
  return parent, $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$;
procedure ChildrenGen(T, op, $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R, \bar{\epsilon}$)
  if op = OP.SUM **then**
    $\epsilon_L \leftarrow \bar{\epsilon}, \epsilon_R \leftarrow \bar{\epsilon}$
  else if op = OP.PRODUCT **then**
    $\epsilon_L \leftarrow \bar{\epsilon} \cdot \frac{\sigma(T[\widetilde{\mathcal{S}}_L])}{\sigma(T[\widetilde{\mathcal{S}}_L]) + \sigma(T[\widetilde{\mathcal{S}}_R])}, \epsilon_R \leftarrow \bar{\epsilon} - \epsilon_L$
  if op = OP.LEAF **then**
    left $\leftarrow$ null; right $\leftarrow$ null;
  else
    left $\leftarrow$ PrivSPN($T[\widetilde{\mathcal{S}}_L], \epsilon_L$); right $\leftarrow$ PrivSPN($T[\widetilde{\mathcal{S}}_R], \epsilon_R$);
  return left, right

Algorithm 3: Row Splitting RowSplit

Input : table T, privacy budget ϵ
Output : row partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$
Param: number of iterations J, minimum table size β

1 $(\mathcal{S}_{L,0}, \mathcal{S}_{R,0}) \leftarrow$ Sample a row partition such that $|\mathcal{S}_{L,0}| = |T|/2$;
2 **for** j *from* 1 *to* J **do**
3 $c_L \leftarrow \sum_{r \in T[\mathcal{S}_{L,j-1}]} r / |\mathcal{S}_{L,j-1}|$ // Center of left cluster;
4 $c_R \leftarrow \sum_{r \in T[\mathcal{S}_{R,j-1}]} r / |\mathcal{S}_{R,j-1}|$ // Center of right cluster;
5 $\text{diff}_i \leftarrow \text{dist}(T[i] - c_L) - \text{dist}(T[i] - c_R), \forall i \in [|T|]$;
6 $\widetilde{\text{diff}}_i \leftarrow \text{diff}_i + \text{Lap}(\frac{\Delta(\text{diff}_i) \cdot J}{\epsilon}), \forall i \in [|T|]$;
7 $\mathcal{S}_{L,j} \leftarrow \arg_i(\widetilde{\text{diff}}_i \le 0)$; $\mathcal{S}_{R,j} \leftarrow \arg_i(\widetilde{\text{diff}}_i > 0)$;
8 Adjust clusters $\mathcal{S}_{L,J}, \mathcal{S}_{R,J}$ to ensure their sizes are not less than β;
9 **return** $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R) = (\mathcal{S}_{L,J}, \mathcal{S}_{R,J})$;

3.3.2 Parent generation. We generate a parent node based on the operation op and perturb it with the privacy budget ϵ_{op}, as follows.

Case 1: op = OP.LEAF *(Line 6).* We generate a leaf node LeafNode($\widetilde{his}$) with a perturbed histogram $\widetilde{his}$. Specifically, the ParentGen procedure first computes a histogram $\text{his}(T)$ over the table T and then perturbs the histogram by adding some Laplace noise $\text{Lap}(\frac{\Delta(\text{his})}{\epsilon_{\text{op}}})$, where Δ returns the global sensitivity of a given function f, i.e., $\Delta(f) = \max_{T,T'} |f(T) - f(T')|$.

Algorithm 4: Column Splitting ColSplit

Input : table T, privacy budget ϵ
Output : column partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$

1 **for** j *from* 1 *to* $|attr(T)|$ **do**
2 $o_i = (\mathcal{S}_{L,j}, \mathcal{S}_{R,j}) \leftarrow$ Sample a column partition such that $|\mathcal{S}_{L,j}| = |attr(T)|/2$;
3 $\rho_j \leftarrow \text{NMI}(T, (\mathcal{S}_{L,j}, \mathcal{S}_{R,j}))$
4 $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R) \leftarrow$ Sample a partition from the set of partitions $O = \{o_j\}_{\forall j}$ with probability defined in Equation (1) ;
5 **return** $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$;

Case 2: op = OP.SUM *(Line 9).* We call the RowSplit mechanism (see Algorithm 3) to compute a differentially private row partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$, where $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R$ are two subsets of row indices. Then, the row partition yields the sum node SumNode($\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R$).

For row splitting, our strategy is to group similar rows into the same cluster. By generating distinct segments, SPNs can more effectively learn and represent the local distributions of different data subsets. To achieve this, we adopt the DP-based K-Means algorithm, DPLloyd [37], which groups data points into K clusters by comparing their distances to K cluster centers. However, the output of DPLloyd is the cluster centers rather than the data points within the clusters, which does not work in our task. Therefore, we adapt DPLloyd to the RowSplit mechanism to support yielding a row partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$ as output.

Concretely, as shown in Algorithm 3, we first uniformly sample a row partition $(\mathcal{S}_{L,0}, \mathcal{S}_{R,0})$ where the clusters are half-sized (Line 1). Then, we update the partition through J iterations. In each iteration j, we first calculate the centers c_L, c_R of the clusters $\mathcal{S}_{L,j-1}, \mathcal{S}_{R,j-1}$ (Lines 3-4). Next, for each row $T[i]$, we compute the difference diff_i between its distances $\text{dist}(T[i], c_L), \text{dist}(T[i], c_R)$ to the two centers (Line 5) and add some Laplace noise $\text{Lap}(\frac{\Delta(\text{diff}_i) \cdot J}{\epsilon})$ to ensure DP (Line 6), where $\Delta(\text{diff}_i)$ equals the absolute difference between the supremum and infimum of the distance function dist_i. Then, if the perturbed difference $\widetilde{\text{diff}}_i$ is positive, the row $T[i]$ is considered closer to the center c_R and thus should be assigned to the right cluster $\mathcal{S}_{R,j}$; otherwise, it is assigned to $\mathcal{S}_{L,j}$ (Line 7). After J iterations, we adjust the sizes of the clusters $\mathcal{S}_{L,J}, \mathcal{S}_{R,J}$ through random sampling to ensure each of them has at least β rows (Line 8); this adjustment enhances the utility of the perturbed histogram in each leaf node, as a larger histogram is more resilient to DP noise. Note that the distance function diff_i is customizable. In our experiments, we use the L1 distance for each numerical attribute and the Hamming distance for each categorical attribute; the distance function is defined as the sum of the attribute-wise distances.

Case 3: op = OP.PRODUCT *(Line 11).* We generate a product node ProdNode($\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R$) by calling the ColSplit mechanism (see Algorithm 4). ColSplit determines a differentially private column partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$, where $\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R$ are subsets of column indices.

For column splitting, our goal is to divide the given table into two subtables with low correlation so that the product of the marginal distributions of the subtables approximates the original joint distribution. Therefore, the ColSplit mechanism attempts to minimize the normalized mutual information (NMI, see Definition 3) of

the split subtables. As shown in Algorithm 4, we first construct a candidate set O of column partitions (Lines 1-3) and then sample a partition from the candidates as the output (Lines 4-5). Note that we conduct $|attr(T)|$ iterations to select $|attr(T)|$ column partitions, rather than all possible column partitions, as the candidate set because it makes the computation tractable. In each iteration j, we uniformly sample a partition $o_j = (\mathcal{S}_{L,j}, \mathcal{S}_{R,j})$ from all half-sized column partitions such that $|\mathcal{S}_{L,j}| = |attr(T)|/2$ since we observed that subtables with similar dimensionalities generally exhibit lower NMI. Subsequently, we calculate NMI ρ_j for candidate o_j (Line 3). After the iterations, we sample a partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$ from the candidate set O with the following probability distribution (Line 4):

$$\Pr[(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R) = o_j] = \frac{\exp(\frac{-\epsilon \cdot \rho_j}{2\Delta(\mathrm{NMI}|O)})}{\sum_{j' \in \left[|attr(T)|\right]} \exp(\frac{-\epsilon \cdot \rho_{j'}}{2\Delta(\mathrm{NMI}|O)})}, \quad (1)$$

where $\Delta(\mathrm{NMI}|O) = \max_{T,T',o \in O} |\mathrm{NMI}(T,o) - \mathrm{NMI}(T',o)|$ is the sensitivity of the NMI w.r.t candidates O. Intuitively, the smaller the correlation between subtables $T[\widetilde{\mathcal{S}}_L], T[\widetilde{\mathcal{S}}_R]$, the lower the information loss from column splitting. Therefore, the lower the NMI ρ_j, the higher the probability of sampling the partition o_j in Equation (1). Moreover, as the privacy budget ϵ decreases, the sampling probabilities become closer to a uniform distribution, increasing the likelihood of outputting a partition with high correlation.

