DP-S4S: Accurate and Scalable Select-Join-Aggregate Query Processing with User-Level Differential Privacy

YUAN QIU, Southeast University, China
XIAOKUI XIAO, National University of Singapore, Singapore
YIN YANG, Hamad Bin Khalifa University, Qatar

Answering Select-Join-Aggregate (SJA) queries with differential privacy (DP) is a fundamental problem with important applications in various domains. The current state-of-the-art methods ensure user-level DP (i.e., the adversary cannot infer the presence or absence of any given individual user with high confidence) and achieve instance-optimal accuracy on the query results. However, these solutions involve solving expensive optimization programs, which may incur prohibitive computational overhead for large databases.

One promising direction to achieve scalability is through sampling, which provides a tunable trade-off between result utility and computational costs. However, applying sampling to differentially private SJA processing is a challenge for two reasons. First, it is unclear *what to sample*, in order to achieve the best accuracy within a given computational budget. Second, prior solutions were not designed with sampling in mind, and their mathematical tool chains are not sampling-friendly. To our knowledge, the only known solution that applies sampling to private SJA processing is S&E, a recent proposal that (i) samples users and (ii) combines sampling directly with existing solutions to enforce DP. We show that both are suboptimal designs; consequently, even with a relatively high sample rate, the error incurred by S&E can be 10x higher than the underlying DP mechanism without sampling.

Motivated by this, we propose *Differentially Private Sampling for Scale* (DP-S4S), a novel mechanism that addresses the above challenges by (i) sampling *aggregation units* instead of users, and (ii) laying the mathematical foundation for SJA processing under *Rényi-DP*, which composes more easily with sampling. Further, DP-S4S can answer both scalar (e.g., COUNT, SUM, etc.) and vector (with GROUP BY predicts) SJA queries. Extensive experiments on real data demonstrate that DP-S4S enables scalable SJA processing on large datasets under user-level DP, while maintaining high result utility.

CCS Concepts: • **Security and privacy** → **Differential privacy**; • **Information systems** → *Database query processing*; • **Mathematics of computing** → Probability and statistics.

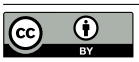
Additional Key Words and Phrases: differential privacy, user-level privacy, sampling, query processing, graph analytics, relational databases

ACM Reference Format:
Yuan Qiu, Xiaokui Xiao, and Yin Yang. 2026. DP-S4S: Accurate and Scalable Select-Join-Aggregate Query Processing with User-Level Differential Privacy. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 165 (June 2026), 24 pages. https://doi.org/10.1145/3802042

1 Introduction

Differential privacy (DP) [13] provides a rigorous framework for releasing query results while protecting individuals' privacy, by requiring outputs to remain statistically similar on any two *neighboring datasets*, so that the adversary cannot infer with high confidence which dataset was

Authors' Contact Information: Yuan Qiu, yuanqiu@seu.edu.cn, Southeast University, Nanjing, Jiangsu, China; Xiaokui Xiao, xkxiao@nus.edu.sg, National University of Singapore, Singapore City, Singapore; Yin Yang, yyang@hbku.edu.qa, Hamad Bin Khalifa University, Doha, Qatar.



ACM 2836-6573/2026/6-ART165
https://doi.org/10.1145/3802042

used to produce the outputs. The precise privacy guarantee depends on the definition of neighboring datasets. In a relational database, *record-level DP* considers two datasets as neighbors if they differ in exactly one record [11, 16, 19, 31, 34]. *User-level DP*, in contrast, treats two datasets as neighbors if they differ in the set of *all the records* contributed by one user [15, 22]. The latter, stronger notion is necessary when each user may contribute multiple, potentially correlated records, e.g., connections in a social network, purchase histories in an e-commerce platform, etc.

This paper focuses on enforcing user-level DP when answering *Select-Join-Aggregate (SJA)* queries, which combine selection predicates, joins, and aggregation operators to capture a broad spectrum of reporting workloads, making them a central target for private query processing systems [10, 23]. A major challenge in processing SJA queries under user-level DP is that such a query often has a high *sensitivity*, meaning that adding or removing a user (which may lead to the addition or removal of many records) can significantly alter the query output. As we review in Section 2.1, a high sensitivity necessitates large amounts of noise to be injected into the query result to satisfy DP, leading to poor result utility.