Definition 3 (Normalized Mutual Information). *The entropy H of a table T is $H(T) = \sum_{x \in \mathcal{X}(T)} \Pr[x|T] \log_2 \frac{1}{\Pr[x|T]}$, where $\mathcal{X}(T)$ is the set of row values in T. For a row or column partition $(\mathcal{S}_L, \mathcal{S}_R)$ of table T, the NMI is:*

$$\mathrm{NMI}(T, (\mathcal{S}_L, \mathcal{S}_R)) = \frac{H(T[\mathcal{S}_L]) + H(T[\mathcal{S}_R]) - H(T)}{\sup(H(T))},$$

where $\sup(H(T)) = \log_2 |T|$ is the upper bound of $H(T)$.

3.3.3 Children generation. If op = OP.LEAF, the children are set to null because the parent node is a leaf node (Lines 19–20). Otherwise, we generate the left and right subtrees for the split subtables $T[\widetilde{\mathcal{S}}_L], T[\widetilde{\mathcal{S}}_R]$. Specifically, we first allocate the remaining privacy budget $\bar{\epsilon}$ among the subtrees (Lines 15–18), which will be discussed in Section 4.1. Then, given the allocated privacy budgets ϵ_L, ϵ_R, we call the PrivSPN procedure for the subtables $T[\widetilde{\mathcal{S}}_L], T[\widetilde{\mathcal{S}}_R]$ to generate the left and right subtrees, respectively (Line 22).

3.3.4 Planning. The Planning procedure takes the role of determining an operation op for the parent node, which can be creating a leaf/sum/product node. Since all these types of operations should be performed in a differentially private manner, Planning also allocates a privacy budget ϵ_{op} for the parent node and calculates the remaining privacy budget $\bar{\epsilon}$ for the children. Concretely, Planning proceeds as follows.

1. Leaf node validation (Lines 1–2). We validate if we should generate a leaf node as the parent node. When the given table has only one attribute, i.e., $|attr(T)| = 1$, the operation must be creating a leaf node, i.e., $\leftarrow$ OP.LEAF, and all the given privacy budget ϵ should be allocated for op, i.e., $\epsilon_{\mathrm{op}} = \epsilon$.

2. Correlation trial (procedure CorrTrial*).* We preliminarily test the performance of column splitting, which will guide the selection of the operation op through the subsequent DecideOP procedure.

Algorithm 5: Operation Planning Planning(T, ϵ)

Input : table T, privacy budget ϵ
Output : next operation op, privacy budget ϵ_{op} for op, remaining privacy budget $\bar{\epsilon}$
Param: threshold for column splitting α, minimum table size β, budget ratios γ_1, γ_2

1 **if** $|attr(T)| = 1$ **then**
2 op $\leftarrow$ OP.LEAF; $\epsilon_{\mathrm{op}} \leftarrow \epsilon$; $\bar{\epsilon} \leftarrow 0$;
3 **else**
4 $\tilde{\rho}, \epsilon_{\mathrm{eval}} \leftarrow \mathrm{CorrTrial}(T, \epsilon)$;
5 op $\leftarrow \mathrm{DecideOP}(T, \tilde{\rho})$;
6 $\epsilon_{\mathrm{op}}, \bar{\epsilon} \leftarrow \mathrm{AllocBudgetOP}(T, \mathrm{op}, \epsilon, \epsilon_{\mathrm{eval}})$;
7 **return** op, $\epsilon_{\mathrm{op}}, \bar{\epsilon}$;
8 **procedure** CorrTrial(T, ϵ)
9 **if** $|T| \geq 2\beta$ *and* $|attr(T)| > 1$ **then**
10 $\epsilon_{\mathrm{eval}} \leftarrow \epsilon \cdot \gamma_1 / \sigma(T)$;
11 $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R) \leftarrow \mathrm{ColSplit}(T, \epsilon_{\mathrm{eval}} \cdot \gamma_2)$;
12 $\tilde{\rho} \leftarrow \mathrm{NMI}(T, (\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)) + \mathrm{Lap}(\frac{\Delta(\mathrm{NMI})}{\epsilon_{\mathrm{eval}} \cdot (1-\gamma_2)})$;
13 **else**
14 $\epsilon_{\mathrm{eval}} \leftarrow 0$; $\tilde{\rho} \leftarrow 0$;
15 **return** $\tilde{\rho}, \epsilon_{\mathrm{eval}}$
16 **procedure** DecideOP$(T, \tilde{\rho})$
17 **if** $|T| \geq 2\beta$ *and* $|attr(T)| > 1$ **then**
18 **if** $\tilde{\rho} \leq \alpha$ **then**
19 op $\leftarrow$ OP.PRODUCT
20 **else**
21 op $\leftarrow$ OP.SUM
22 **else**
23 op $\leftarrow$ OP.PRODUCT
24 **return** op
25 **procedure** AllocBudgetOP$(T, \mathrm{op}, \epsilon, \epsilon_{\mathrm{eval}})$
26 **if** op = OP.PRODUCT *and* $|attr(T)| = 2$ **then**
27 $\epsilon_{\mathrm{op}} \leftarrow 0$;
28 **else**
29 $\epsilon_{\mathrm{op}} \leftarrow \epsilon / \sigma(T) - \epsilon_{\mathrm{eval}}$;
30 **return** $\epsilon_{\mathrm{op}}, \epsilon - \epsilon_{\mathrm{eval}} - \epsilon_{\mathrm{op}}$

Specifically, we consume a privacy budget $\epsilon_{\mathrm{eval}} \cdot \gamma_2$ to generate a trial column partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$ and an additional privacy budget of $\epsilon_{\mathrm{eval}} \cdot (1 - \gamma_2)$ to compute its noisy NMI $\tilde{\rho}$ (Lines 11-12):

$$\tilde{\rho} = \mathrm{NMI}(T, (\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)) + \mathrm{Lap}(\frac{\Delta(\mathrm{NMI})}{\epsilon_{\mathrm{eval}} \cdot (1 - \gamma_2)}).$$

Intuitively, the noisy NMI metric $\tilde{\rho}$ serves as a predictor of whether the column splitting in the ParentGen procedure will result in a high-fidelity data partition, which is useful for planning the next operation in DecideOP. Note that when either column or row splitting is infeasible, we do not need to evaluate the correlation since the operation must be the feasible one (Line 14).

3. Operation decision (procedure DecideOP*).* We determine the operation op for the parent node. When the size of the given table is too small to allow for row splitting and the table contains more than one attribute, the operation must be column splitting (Lines

Algorithm 6: Private Fanout Construction PrivFanout

Input : Table T_i, SPN t_i, foreign key $FK_{i,j}$,privacy budget ϵ
Output : Modified SPN t_i'

1 $\mathcal{A} \leftarrow$ Find the attribute with the most leaf nodes in t_i;
2 $t_i' \leftarrow$ Make a copy of t_i;
3 **for** *each leaf node* $\text{leaf}_{\mathcal{A}}$ *of* t_i *of* $\mathcal{A}$ **do**
4 **for** *each value* fk *of* $FK_{i,j}$ **do**
5 $\mathcal{F}_{i,j}[fk] \leftarrow$ count the rows in $\text{leaf}_{\mathcal{A}}$ for which the foreign key $FK_{i,j}$ equals fk;
6 $\widetilde{\mathcal{F}}_{i,j} \leftarrow \mathcal{F}_{i,j} + \text{Lap}(\frac{\Delta(\mathcal{F}_{i,j})}{\epsilon})$;
7 $\text{leaf}_{FK_{i,j}} \leftarrow \text{LeafNode}(\widetilde{\mathcal{F}}_{i,j})$;
8 $\text{parent} \leftarrow \text{ProdNode}(\mathcal{A}, FK_{i,j})$;
9 Replace $\text{leaf}_{\mathcal{A}}$ in t_i' with subtree $(\text{parent}, \text{leaf}_{\mathcal{A}}, \text{leaf}_{FK_{i,j}})$

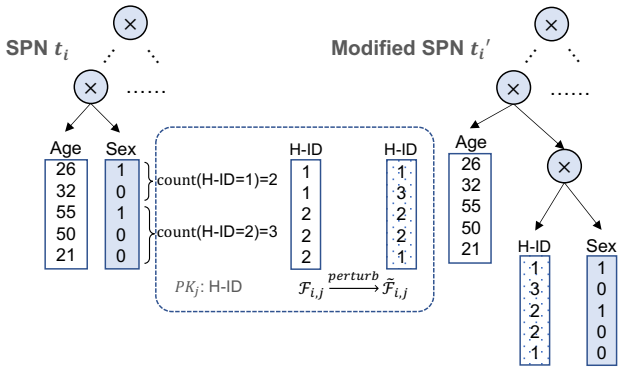


Figure 3: Fanout table construction.

22–23). However, when both column splitting and row splitting are feasible, we choose one of them according to whether the noisy NMI $\tilde{\rho}$ exceeds a predefined threshold α (Lines 17–21). Intuitively, when the correlation (measured by $\tilde{\rho}$) between the vertically split subtables is too high, the information loss regarding data distribution caused by the column splitting is excessive; consequently, row splitting should be prioritized in this case.

4. Budget allocation (procedure AllcBudgetOP*).* We allocate privacy budgets $\epsilon_{\text{op}}, \bar{\epsilon}$ to the parent node and its children, respectively. In the case where the operation is to vertically split a table with only two attributes, no privacy budget should be consumed for the parent node because there is only one possible column partition (Lines 26–27). In other cases, we uniformly allocate privacy budgets among all nodes of the SPN tree. That is, we divide the privacy budget ϵ by a scale metric $\sigma(T)$ to determine the privacy budget $\epsilon_{\text{eval}} + \epsilon_{\text{op}}$ for correlation trial and parent node generation, where $\sigma(T)$ represents the maximum possible number of nodes in the SPN tree of table T:

$$\sigma(T) = 2|T| \cdot |attr(T)|/\beta - 1.$$

Then, we allocate a privacy budget $\epsilon_{\text{eval}} = \epsilon/\sigma(T) \cdot \gamma_1$ (Line 10) for correlation trial and $\epsilon_{\text{op}} = \epsilon/\sigma(T) - \epsilon_{\text{eval}}$ (Line 29) for parent node generation such that $\epsilon_{\text{eval}} + \epsilon_{\text{op}} = \epsilon/\sigma(T)$, where $\gamma_1 \in [0, 1]$.

3.4 Private Fanout Construction

After constructing SPNs for all private tables in the input database, we follow prior work [44, 45] to utilize *fanout distribution*, i.e., the distribution of the foreign key in the referencing table, to model their primary-foreign key references. However, to maintain fanout distributions for all joined keys, prior work [44, 45] uses a full outer join of all the tables, which causes large space overhead.

To address this issue, we propose Algorithm 6, which models primary-foreign key references by augmenting SPNs with leaf nodes that store fanout distributions. Specifically, to construct the reference relation between tables T_i and T_j where T_i refers to T_j, we first find a certain attribute $\mathcal{A}$ and traverse all its leaf nodes. For each leaf node $\text{leaf}_{\mathcal{A}}$, we identify the rows in T_i whose column of attribute $\mathcal{A}$ is stored in $\text{leaf}_{\mathcal{A}}$ and count the fanout frequencies of the foreign key $FK_{i,j}$ w.r.t. the identified rows in a *fanout table* $\mathcal{F}_{i,j}$ (Lines 3–5). Then, the fanout table is perturbed by adding some Laplace noise and stored in a new leaf node $\text{leaf}_{FK_{i,j}}$ (Lines 6–7). Finally, we replace $\text{leaf}_{\mathcal{A}}$ with a product node that connects $\text{leaf}_{\mathcal{A}}$ and $\text{leaf}_{FK_{i,j}}$ (Lines 8–9). Additionally, to minimize the impact of Laplace noise on the fanout distribution, we require attribute $\mathcal{A}$ to have the largest leaf nodes. Intuitively, the overall fanout distribution of a foreign key is the average of the distributions stored in its leaf nodes. Thus, the variance caused by the Laplace noise decreases as the number of leaf nodes increases.

Example 2. Figure 3 shows an example of how PrivBench augments the original SPN t_i with the leaf nodes of the foreign key $\text{H} - \text{ID}$. First, we identify that the attribute Sex possesses the most leaf nodes in t_i. For each leaf node of attribute Sex, we locate the corresponding foreign keys using the row indices. Next, we compute the fanout table, perturb the fanouts, and store the results in a new leaf node. Finally, the new leaf node is added to the SPN as a sibling to the leaf node of attribute Sex and as the child of a newly added product node, resulting in a modified SPN t_i'.

3.5 SPN-Based Database Synthesis

Algorithm 7 shows how we synthesize a table $\widehat{T}_i$ given an SPN t_i. If the root node of t_i is a leaf node, we synthesize a table $\widehat{T}_i$ by histogram sampling. Specifically, each column $\widehat{T}_i$ is sampled from the marginal distribution represented by the perturbed histogram or fanout table stored in the leaf (Lines 1–2), and the size of $\widehat{T}_i$ matches the total count of the histogram or fanout table. If it is a sum (product, resp.) node, we recursively apply the SampleDataFromSPN procedure to the left and right children of t_i and vertically (horizontally, resp.) concatenate the returned tables $\widehat{T}_L$ and $\widehat{T}_R$ (Lines 4–9). Finally, we obtain a table $\widehat{T}_i$ that has a data distribution similar to that of the private table T_i. By synthesizing tables for all modified SPNs, we obtain a synthetic database $\widehat{D}$. Since the input SPN t_i is constructed in a differentially private manner, sampling data from it does not consume any privacy budget according to the post-processing property of DP [8].

4 THEORETICAL ANALYSIS

In this section, we provide a rigorous analysis of the privacy guarantee and time complexity of PrivBench. All the missing proofs can be found in our technical report [11].

Algorithm 7: SampleDataFromSPN

Input : SPN t_i
Output : Synthetic table $\widehat{T}_i$

1 **if** t_i.root *is a leaf node* **then**
2 $\quad \widehat{T}_i \leftarrow$ Sample data from marginal distribution represented by histogram $t_i.\widetilde{\text{his}}$ or fanout table $t_i.\widetilde{\mathcal{F}}_{i,j}$
3 **else**
4 $\quad \widehat{T}_L \leftarrow$ SampleDataFromSPN(t_i.left);
5 $\quad \widehat{T}_R \leftarrow$ SampleDataFromSPN(t_i.right);
6 $\quad$ **if** t_i.root *is a sum node* **then**
7 $\quad\quad \widehat{T}_i \leftarrow$ Vertically concatenate $\widehat{T}_L$ and $\widehat{T}_R$;
8 $\quad$ **if** t_i.root *is a product node* **then**
9 $\quad\quad \widehat{T}_i \leftarrow$ Horizontally concatenate $\widehat{T}_L$ and $\widehat{T}_R$;
10 **return** $\widehat{T}_i$

4.1 Privacy Analysis on PrivSPN

In this subsection, we first prove that the Planning, ParentGen, and ChildrenGen procedures satisfy DP, and then prove that PrivSPN therefore achieves DP.