To address this issue, state-of-the-art approaches to SJA processing under user-level DP (e.g., [9, 10]) commonly employ a technique called *truncation*. The main idea is to impose a limit on the influence that any single user can have on the SJA query result. For instance, consider a simple query: edge counting in a social network, in which a user can have a large number of connections (e.g., up to 5,000 on Facebook [32]). With truncation, the query engine can impose an upper bound, e.g., up to 10 edges are counted for each user, which reduces the query sensitivity dramatically, leading to a lower noise requirement to satisfy DP, as we explain in Section 2.1.

Truncation, however, also has a negative impact on query result accuracy, since it introduces a bias by capping the influence of every user. Thus, to design an effective truncation scheme that maximizes result utility, one must carefully balance the error due to truncation bias and the noise injected to satisfy DP, which is a non-trivial problem. Prior work R2T [9] addresses this by considering all possible truncation thresholds that are a power of two, which guarantees asymptotically optimal result accuracy, as we elaborate further in Section 2.3. The downside of R2T, however, is that for each truncation threshold, it needs to solve a linear program to choose aggregation units to retain [22]. Since it is computationally expensive to solve so many linear programs, R2T does not scale well to large datasets [15], as shown in our experiments in Section 5. Further, R2T is limited to SJA queries with scalar outputs, i.e., each query returns a single numeric result. For SJA queries with *vector outputs*, e.g., those involving a GROUP BY clause, the best known solution to our knowledge is PMSJA [10], which involves solving even more expensive quadratically constrained quadratic programs (QCQPs) [27, 29], which exacerbates the scalability issue.

A natural approach to improving efficiency is sampling, which enables the user to reach a desirable trade-off between query result accuracy and computational costs by choosing an appropriate sample size. Unfortunately, incorporating sampling into user-level private SJA processing is challenging, since (i) it is unclear what objects should be sampled to stay within a computational budget, and (ii) existing truncation-based pipelines are calibrated in the (ε, δ)-DP model, which is not easy to compose with sampling. To our knowledge, the only user-level DP solution with sampling is S&E [15], which (i) samples users and then extends the sample by exploring their neighboring records, and (ii) applies an existing solution such as R2T or PMSJA on the extended sample set. Since the sampled users may remain highly correlated, the privacy analysis demands injecting far more noise, yielding errors that can exceed those of R2T or PMSJA by more than an order of magnitude.

This paper presents *Differentially Private Sampling for Scale* (DP-S4S), a mechanism that delivers user-level DP accuracy comparable to R2T and PMSJA while scaling to much larger databases. DP-S4S samples *aggregation units* (e.g., join tuples) instead of users, which keeps samples compact and

avoids the worst cross-user correlations as in S&E. We show that the error introduced by sampling can be partially compensated for by its privacy amplification effect, which reduces the noise level to satisfy DP. Meanwhile, for vector SJA queries, we build new mathematical infrastructure to adapt the idea of PMSJA to satisfy *Rényi DP* (which composes well with sampling), before converting the privacy guarantee to the standard DP notion. In particular, we design the first smooth sensitivity mechanism under Rényi DP, which can be of independent interest.

Extensive experiments with real data demonstrate that for scalar queries, DP-S4S sustains the accuracy of R2T while dramatically improving efficiency. For vector queries, the extended version of DP-S4S matches PMSJA at a fraction of the runtime. Both versions of DP-S4S consistently outperform S&E, achieving more than 10× lower error at smaller sample rates under the same privacy budget.

2 Background

Section 2.1 provides preliminaries on the definitions and properties of differential privacy. Section 2.2 clarifies the problem setting, and Section 2.3 reviews notable existing solutions.

2.1 Differential Privacy

In this paper, we follow the standard centralized differential privacy setting, in which a trusted data curator has access to the raw data, and releases only query results that under differential privacy constraints. The adversary aims to infer private information from the DP-compliant query outputs. Since DP offers information-theoretic privacy guarantees, we do not rely on any assumption on the computational power of the adversary. Let $\mathcal{I}$ be the domain of all instances of a given relational database schema, *differential privacy* (*DP*) is defined as follows:

Definition 2.1 (Differential Privacy [13]). For $\varepsilon > 0$ and $\delta \geq 0$, a randomized mechanism $\mathcal{M} : \mathcal{I} \to \mathcal{O}$ is (ε, δ)-DP, if for any pair of neighboring database instances $\mathbf{I} \sim \mathbf{I}'$ and any subset of outputs $O \subseteq \mathcal{O}$, it holds that

$$\Pr[\mathcal{M}(\mathbf{I}) \in O] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(\mathbf{I}') \in O] + \delta\,.$$

When $\delta = 0$, we say the mechanism satisfies *pure DP*. Otherwise (i.e., when $\delta > 0$), we say the mechanism satisifes *approximate DP*. Note that in the above definition, the concept of "neighboring" database instances $\mathbf{I} \sim \mathbf{I}'$ is left abstract, which we will clarify with concrete definitions in Section 2.2.

A common approach to answering numerical queries under differential privacy is to inject random noise into the query results, whose scale is calibrated to the *sensitivity* of the query. The widely used notion of *global sensitivity* (*GS*) considers all possible neighboring instances in the domain $\mathcal{I}$, as follows.

Definition 2.2 (Global Sensitivity [14]). Given $F : \mathcal{I} \to \mathbb{R}^d$, the global sensitivity of F is defined as

$$\mathrm{GS}_F := \max_{\mathbf{I},\mathbf{I}' \in \mathcal{I} : \mathbf{I} \sim \mathbf{I}'} \|F(\mathbf{I}) - F(\mathbf{I}')\|\,.$$

GS can be defined over L_1 or L_2 norms. The Laplace mechanism calibrates noise to the L_1-GS of the query to achieve pure DP:

Lemma 2.3 (Laplace Mechanism). *Given $F : \mathcal{I} \to \mathbb{R}^d$,*

$$\mathcal{M}_{\mathrm{Lap}}(\mathbf{I}) := F(\mathbf{I}) + b \cdot \mathrm{Lap}^d(0, 1)$$

satisfies $(\varepsilon, 0)$-DP when $b \geq \mathrm{GS}_{L_1}/\varepsilon$.

An important property of DP mechanisms is *composability*, meaning that multiple DP mechanisms can be combined to form a composite mechanism that still satisfies DP, as follows.

Lemma 2.4 (Composition Theorems [13, 14]). *Let $\mathcal{M}$ be the adaptive composition of DP algorithms $\mathcal{M}_1, \dots \mathcal{M}_k$, then*

(1) *(Basic Composition) If each $\mathcal{M}_i$ is $(\varepsilon_i, \delta_i)$-DP, then $\mathcal{M}$ is $(\sum_i \varepsilon_i, \sum_i \delta_i)$-DP.*
(2) *(Advanced Composition) If each $\mathcal{M}_i$ is (ε, δ)-DP, then for any $\delta' > 0$, $\mathcal{M}$ is $(\varepsilon\sqrt{2k\ln(1/\delta')} + k\varepsilon\frac{e^\varepsilon - 1}{e^\varepsilon + 1}, k\delta + \delta')$-DP.*

The composability properly enables DP mechanism designers to split the problem into subtasks, each with a portion of the privacy parameters ε and δ called its *privacy budget*.

Rényi DP. In Lemma 2.4, the basic composition is clean but loose for approximate DP, whereas the advanced composition is tighter but with a messy formula. Next, we introduce an alternative definition of differential privacy: Rényi DP, which has tight and clean composition, and can be easily converted to standard DP, as follows.

Definition 2.5 (Rényi Differential Privacy (RDP) [26]). For $\alpha > 1$ and $\rho > 0$, a randomized mechanism $\mathcal{M} : \mathcal{I} \to \mathcal{O}$ is (α, ρ)-RDP, if for any pair of neighboring datasets $\mathbf{I} \sim \mathbf{I}'$, it holds that

$$\mathrm{D}_\alpha(\mathcal{M}(\mathbf{I}) \| \mathcal{M}(\mathbf{I}')) \le \rho\,,$$

where $\mathrm{D}_\alpha(P\|Q) := \frac{1}{\alpha-1}\ln\int_{z\in\mathrm{supp}(Q)} P(z)^\alpha Q(z)^{1-\alpha}\,\mathrm{d}z$ is the Rényi divergence of order α.