4.1.1 Analysis of Planning. The Planning procedure consists of the CorrTrial, DecideOP, and AllocBudgetOP subprocedures. If the subprocedures satisfy DP, we can conclude that Planning ensures DP according to the sequential composition theorem of DP [8].

CorrTrial. (1) When $|T| \geq 2\beta$ and $|attr(T)| > 1$, the CorrTrial procedure employs the $(\epsilon_{\text{eval}} \cdot \gamma_2)$-DP mechanism $\text{ColSplit}(T, \epsilon_{\text{eval}} \cdot \gamma_2)$ to compute a perturbed partition $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$. Then, it calculates the NMI of $(\widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R)$ and injects the Laplace noise $\text{Lap}(\frac{\Delta(\text{NMI})}{\epsilon_{\text{eval}} \cdot (1-\gamma_2)})$ into it, which ensures $\big(\epsilon_{\text{eval}} \cdot (1-\gamma_2)\big)$-DP. Consequently, according to the sequential composition theorem of DP [8], $\text{CorrTrial}(T, \epsilon)$ satisfies ϵ_{eval}-DP. (2) When $|T| < 2\beta$ or $|attr(T)| = 1$, since varying the value of any row in T does not have any impact on the output $(\tilde{\rho}, \epsilon_{\text{eval}})$, $\text{CorrTrial}(T, \epsilon)$ satisfies 0-DP according to Definition 1. Note that we adopt the bounded DP interpretation; in the case of unbounded DP, adding/removing a row from T causes the procedure to take the other conditional branch (i.e., Lines 9–12), thereby breaking the DP guarantee.

DecideOP *and* AllocBudgetOP. Similar to the second case of CorrTrial, for both procedures DecideOP and AllocBudgetOP, because changing the value of any row in T does not affect the table size $|T|$ and the number of attributes $|attr(T)|$, it also does not impact their outputs. Therefore, both procedures ensure 0-DP.

Planning. When $|attr(T)| = 1$, Planning satisfies 0-DP since any change to T does not impact the output (Lines 1–2); otherwise, as it sequentially combines CorrTrial, DecideOP, and AllocBudgetOP, $\text{Planning}(T, \epsilon)$ satisfies ϵ_{eval}-DP according to the sequential composition theorem of DP [8], where the value of ϵ_{eval} differs in different cases (see Lines 10 and 14).

Lemma 1. *If $|attr(T)| > 1$, $\text{Planning}(T, \epsilon)$ satisfies table-level $\big(\epsilon \cdot \gamma_1 / \sigma(T)\big)$-DP; otherwise, it satisfies table-level 0-DP.*

4.1.2 Analysis of ParentGen. Then, we show that in any case of the given operation op, ParentGen achieves DP. (1) When op = OP.LEAF, we ensure DP using the Laplace mechanism. That is, we inject Laplacian noise $\text{Lap}(\frac{\Delta(\text{his})}{\epsilon_{\text{op}}})$ into the histogram $\text{his}(T)$, which satisfies ϵ_{op}-DP. (2) When op = OP.SUM, we call $\text{RowSplit}(T, \epsilon_{\text{op}})$ to generate a perturbed row partition $(\widetilde{S}_L, \widetilde{S}_R)$, which achieves ϵ_{op}-DP. (3) When op = OP.PRODUCT, our column splitting mechanism $\text{ColSplit}(T, \epsilon_{\text{op}})$ is essentially an instance of the exponential mechanism [9], thereby guaranteeing ϵ_{op}-DP.

Lemma 2. $\text{ParentGen}(T, \text{op}, \epsilon_{\text{op}})$ *satisfies tabel-level ϵ_{op}-DP.*

4.1.3 Analysis of ChildrenGen. Next, we analyze the DP guarantee of $\text{ChildrenGen}(T, \text{op}, \widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R, \bar{\epsilon})$ in different cases. (1) When op = OP.LEAF, since ChildrenGen always returns two null children, it must satisfy 0-DP. Therefore, in this case, $\text{PrivSPN}(\mathsf{T}, \epsilon)$ achieves ϵ-DP. (2) When op = OP.SUM, Theorem 3 demonstrates a parallel composition property of DP in the case of row splitting: If constructing the left and right subtrees satisfies ϵ_L-DP and ϵ_R-DP, respectively, then the entire process of constructing both subtrees satisfies $\max\{\epsilon_L, \epsilon_R\}$-DP. Therefore, to optimize the utility of the subtrees, we maximize each subtree's privacy budget by setting $\epsilon_L = \epsilon_R = \bar{\epsilon}$. Note that Theorem 3 differs from the celebrated parallel composition theorem [8]: their theorem assumes unbounded DP while our theorem considers bounded DP. (3) When op = OP.PRODUCT, we allocate privacy budgets based on the scales of the subtrees (Line 18). Intuitively, a subtree with a larger scale should be assigned with a larger privacy budget to balance their utility. Thus, their privacy budgets ϵ_L, ϵ_R are proportional to their scales $\sigma(T[\widetilde{S}_L]), \sigma(T[\widetilde{S}_R])$. Note that publishing the scales does not consume any privacy budget since varying any row of T does not change $\sigma(T[\widetilde{S}_L])$ and $\sigma(T[\widetilde{S}_R])$.

Theorem 3 (Parallel composition under bounded DP). *Given a row partition $(\mathcal{S}_1, \ldots, \mathcal{S}_K)$, publishing $\mathcal{M}_1(T[\mathcal{S}_1]), \ldots, \mathcal{M}_K(T[\mathcal{S}_K])$ satisfies table-level $\max\{\epsilon_1, \ldots, \epsilon_K\}$-DP, where $\mathcal{S}_k$ is a subset of row indices and $\mathcal{M}_k$ is a table-level ϵ_k-DP algorithm, $\forall k \in [K]$.*

Consequently, when op = OP.SUM or op = OP.PRODUCT, we can prove the DP guarantee by mathematical induction: when $\text{PrivSPN}(T[\widetilde{S}_L], \epsilon_L)$ (resp. $\text{PrivSPN}(T[\widetilde{S}_R], \epsilon_R)$) returns a subtree with only a leaf node, it satisfies ϵ_L-DP (resp. ϵ_R-DP); otherwise, assuming $\text{PrivSPN}(T[\widetilde{S}_L], \epsilon_L)$ (resp. $\text{PrivSPN}(T[\widetilde{S}_R], \epsilon_R)$) satisfies ϵ_L-DP (resp. ϵ_R-DP), according to the sequential composition theorem [8] or Theorem 3, we conclude that $\text{ChildrenGen}(T, \text{op}, \widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R, \bar{\epsilon})$ satisfies $\bar{\epsilon}$-DP, where $\bar{\epsilon} = \epsilon_L + \epsilon_R$ or $\bar{\epsilon} = \max\{\epsilon_L, \epsilon_R\}$.

Lemma 3. *When* op = OP.LEAF, $\text{ChildrenGen}(T, \text{op}, \widetilde{\mathcal{S}}_L, \widetilde{\mathcal{S}}_R, \bar{\epsilon})$ *satisfies table-level 0-DP; otherwise, it satisfies table-level $\bar{\epsilon}$-DP.*

4.1.4 Analysis of PrivSPN. Based on Lemmas 1, 2, and 3, we conclude the DP guarantee of PrivSPN through sequential composition.

Lemma 4. $\text{PrivSPN}(T, \epsilon)$ *satisfies table-level ϵ-DP.*

In addition, since PrivSPN allocates the total privacy budget ϵ from the root to the leaf nodes in a top-down manner, it may result in insufficient budgets allocated to the leaves. Consequently, a practical question arises: If we would like to guarantee a privacy budget of ϵ_{leaf} for each leaf node, how large should the total privacy budget ϵ be? Theorem 4 answers this question: A privacy budget $\epsilon = \big(\epsilon_{\text{leaf}} \cdot \sigma(T)\big)$ is sufficient. This also explains why the privacy

budgets ϵ_L and ϵ_R should be proportional to the corresponding scales $\sigma(T[\widetilde{S_L}])$ and $\sigma(T[\widetilde{S_R}])$ in Line 18 of Algorithm 2.

Theorem 4. *Consider a procedure* PrivSPN(T, ϵ) *that publishes an SPN with K leaf nodes* PrivSPN$(S_1, \epsilon_{\text{leaf}}), \ldots,$ PrivSPN$(S_K, \epsilon_{\text{leaf}})$, *where S_k is a single-column subtable of table T, $\forall k \in [K]$.* PrivSPN(T, ϵ) *satisfies table-level $\big(\epsilon_{\text{leaf}} \cdot \sigma(T)\big)$-DP.*

4.2 Privacy Analysis on PrivFanout

For each leaf node leaf_{FK}, PrivFanout injects the Laplace noise into the fanout table $\mathcal{F}_{i,j}$ (Line 6), which implements an ϵ-differentially private Laplace mechanism. Then, according to Theorem 3, publishing all the leaf nodes leaf_{FK} with perturbed fanout tables also satisfies ϵ-DP due to the property of parallel composition. Therefore, PrivFanout ensures DP.

Lemma 5. PrivFanout$(T_i, t_i, FK_{i,j}, \epsilon)$ *satisfies table-level ϵ-DP.*

4.3 Privacy Analysis on PrivBench

Given that both PrivSPN and PrivFanout satisfy DP, the DP guarantee of the PrivBench algorithm is established according to Theorem 5. Note that to measure the indistinguishability level of database-level DP, we cannot simply accumulate the privacy budgets ϵ_i assigned to each private table $T_i \in D$ for table-level DP. Intuitively, while table-level DP only guarantees the indistinguishability of neighboring tables, database-level DP requires indistinguishability of two tables that differ in at most τ_i tuples because those tuples may depend on the same tuple in the primary private table. Consequently, as shown in Corollary 1, the indistinguishability levels of procedures PrivSPN and PrivFanout at the table level should be reduced by factor τ_i for database-level DP. Therefore, to ensure database-level ϵ-DP for PrivBench, we can allocate the total privacy budget ϵ as follows:

$$\epsilon_i^s = \frac{\epsilon \cdot \gamma}{\tau_1 + \cdots + \tau_n}, \quad \epsilon_i^f = \frac{\epsilon \cdot (1-\gamma)}{\tau_1 \cdot |FK(T_1)| + \cdots + \tau_n \cdot |FK(T_n)|},$$

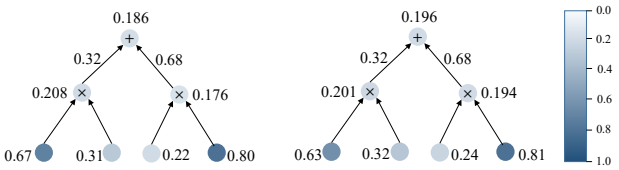
where $\gamma \in [0, 1]$ is the ratio of privacy budget allocated for SPN construction, and $|FK(T_i)|$ is the number of foreign keys in table T_i. In our experiments, we follow the above setting to allocate privacy budgets $\epsilon_i^s, \epsilon_f^s$ in PrivBench.

Theorem 5. *Given that $\mathcal{M}_i$ satisfies table-level ϵ_i-DP for all $i \in [n]$, the composite mechanism $\mathcal{M}(D) = \big(\mathcal{M}_1(T_1), ..., \mathcal{M}_n(T_n)\big)$ satisfies database-level $(\sum_{i\in[n]} \tau_i \cdot \epsilon_i)$-DP.*

Corollary 1 (of Theorem 5). PrivBench *satisfies database-level $(\sum_{i\in[n]} \tau_i \cdot \epsilon_i^s + \sum_{T_i \text{ refers to } T_j} \tau_i \cdot \epsilon_i^f)$-DP.*

4.4 Time Complexity Analysis

We show that PrivBench(D) completes in polynomial time. (1) To construct a private SPN t_i for a table T_i, PrivSPN needs to recursively call itself to generate a full binary tree, where each node requires one call of PrivSPN. Because each leaf node stores a single-column subtable of T_i with at least β rows and one attribute, t_i can have at most $(|attr(T_i)| \cdot |T_i|/\beta)$ leaf nodes. Consequently, t_i can have at most $(2|attr(T_i)| \cdot |T|/\beta - 1)$ nodes, which can be computed by PrivSPN(T_i, ϵ_i^s) with $O(|attr(T_i)| \cdot |T_i|)$ recursive calls. Then, we can easily observe that the non-recursive work including running the Planning and ParentGen procedures can finish in polynomial time. Therefore, PrivSPN(T_i, ϵ_i^s) completes in polynomial time. (2) To construct a fanout table for each pair of referential tables T_i, T_j, PrivFanout needs to enumerate at most $(|attr(T_i)| \cdot |T_i|/\beta)$ leaf nodes of t_i to replace each leaf with a subtree, which finishes in $O(|attr(T_i)| \cdot |T_i|)$ time. (3) For each modified SPN t_i', SampleDataFromSPN(t_i') requires $O(|attr(T_i)| \cdot |T_i|)$ recursive calls and some polynomial-time non-recursive work; thus, it finishes in $O(|attr(T_i)| \cdot |T_i|)$ time. Given that PrivSPN, PrivFanout, and SampleDataFromSPN finish in polynomial time, PrivBench completes in polynomial time.



(a) Selectivity over Original Data (b) Selectivity over Generated Data

Figure 4: An example of Query Heat Map for a given query. The edge values indicate the proportions of rows assigned to each subtree of a sum node, while the node values represent the selectivities for the given query.

Lemma 6. *Given a database $D = \{T_1, \ldots, T_n\}$,* PrivBench(D) *finishes in $O\big(\sum_{i\in[n]} \text{poly}(|attr(T_i)|, |T_i|)\big)$ time.*

5 DISCUSSION

In this section, we discuss several practical issues regarding PrivBench.

Privacy-preserving/secure data valuation. In recent years, research on data trading has been gaining increasing attention in the database community [6, 20, 23, 25, 48–50]. A key research question is how to address the Arrow Information Paradox for privacy-preserving/secure data valuation. Specifically, since the utility of data is task-specific, data buyers often need to test the data to determine its value before reaching a deal. However, once a buyer gains access to the data, the seller faces the risk of the data being replicated without payment. To tackle this issue for trading machine learning models, Zheng et al. [49] utilized homomorphic encryption (HE) to enable buyers to evaluate the value of data in an encrypted environment. PrivBench can also address the paradox by allowing buyers to assess the value of data based on synthetic databases while ensuring the privacy and security of the original data. Compared to HE-based methods, PrivBench-empowered data valuation, though sacrificing some accuracy due to the injection of DP noise, offers better computational efficiency and facilitates trustworthy, real-time data trading.

Parameter selection. Here, we discuss how to configure the parameters of PrivBench. The privacy budget ϵ is a general parameter in DP, and various methods have been proposed for selecting it, such as economic methods [14] or voting mechanisms [18]. The other parameters, including α, β, γ, γ_1, and γ_2, however, are unique to PrivBench and can be determined empirically. In practice, since DP assumes that the benchmark publisher is trusted, they can access

Table 2: Summary of datasets, workloads, and baselines.

Dataset	#(Private) Tables	(#Rows, #Cols.) of Private Table	Workload	Baseline
Adult	1 (1)	(45222, 15)	SAM-1000	PrivSyn, PrivBayes, AIM, DataSynthesizer, MST, Exponential-PreFair, Greedy-PreFair, DPGAN (GPU-accelerated), PrivMRF (GPU-accelerated)
California	2 (2)	Primary: (616115, 10) Secondary: (1690642, 16)	California-400	PrivLava (GPU-accelerated)
JOB-light	6 (2)	Primary: (2283757, 2) Secondary: (35824707, 3)	JOB-229, MSCN-226	PrivLava (GPU-accelerated)

both the original and synthetic data to evaluate the latter's quality in terms of data distribution similarity and query runtime similarity. Therefore, every time the publisher releases a benchmark, they can incrementally optimize the parameters based on experimental results to improve the performance of future benchmarks. Additionally, in our experiments, we apply the same parameters to all datasets. The results demonstrate that PrivBench consistently outperforms other methods, which suggests the transferability of the parameter settings among different datasets. In other words, even when adopting the same parameters for different benchmarking tasks, publishers still can achieve high-quality synthetic databases.