Lemma 2.6 (RDP Composition [26]). *Let $\mathcal{M}$ be the adaptive composition of DP algorithms $\mathcal{M}_1, \dots \mathcal{M}_k$, where each $\mathcal{M}_i$ is (α, ρ_i)-RDP, then $\mathcal{M}$ is $(\alpha, \sum_i \rho_i)$-RDP.*

Lemma 2.7 (Conversion between DP and RDP [5, 26]). *If $\mathcal{M}$ is $(\varepsilon, 0)$-DP, then $\mathcal{M}$ is also $(\alpha, \frac{\alpha\varepsilon^2}{2})$-RDP for any $\alpha > 1$. If $\mathcal{M}$ is (α, ρ)-RDP, then $\mathcal{M}$ is also $\left(\rho + \frac{\ln(1/\delta)}{\alpha-1}, \delta\right)$-DP for any $\delta > 0$.*

A common mechanism for enforcing Rényi DP is the Gaussian mechanism, which is based on the concept of Gaussian Rényi divergence, as follows.

Lemma 2.8 (Gaussian Rényi Divergence [18]). *Let $\mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{1}_d)$ denote a multivariate Gaussian distribution, then*

$$\begin{aligned}&\mathrm{D}_\alpha(\mathcal{N}(\boldsymbol{\mu}_1, \sigma_1^2\mathbf{1}_d)\|\mathcal{N}(\boldsymbol{\mu}_2, \sigma_2^2\mathbf{1}_d))\\ =&\frac{\alpha}{2}\cdot\frac{\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_2^2}{(1-\alpha)\sigma_1^2 + \alpha\sigma_2^2} + \frac{d}{2(\alpha-1)}\ln\frac{\sigma_1^{2(1-\alpha)}\sigma_2^{2\alpha}}{(1-\alpha)\sigma_1^2 + \alpha\sigma_2^2}\end{aligned}$$

if $(1-\alpha)\sigma_1^2 + \alpha\sigma_2^2 > 0$, otherwise infinite.

Lemma 2.9 (Gaussian Mechanism). *Given $F : \mathcal{I} \to \mathbb{R}^d$,*

$$\mathcal{M}_{\mathrm{Gauss}}(\mathbf{I}) := F(\mathbf{I}) + \sigma\cdot\mathcal{N}(0, \mathbf{1}_d)$$

satisfies (α, ρ)-RDP when $\sigma \ge \mathrm{GS}_{L_2}\cdot\sqrt{\alpha/2\rho}$.

Local sensitivity and smooth sensitivity. Lastly, we define *local sensitivity* of a function F with respect to a given instance $\mathbf{I}$ as:

$$\mathrm{LS}_F(\mathbf{I}) := \max_{\mathbf{I}'\in\mathcal{I}:\mathbf{I}\sim\mathbf{I}'} \|F(\mathbf{I}) - F(\mathbf{I}')\|\,.$$

Sometimes, a function F's local sensitivity is far smaller than its global sensitivity, which is a situation we encounter later in Section 4. However, it is known that injecting noise directly based on local sensitivities cannot guarantee DP [33]. The following definition provides a middle ground between global and local sensitivities.

Lemma 2.10 (Smooth-sensitivity [28]). *Given $F : \mathcal{I} \to \mathbb{R}^d$ and an upper bound function $B(\mathbf{I}) \geq \mathrm{LS}_F(\mathbf{I})$, the γ-smooth-sensitivity of F is defined as*

$$\mathrm{SS}_F(\mathbf{I}, \gamma) := \max_{\mathbf{I}' \in \mathcal{I}} \{B(\mathbf{I}') \cdot e^{-\gamma \cdot \mathrm{dist}(\mathbf{I}, \mathbf{I}')}\},$$

where $\mathrm{dist}(\cdot, \cdot)$ is defined over the neighboring relationship. It satisfies $\mathrm{SS}_F(\mathbf{I}, \gamma) \geq \mathrm{LS}_F(\mathbf{I})$ and $\mathrm{SS}_F(\mathbf{I}, \gamma) \leq e^{\gamma} \cdot \mathrm{SS}_F(\mathbf{I}', \gamma)$ for any $\mathbf{I} \sim \mathbf{I}'$.