Visualization. Visualization enables us to intuitively evaluate whether the data generated by PrivBench closely approximates the original data as a benchmark. As shown in Figure 4, after synthesizing data using PrivBench, we can obtain a pair of SPNs built on the original and synthetic data, respectively. After testing a bundle of queries, we can determine the selectivity value for each leaf node. Subsequently, we can color the leaf nodes based on the magnitude of the selectivity value, with darker colors indicating higher values, creating a colored SPN. Ultimately, by comparing the colored SPN of the original data with that of the synthetic data, we can assess whether the two sets of data perform similarly in benchmarking.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Settings

Research questions. In this section, we conduct extensive experiments to answer the following research questions.

- **Data distribution similarity**: Is the database synthesized by PrivBench closer to the original database in data distribution?
- **Query runtime similarity**: For executing query workloads, is the database synthesized by PrivBench closer to the original database in terms of query runtime performance?
- **Synthesis time**: Can PrivBench synthesize databases efficiently?

Datasets, query workloads, and baselines. We use the following datasets, query workloads, and baselines to verify the performance of PrivBench, which are summarized in Table 2.

The *Adult* dataset [17] contains a single private table of census information about individuals. We execute *SAM-1000* [44], a workload with 1000 randomly generated queries, to evaluate the query runtime performance of PrivBench and baselines. The baselines cover mainstream differentially private single-relation data synthesis methods, including DataSynthesizer [32], PrivBayes [46], PrivSyn [47], MST [28], PrivMRF [3], AIM [29], Exponential-PreFair [34], Greedy-PreFair [34], and DPGAN [26]. Note that PrivMRF is the single-relation version of PrivLava [4].

The *California* dataset [4] consists of two private tables about household information. We randomly generate 400 queries following the method from prior work [16] to create the query workload for this dataset, named *California-400*. We use PrivLava [4] as the sole baseline, as it is the only SOTA method that supports multi-relation database synthesis.

The *JOB-light* dataset [44] includes six tables related to movies, which are extracted from the *Internet Movie Database (IMDB [1])*. We designate two of these tables as private, while the remaining four are public. Consequently, we generate synthetic tables only for the two private tables and combine these synthetic tables with the public tables into a single database for testing. We adopt the *MSCN* and *JOB* query workloads from prior work [44] and extract subsets of 226 and 229 queries, respectively, that involve operations over the two private tables. These subsets, named *MSCN-226* and *JOB-229*, serve as the query workloads for the Job-light dataset. PrivLava [4] serves as the baseline for multi-relation synthesis.

Metrics. For the first research question, we evaluate λ-way KLD to measure data distribution similarity between original and synthetic data. Specifically, following prior work [3], we enumerate all possible λ-way marginals for each private relation, where $\lambda \in \{2, 3, 4\}$. For each λ-way marginal, we compute the KLD between the original and synthetic tables. The λ-way KLD is then obtained as the average KLD across all λ-way marginals for all private relations. To prevent infinite KLD values, a small constant of 10^{-10} is added to each probability distribution. For the second research question regarding query runtime similarity, in addition to the Q-error metric, we also use the *query runtime discrepancy*, which measures the average percentage difference in query runtime. For the third research question, we report the time consumed for database synthesis.

Parameters. We set the default parameters of PrivBench as follows: $\alpha = 0.5, \beta = 10000, \gamma = 0.9, \gamma_1 = 0.5, \gamma_2 = 0.5$. Then, the total privacy budget ϵ is set to 3.2 by default, following the setting in the SOTA work, PrivLava [4]. The setting of the number of iterations J for RowSplit follows prior analysis [37]. Additionally, the baselines PrivSyn, MST, PrivMRF, AIM, Exponential-PreFair, Greedy-PreFair, DPGAN, and PrivLava only satisfy (ϵ, δ)-DP, where the δ parameter allows the privacy guarantee to be violated with a small probability. We set $\delta = 10^{-12}$ for all these baselines.

Environment. All experiments are implemented in Python and performed on a Linux server with an Intel(R) Core(R) Silver i9-13900K 3.0GHz CPU, an NVIDIA GeForce RTX 4090 GPU, and 64GB RAM. The DBMS we use to test query execution is PostgreSQL 15.2 with default settings for all parameters. We accelerate matrix computation using the GPU for the computationally inefficient baselines DPGAN, PrivMRF, and PrivLava.

Table 3: Performance of data synthesis methods on the Adult dataset with the SAM-1000 query workload.

Model	KLD			Q-error				Query runtime discrepancy (%)				Synthesis time (s)	
	2-way	3-way	4-way	Mean	Median	75th	MAX	Mean	Median	75th	MAX	Learning	Inference
PrivSyn	13.01	14.68	15.03	2.56	1.30	1.60	22.79	8.76	5.86	13.83	47.12	72.61	10.59
PrivBayes	10.37	12.64	13.74	1.41	1.31	1.55	3.45	9.67	3.71	13.95	95.82	12.58	5.32
AIM	7.75	10.13	11.63	**1.34**	**1.24**	**1.50**	**2.98**	40.60	39.48	41.97	77.04	1865	144.17
DataSynthesizer	6.92	9.84	11.94	1.40	1.26	1.58	3.55	8.36	3.08	11.09	62.42	**0.12**	0.51
MST	5.24	7.07	8.53	1.38	**1.24**	1.52	5.71	14.73	14.47	16.57	35.34	178.80	0.11
Exponential-PreFair	5.12	6.98	8.48	1.43	1.30	1.61	7.2	18.50	17.93	20.23	37.78	1736	0.14
Greedy-PreFair	4.73	6.52	8.02	1.44	1.29	1.60	17.16	4.69	3.73	6.21	26.54	180.33	**0.10**
DPGAN	4.56	6.68	8.55	1.37	1.27	1.52	3.53	8.18	6.28	10.77	57.68	746.53	0.49
PrivMRF (PrivLava)	4.55	6.29	7.79	1.38	1.25	1.52	4.19	3.11	2.61	4.29	21.6	1703	41.36
PrivBench	**1.71**	**2.79**	**4.05**	1.40	1.25	1.56	7.78	**1.84**	**1.48**	**2.38**	**12.96**	2.35	0.13

Table 4: Performance of database synthesis methods on the California dataset with the California-400 query workload.

Model	2-way KLD						3-way KLD						3-way KLD					
	Privacy budget ϵ						Privacy budget ϵ						Privacy budget ϵ					
	0.1	0.2	0.4	0.8	1.6	3.2	0.1	0.2	0.4	0.8	1.6	3.2	0.1	0.2	0.4	0.8	1.6	3.2
PrivLava	2.30	1.28	1.03	0.93	0.88	0.88	3.93	2.21	1.77	1.56	1.48	1.46	5.50	3.20	2.59	2.27	2.14	2.09
PrivBench	0.87	0.77	0.69	0.63	0.60	0.58	1.85	1.74	1.62	1.54	1.51	1.50	2.94	2.84	2.67	2.58	2.56	2.55

Model	Q-error						Query runtime discrepancy (%)						Synthesis time (s)	
	Privacy budget ϵ						Privacy budget ϵ						Learning	Inference
	0.1	0.2	0.4	0.8	1.6	3.2	0.1	0.2	0.4	0.8	1.6	3.2		
PrivLava	791.15	66.46	8.31	2.26	1.45	1.33	5.20	4.10	3.83	3.05	2.88	3.18	1520	26013
PrivBench	26.07	5.57	7.17	5.11	5.18	5.18	5.70	5.34	4.59	4.18	4.26	4.29	38.31	4.92

Table 5: Performance of database synthesis methods on the JOB-light dataset with the JOB-229 and MSCN-226 query workloads.

Model	2-way KLD	Q-error		Query runtime discrepancy (%)		Synthesis time	
		JOB-229	MSCN-226	JOB-229	MSCN-226	Learning	Inference
PrivLava	5.40	152.05	34.52	32.77	35.47	8101	18104
PrivBench	3.16	3.33	11.32	20.10	13.24	40.72	30.22

Model	Q-error										
	JOB-229							MSCN-226			
	Cardinality			No. joins				Cardinality			No. joins	
	Low	Med.	High	1	2	3	4	Low	Med.	High	1	2
PrivLava	421.59	13.80	22.69	8.73	15.79	74.49	1827.51	84.43	9.04	10.43	23.21	39.59
PrivBench	4.81	1.67	3.55	3.31	2.96	3.68	4.51	29.39	2.09	2.62	22.53	6.30

6.2 Evaluation on Distribution Similarity

Single-relation synthesis. Table 3 shows the results on the Adult dataset. We can see that for different values of λ, PrivBench performs significantly better than all baselines on the λ-way KLD metric. This indicates that PrivBench can synthesize single-table data with a higher fidelity of data distribution.

Multi-relation synthesis. Tables 4 and 5 compare multi-relation database synthesis methods on the California and JOB-light datasets. We can observe that in various cases, the performance of PrivBench in terms of KLD is comparable to or even better than that of the SOTA method PrivLava. Moreover, as shown in Table 4, as the privacy budget decreases, the deterioration trend of the KLD metric for PrivBench is more moderate compared to PrivLava. This indicates that PrivBench has a greater advantage in scenarios where the original data is highly sensitive or data sharing is strictly restricted, resulting in a tiny privacy budget.

6.3 Evaluation on Query Runtime Similarity

Single-relation synthesis. For the Adult dataset, as shown in Table 3, on Q-error-related metrics, both PrivBench and the SOTA methods perform exceptionally well. However, in terms of query runtime discrepancy, PrivBench significantly outperforms them. This indicates that the single-table benchmarks generated by PrivBench can better preserve similarity in query runtime performance.

Multi-relation synthesis. Table 4 presents the Q-error and runtime discrepancy for PrivBench and PrivLava under different privacy budgets for the California dataset. Similar to the case with the KLD metric, PrivBench underperforms compared to PrivLava on the Q-error metric when the privacy budget is relatively large. However, as the privacy budget decreases, PrivLava's Q-error performance deteriorates rapidly, while PrivBench remains relatively stable and significantly surpasses PrivLava when the privacy budget is very limited. This demonstrates PrivBench's robustness in data-sensitive

scenarios. Additionally, PrivBench and PrivLava both perform excellently in query runtime discrepancy, with no significant difference between them.

For the JOB-light dataset, we present the performance of the two synthesis methods under different types of queries in Table 5. Specifically, the JOB-229 and MSCN-226 query workloads are divided into three groups based on query cardinality levels: low, medium, and high. The results indicate that PrivBench achieves consistently lower Q-error across all cardinality levels. When the cardinality level is low, PrivLava's Q-error deteriorates significantly, whereas PrivBench performs markedly better than PrivLava. Moreover, the queries are also categorized based on the number of joins in Table 5. As the number of joins increases, the Q-error of PrivLava can become extremely large, while that of PrivBench remains low. This suggests that PrivBench may perform better on enterprise-level databases with complex internal dependencies among tables.

6.4 Evaluation on Synthesis Time

Single-relation synthesis. We report the time required to synthesize the Adult dataset in Table 3. The synthesis time can be divided into the learning time needed to build the synthesis model and the inference time required to sample data from the model. In terms of the overall synthesis time, PrivBench outperforms all the baselines. Among them, PrivMRF and DPGAN require extensive matrix computations and even utilize the GPU to accelerate these operations, yet they still require a substantial amount of time to train the data synthesis models. In contrast, PrivBench ensures high fidelity of synthetic data in terms of data distribution and query runtime performance, while requiring only minimal time for model learning and inference. This suggests that PrivBench can effectively adapt to frequent benchmark updates and even support real-time, high-fidelity benchmark releases.

Multi-relation synthesis. As shown in Tables 4 and 5, PrivBench remains highly efficient in synthesizing multi-table databases, with a significantly lower time cost than PrivLava. Note that in our experiment, PrivLava also employs the GPU to greatly accelerate its model learning and inference, while PrivBench relies solely on CPU computation. Although PrivBench requires traversing leaf nodes to construct fanout tables, the workload of fanout construction is light, and the number of leaf nodes is linear to both the number of records and attributes. Consequently, modeling primary-foreign key dependencies among tables is also efficient in PrivBench.

7 RELATED WORK

DP-based data synthesis. With the growing awareness of user privacy and the introduction of data protection regulations worldwide, DP-based data synthesis has gained increasing attention in recent years [3–5, 10, 15, 24, 26–30, 32, 34, 36, 39, 41–43, 46, 47]. The vast majority of existing work focuses on enhancing the fidelity of the synthetic data in data distribution while overlooking the fidelity in query runtime performance, making them less promising for benchmark publishing scenarios. Among them, deep learning-based methods [5, 10, 15, 26, 27, 39, 43] train neural networks to fit the joint distribution of the original data. Query workload-based methods [28, 29, 41] optimize some given queries' accuracy, rather than their runtime performance, to learn a workload-optimal data distribution. Graphical methods [3, 4, 30, 32, 34, 46] achieve high-fidelity data distribution by learning the marginal distribution of the data through graphical models, but these efforts still neglect query runtime performance. To the best of our knowledge, PrivBench is the first differentially private data synthesis framework that simultaneously optimizes fidelity in both data distribution and query runtime. Additionally, apart from the recent SOTA, PrivLava [4], PrivBench is the only framework that can synthesize multi-relation databases while ensuring database-level DP.

SPN-based data management. SPNs [33], which excel in representing complex dependencies and data distributions, have been employed for various data analysis tasks, such as image processing and natural language processing [35]. However, the application of SPNs in data management tasks remains underdeveloped. Hilprecht et al. [13] have employed SPNs for approximate query processing and cardinality estimation. Recently, Kroes et al. [21] proposed the first SPN-based data synthesis method. However, unlike PrivBench, their approach only supports single-table databases and does not provide privacy guarantees. To the best of our knowledge, PrivBench provides the first differentially private method for SPN construction. Moreover, Treiber [40] proposed an SPN inference framework based on secure multiparty computation, called CryptoSPN, which ensures that parties with distributed data can compute query results without sharing data. However, since CryptoSPN does not protect the privacy of query results, attackers may still infer the original data from the query results, which compromises privacy.

8 CONCLUSION

In this paper, we delve into the domain of synthesizing databases that preserve privacy for benchmark publishing. Our focus is on creating a database that upholds DP while ensuring that the performance of query workloads on the synthesized data closely aligns with the original data. We propose PrivBench, an innovative synthesis framework designed to generate high-fidelity data while ensuring robust privacy protection. PrivBench utilizes SPNs at its core to segment and sample data from SPN leaf nodes and conducts subsequent operations on these nodes to ensure privacy preservation. It allows users to adjust the granularity of SPN partitions and determine privacy budgets through parameters, crucial for customizing levels of privacy preservation. The data synthesis algorithm is proven to uphold DP. Experimental results highlight PrivBench's capability to create data that not only maintains privacy but also demonstrates high query performance fidelity, showcasing improvements in query runtime discrepancy, query cardinality error, and KLD over alternative approaches. The promising research directions for future work are twofold: (i) identifying the optimal privacy budget allocation scheme and (ii) generating data and query workloads simultaneously.

ACKNOWLEDGMENTS

This work was supported by National Key R&D program of China 2021YFB3301500, SZU Research Instrument Development and Cultivation 2023YQ017, Guangdong Province Key Laboratory of PHPC 2017B030314073, JSPS KAKENHI 21K19767, 23K17456, 23K24851, 23K25157, 23K28096, and CREST JPMJCR22M2.