2.2 SJA Queries with User-Level DP

SJA queries. Consider a database schema $\mathcal{R} = \{R_1, \ldots, R_s\}$ where each R_i is a logical relation. A multi-way join is defined as

$$J = R_1(\mathbf{x}_1) \bowtie R_2(\mathbf{x}_2) \bowtie \cdots \bowtie R_s(\mathbf{x}_s),$$

where each $\mathbf{x}_i$ is a set of attributes. The set of variables in the full join is denoted $var(J) = \mathbf{x}_1 \cup \cdots \cup \mathbf{x}_s$.

A database instance $\mathbf{I}$ defined over $\mathcal{R}$ consists of tables $\mathbf{I}(R_1), \ldots, \mathbf{I}(R_s)$. Correspondingly, the join result $J(\mathbf{I})$ is a flat table T where each tuple takes value in $\mathrm{dom}(var(J))$. A Select-Join-Aggregate (SJA) query is defined by both the join J and an aggregation function $f : \mathrm{dom}(var(J)) \to \mathbb{R}_{\geq 0}$ that assigns a weight $f(t) = w_t$ to each join result $t \in J(\mathbf{I})$. Evaluating query f on $J(\mathbf{I})$ gives the query result

$$Q(\mathbf{I}) := f(T) = \sum_{t \in T} w_t,$$

where $T = J(\mathbf{I})$. Our target is to answer scalar or vector SJA queries. We refer to each join result $t \in T$ in the flat table as a *join tuple* or an *aggregation unit* interchangeably. We also assume the range of f is $w_t \in [0, 1]$ for simplicity. This can always be done through normalization: if $w_t \in [0, W]$, then we can consider an $f' = f/W$ and rescale the query result by W to answer f.

User-level DP for SJA queries. Next, we define user-level DP in the context of SJA query processing. Recall from Section 2.1 that in the DP definition, the concept of neighboring databases instances $\mathbf{I} \sim \mathbf{I}'$ is left abstract. In this paper, we adopt "user-level DP with foreign-key constraints" [9, 15, 23]. Specifically, we assume that there is a primary private relation $R_p \in \mathcal{R}$ where each record $u \in \mathbf{I}(R_p)$ is a private user. The set of all users is denoted as $U := \mathbf{I}(R_p)$.

We say a record $r \in \mathbf{I}(R)$ is *contributed* by user u, denoted $u \rightsquigarrow r$, following a recursive definition: 1) $u \rightsquigarrow u$; 2) if there is an $r' \in \mathbf{I}(R')$ such that $u \rightsquigarrow r'$, and there is a foreign key in r that references the primary key in r', then we have r is also contributed by u, i.e., $u \rightsquigarrow r$. Two instances $\mathbf{I}$ and $\mathbf{I}'$ are neighbors (i.e., $\mathbf{I} \sim \mathbf{I}'$), if there exists a user u such that for each record r that $\mathbf{I}$ and $\mathbf{I}'$ differ, we have $u \rightsquigarrow r$. We call such a user u a *witness*.

Let $n = |J(\mathbf{I})|$ be the number of join results, and $m = |U|$ be the number of users. Note that by our definition, a user may contribute to multiple tuples, and a tuple may be contributed by multiple users. We assume that each user can have at most Δ tuples, and that each tuple is contributed by l users. Note that this model generalizes DP definitions in the literature: when $l = 1$, it degenerates to "user-DP without self-joins" [17, 25]; when $\Delta = l = 1$, it reduces to tuple-level DP; without a constraint on Δ or l, it is user-level DP.

2.3 Existing Solutions

Although there are generic mechanisms to enforce differential privacy, e.g., Laplace and Gaussian mechanisms reviewed in Section 2.1, they cannot be directly applied to the problem of SJA query processing under user-level DP as described in Section 2.2, in which the problem setting incurs a prohibitively high global sensitivity (Definition 2.2). In particular, assume that a record can have up to D matching partners in one join operation, and that the aggregate is COUNT. With t joins, in

the worst case, one user's record u can contribute $O(D^t)$ join results, leading to a global sensitivity of $O(D^t)$, and, thus, overwhelming noise required to satisfy DP.

Trucation [22]. To reduce the global sensitivity of the SJA query, a common technique is *truncation*, which imposes a constraint on the maximum influence of a user on the query result. Formally, given a *truncation threshold* τ, we compute the result of a truncated SJA query where the contribution of any user is capped to τ, meaning that the global sensitivity of the truncated query is bounded by τ [22]. This step involves solving a linear program [22], which minimizes the error after injecting noise to satisfy DP.

Clearly, the result of the truncated SJA query can differ from the original query, which we call the *truncation bias*. Hence, the truncation threshold τ is a critical system parameter that balances truncation bias with DP noise: a larger τ reduces the truncation error, but at the same leads to a larger global sensitivity, and, thus, higher noise needed to satisfy DP, and vice versa. The optimal value for τ depends upon the maximum contribution of a user in the dataset, called the *downward sensitivity*, defined as

$$\mathrm{DS}_F(\mathbf{I}) := \max_{\mathbf{I}' \subseteq \mathbf{I}: \mathbf{I} \sim \mathbf{I}'} \|F(\mathbf{I}) - F(\mathbf{I}')\|.$$

Note that choosing the truncation threshold τ using the exact downward sensitivity $\mathrm{DS}_F(\mathbf{I})$ would violate differential privacy [9, 22], since $\mathrm{DS}_F(\mathbf{I})$ itself may depend upon private information.

R2T [9]. The R2T algorithm chooses an appropriate trucation threshold τ by considering all the candidate τ's that are powers of 2, each of which is allocated with a portion of the privacy budget. For every τ, the algorithm solves the linear program in [22], whose target value gives the truncated query result. Then, R2T injects random noise according to the corresponding τ's to satisfy DP. After subtracting the error bound, all of them become under-estimates with high probability, and the maximum among them provides a balance between the truncation error and the DP noise. Since the candidate τ's are powers of 2, the number of candidates is logarithmic; accordingly, the authors of [9] prove that the utility guarantee of R2T matches the instance-optimal lower bound $\Omega(\mathrm{DS}(\mathbf{I}))$ up to polylogarithmic factors.

PMSJA [10]. R2T is designed for *scalar* queries that return a single value. For SJA queries that return *vectors* (e.g., GROUP BY aggregates), a natural idea is to exploit the composition property in Lemma 2.4 to split the privacy budget among the vector entries. In particular, using advanced composition in Lemma 2.4, the L_2 error of the noisy result is then $\tilde{O}\left(\sqrt{d} \cdot \sqrt{\sum_{i=1}^{d} \mathrm{DS}_i^2(\mathbf{I})}\right)$, where $\mathrm{DS}_i(\mathbf{I})$ is the maximum contribution of an user to the i-th entry. This is optimal if the user with maximum contribution is the same across all entries. However, if each user contributes to a different entry, this error becomes $\tilde{O}(d)$, whereas the optimal bound is $\tilde{O}(\sqrt{d})$ as $\mathrm{DS}_F(\mathbf{I}) = 1$. To address this issue, the PMSJA algorithm [10] resorts to using a quadratically constrained quadratic program (QCQP) combined with local-sensitivity based mechanisms to achieve instance-optimal error $\tilde{O}(\sqrt{d} \cdot \mathrm{DS}_F(\mathbf{I}))$, which we revisit in Section 4.

S&E [15]. The user-level DP solutions presented so far for SJA queries all involve solving expensive optimization programs, which are inherently difficult to scale. To reduce the computational costs, *sample-and-explore* (S&E) applies sampling to the set of users.

Specifically, S&E takes as input the number of iterations; in each iteration, it samples a user uniformly at random. The algorithm then builds an "explored" instance containing all the aggregation units where the sampled user appears first. After that, a DP mechanism (e.g. R2T or PMSJA) is applied on the explored instance. To obtain the estimator for an iteration, the output is scaled back by the number of users m, which is also the inverse of the probability that each aggregation unit is sampled. S&E then takes the average over multiple iterations to reduce the sample variance.