REFERENCES

[1] Internet Movie Database. https://www.imdb.com/.
[2] TPC benchmarks. https://www.tpc.org/.
[3] K. Cai, X. Lei, J. Wei, and X. Xiao. Data synthesis via differentially private Markov random field. *PVLDB*, 14(11):2190–2202, 2021.
[4] K. Cai, X. Xiao, and G. Cormode. Privlava: Synthesizing relational data with foreign keys under differential privacy. *SIGMOD*, 1(2):142:1–142:25, 2023.
[5] D. Chen, T. Orekondy, and M. Fritz. GS-WGAN: A gradient-sanitized approach for learning differentially private generators. *NeurIPS*, 33:12673–12684, 2020.
[6] L. Chen, P. Koutris, and A. Kumar. Towards model-based pricing for machine learning in a data marketplace. In *SIGMOD*, pages 1535–1552, 2019.
[7] W. Dong, J. Fang, K. Yi, Y. Tao, and A. Machanavajjhala. R2T: Instance-optimal truncation for differentially private query evaluation with foreign keys. In *SIMOD*, pages 759–772, 2022.
[8] C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.
[9] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *FnT TCS*, 9(3-4):211–407, 2014.
[10] C. Ge, S. Mohapatra, X. He, and I. F. Ilyas. Kamino: Constraint-aware differentially private data synthesis. *PVLDB*, 14(10):1886–1899, 2021.
[11] Y. Ge, J. Qin, S. Zheng, Y. Zhong, B. Tang, Y.-X. Qiu, R. Mao, Y. Yuan, M. Onizuka, and C. Xiao. Privacy-enhanced database synthesis for benchmark publishing. *arXiv preprint arXiv:2405.01312*, 2024.
[12] Y. Han, Z. Wu, P. Wu, R. Zhu, J. Yang, L. W. Tan, K. Zeng, G. Cong, Y. Qin, A. Pfadler, Z. Qian, J. Zhou, J. Li, and B. Cui. Cardinality estimation in DBMS: A comprehensive benchmark evaluation. *PVLDB*, 15(4):752–765, 2021.
[13] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. DeepDB: Learn from data, not from queries! *PVLDB*, 13(7):992–1005, 2020.
[14] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth. Differential privacy: An economic method for choosing epsilon. In *CSF*, pages 398–410, 2014.
[15] J. Jordon, J. Yoon, and M. van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *ICLR*, 2019.
[16] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*, 2019.
[17] R. Kohavi et al. Scaling up the accuracy of Naive-Bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207, 1996.
[18] N. Kohli and P. Laskowski. Epsilon voting: Mechanism design for parameter selection in differential privacy. In *PAC*, pages 19–30, 2018.
[19] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. PrivateSQL: A differentially private SQL query engine. *PVLDB*, 12(11):1371–1384, 2019.
[20] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. *JACM*, 62(5):1–44, 2015.
[21] S. K. S. Kroes, M. van Leeuwen, R. H. H. Groenwold, and M. P. Janssen. Generating synthetic mixed discrete-continuous health records with mixed sum-product networks. *JAMIA*, 30(1):16–25, 2022.
[22] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.
[23] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. *TODS*, 39(4):1–28, 2014.
[24] H. Li, L. Xiong, L. Zhang, and X. Jiang. DPSynthesizer: Differentially private data synthesizer for privacy preserving data sharing. *PVLDB*, 7(13):1677–1680, 2014.
[25] J. Liu, J. Lou, J. Liu, L. Xiong, J. Pei, and J. Sun. Dealer: An end-to-end model marketplace with differential privacy. *PVLDB*, 14(6):957–969, 2021.
[26] T. Liu, J. Fan, G. Li, N. Tang, and X. Du. Tabular data synthesis with generative adversarial networks: design space and optimizations. *VLDB J.*, 33(2):255–280, 2024.
[27] Y. Long, B. Wang, Z. Yang, B. Kailkhura, A. Zhang, C. A. Gunter, and B. Li. G-PATE: Scalable differentially private data generator via private aggregation of teacher discriminators. In *NeurIPS*, pages 2965–2977, 2021.
[28] R. McKenna, G. Miklau, and D. Sheldon. Winning the NIST contest: A scalable and general approach to differentially private synthetic data. *JPC*, 11(3), 2021.
[29] R. McKenna, B. Mullins, D. Sheldon, and G. Miklau. AIM: An adaptive and iterative mechanism for differentially private synthetic data. *PVLDB*, 15(11):2599–2612, 2022.
[30] R. McKenna, D. Sheldon, and G. Miklau. Graphical-model based estimation and inference for differential privacy. In *ICML*, volume 97, pages 4435–4444, 2019.
[31] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB*, 2(1):982–993, 2009.
[32] H. Ping, J. Stoyanovich, and B. Howe. Datasynthesizer: Privacy-preserving synthetic datasets. In *SSDBM*, pages 42:1–42:5, 2017.
[33] H. Poon and P. M. Domingos. Sum-product networks: A new deep architecture. In *ICCVW*, pages 689–690, 2011.
[34] D. Pujol, A. Gilad, and A. Machanavajjhala. PreFair: Privately generating justifiably fair synthetic data. *PVLDB*, 16(6):1573–1586, 2023.
[35] R. Sánchez-Cauce, I. París, and F. J. Díez. Sum-product networks: A survey. *TPAMI*, 44(7):3821–3839, 2022.
[36] J. Snoke and A. B. Slavkovic. pmse mechanism: Differentially private synthetic data with maximal distributional similarity. In *PSD*, pages 138–159, 2018.
[37] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin. Differentially private K-Means clustering. In *CODASPY*, pages 26–37, 2016.
[38] Y. Tao, X. He, A. Machanavajjhala, and S. Roy. Computing local sensitivities of counting queries with joins. In *SIGMOD*, pages 479–494, 2020.
[39] R. Torkzadehmahani, P. Kairouz, and B. Paten. DP-CGAN: Differentially private synthetic data and label generation. In *CVPRW*, pages 98–104, 2019.
[40] A. Treiber, A. Molina, C. Weinert, T. Schneider, and K. Kersting. CryptoSPN: Privacy-preserving sum-product network inference. In *ECAI*, pages 1946–1953, 2020.
[41] G. Vietri, G. Tian, M. Bun, T. Steinke, and S. Wu. New oracle-efficient algorithms for private synthetic data release. In *ICML*, pages 9765–9774, 2020.
[42] T. Wang, N. Li, and Z. Zhang. DPSyn: Experiences in the NIST differential privacy data synthesis challenges. *JPC*, 11(2), 2021.
[43] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou. Differentially private generative adversarial network. *CoRR*, abs/1802.06739, 2018.
[44] J. Yang, P. Wu, G. Cong, T. Zhang, and X. He. SAM: Database generation from query workloads with supervised autoregressive models. In *SIGMOD*, pages 1542–1555, 2022.
[45] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica. NeuroCard: One cardinality estimator for all tables. *PVLDB*, 14(1):61–73, 2020.
[46] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via bayesian networks. *TODS*, 42(4):25:1–25:41, 2017.
[47] Z. Zhang, T. Wang, N. Li, J. Honorio, M. Backes, S. He, J. Chen, and Y. Zhang. PrivSyn: Differentially private data synthesis. In *USENIX Security*, pages 929–946, 2021.
[48] S. Zheng, Y. Cao, and M. Yoshikawa. Money cannot buy everything: Trading mobile data with controllable privacy loss. In *MDM*, pages 29–38, 2020.
[49] S. Zheng, Y. Cao, and M. Yoshikawa. Secure Shapley value for cross-silo federated learning. *PVLDB*, 16(7):1657–1670, 2023.
[50] S. Zheng, Y. Cao, M. Yoshikawa, H. Li, and Q. Yan. FL-Market: Trading private models in federated learning. In *IEEE BigData*, pages 1525–1534, 2022.