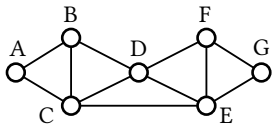


Fig. 1. A social network G.

User	Triangle Count	
	before trunc.	after trunc.
A, G	1	1
B, F	2	2
C, D, E	3	2

Table 1. Users' triangle counts in G.

Triangle	Weight
$\triangle_{ABC}$, $\triangle_{EFG}$	1
$\triangle_{BCD}$, $\triangle_{DEF}$	1
$\triangle_{CDE}$	0

Table 2. A truncation with $\tau = 2$.

The privacy guarantee of S&E relies on a given upper bound C on the number of *collaborators* of each user. Two users *collaborate* if they contribute to the same aggregation unit. The intuition is that on a neighboring dataset with an extra user, the explored instance will be the same unless the sampled user happens to be the witness or one of their collaborators, which occurs with probability approximately C/m.

A major issue for S&E is that its privacy cost is linear to C. which leads to large errors as we show in Section 3.3. The proposed approach DP-S4S, presented in the next section, overcomes this problem. In particular, our amplification bound has no assumption on C, and achieves a strict amplification in the privacy budget. As a result, the result utility of DP-S4S may even surpass that of the R2T mechanism *without sampling* in some cases, as shown in our experiments in Section 5.

3 DP-S4S for Scalar Queries

This section presents DP-S4S for scalar SJA queries that return a single aggregate value. The extension to vector SJA queries is deferred to Section 4. The main idea is that instead of sampling users as in S&E, DP-S4S samples *aggregate units*, i.e., the join tuples upon which the scalar query is aggregating.

For example, consider a triangle count query with pure DP in a social network G, illustrated in Figure 1, where each vertex is a user. Table 1 lists the number of triangles contributed by each user, before and after applying a truncation threshold of 2. The 5 triangles shown in Table 2 are the aggregate units. DP-S4S first constructs a sample set S of such aggregate units, and then applies truncations on S to reduce the L_1 sensitivity of the aggregation on S; after that, it generates the aggregation result on the truncated S, injects Laplace noise, and then returns the noisy result scaled up by a factor decided by the sampling rate. The truncation threshold τ used on S is decided using a differentially private algorithm, which ensures that the choice of τ does not lead to a privacy breach.

In the following, we first focus on pure DP (i.e., $(\varepsilon, 0)$-DP) in Sections 3.1-3.3, and extend the result to approximate DP in Section 3.4. In particular, Section 3.1 explains the truncation of a sample set S given a threshold τ. Section 3.2 focuses on the DP algorithm for choosing τ. Section 3.3 analyzes the theoretical guarantee of DP-S4S and compares against existing solutions.

3.1 Sample and Truncate

Recall from Section 2.2 that for a query f over the join tuples T, our target is to compute $f(T) := \sum_{t \in T} w_t$. Similar to Dong et al. [9], the proposed mechanism DP-S4S applies after computing all the joins and before the final aggregation. Specifically, DP-S4S samples each join tuple $t \in T$ with probability q to form S, and rescales the aggregation result $f(S)$ by $1/q$ as an estimator for $f(T)$. Thus, the problem reduces to how to release $f(S)$ accurately with user-level DP. Given a truncation threshold τ, we can simply apply the steps in [9, 22] reviewed in Section 2.2, i.e., solving a linear program to compute the optimal truncation for each user, and then applying the Laplace mechanism to achieve pure DP.

Algorithm 1 Sample-and-Truncate for pure DP

Input: Join tuples T, query f, truncation threshold τ, sample rate q, privacy parameter ε
Output: Private query result $\hat{f}(T, \tau)$

1: $S \leftarrow \emptyset$
2: **for each** $t \in T$ **do**
3: Sample t into S with probability q
4: $\bar{f}(S, \tau) \leftarrow$ Optimal target value to problem (1) on S, τ
5: Release $\hat{f}(S, \tau) \leftarrow \bar{f}(S, \tau) + \mathrm{Lap}(\frac{\tau}{\varepsilon})$
6: Report $\hat{f}(T, \tau) \leftarrow \frac{1}{q} \cdot \hat{f}(S, \tau)$

Formally, Algorithm 1 presents the above algorithm for releasing $f(T, \tau)$ for a given threshold τ under $(\varepsilon, 0)$-DP. The linear program for computing optimal truncation [9, 22] is as follows.

$$\begin{aligned} \underset{x_t}{\text{maximize}} \quad & \sum_{t \in S} x_t & & (1) \\ \text{subject to} \quad & \sum_{t : u \rightsquigarrow t} x_t \le \tau, & \forall u \in U \\ & 0 \le x_t \le w_t & \forall t \in S \end{aligned}$$

where each variable $x_t \in [0, w_t]$ is the fractional contribution of join tuple t in the sample set S to the truncated query, subject to the constraint that the total contribution of u does not exceed the truncation threshold τ. For a given query, minimizing the truncation bias is equivalent to maximizing the remaining weights; thus, the optimal target value $\bar{f}(S, \tau) := \sum_{t \in S} x_t^*$ is used as the truncated query result. Note that for any $\tau \ge \mathrm{DS}(\mathbf{I})$ (see Section 2.3), the optimal solution is $x_t^* = w_t$ and we have $\bar{f}(S, \tau) = \sum_{t \in S} w_t = f(S)$.

After obtaining the result of the truncated query, the algorithm proceeds to add Laplace noise with scale τ/ε (Line 5), and then scale the result back by $1/q$ to correct for sampling (Line 6).

Although Algorithm 1 is simple, its privacy analysis is highly non-trivial. A major contribution of this work is to show that sampling join tuples (rather than users), followed by an ε-user-level DP mechanism on the sample, preserves a stronger ε'-user-level DP guarantee, for $\varepsilon' \le \varepsilon$. Note that prior work has only considered either sampling users for user-level DP [15] or tuples for tuple-level DP [3], but not a combination of both. In addition, when the subsequent pure DP mechanism is truncation-based, we can further achieve an *amplified* privacy guarantee due to randomness in subsampling. Formally, we have the following result 3.1.

Lemma 3.1 (Sample-and-Truncate Amplification). *Algorithm 1 satisfies ε'-user-level DP for*

$$\varepsilon'(\varepsilon, \tau, \Delta, q) = \ln\Bigg(\max\Bigg\{ \sum_{k=0}^{\lfloor \tau \rfloor} p_k \exp\left(\frac{k}{\tau}\varepsilon\right) + \Pr[\mathrm{Bin}(\Delta, q) > \tau] \exp(\varepsilon), \\ \frac{1}{\sum_{k=0}^{\lfloor \tau \rfloor} p_k \exp\left(-\frac{k}{\tau}\varepsilon\right) + \Pr[\mathrm{Bin}(\Delta, q) > \tau] \exp(-\varepsilon)} \Bigg\} \Bigg),$$

where $p_k = \Pr[\mathrm{Bin}(\Delta, q) = k] = \binom{\Delta}{k} q^k (1-q)^{\Delta - k}$.

Proof. We will show that $\mathcal{M}(T) = \hat{f}(S, \tau)$ is private, as the final step is a post-processing. Denote the independent Bernoulli (a.k.a. Poisson) subsampling process by λ, then the probability

of observing output z can be expressed as[1]

$$\Pr[\mathcal{M}(T) = z] = \sum_{S \subseteq T} \Pr[\lambda(T) = S] \cdot \Pr[\hat{f}(S, \tau) = z] .$$

Consider a user-level neighbor $T' = T \cup T_u$ where the witness is u and T_u are the records contributed by u. WLOG, we assume $|T_u| = \Delta$, which brings the maximum divergence. Due to independent sampling,

$$\Pr[\mathcal{M}(T') = z] = \sum_{S \subseteq T} \Pr[\lambda(T) = S] \cdot \sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \cdot \Pr[\hat{f}(S \cup S_u, \tau) = z] .$$

We will use the fact that for $p_i, A_i, B_i > 0$,

$$\frac{\sum_i p_i A_i}{\sum_i p_i B_i} \leq \max_i \frac{A_i}{B_i} ,$$

which can be verified by $A_i \leq B_i \cdot \max_i \frac{A_i}{B_i}$ for any i. Borrowing ideas from [20], we consider

$$\begin{aligned}
\frac{\Pr[\mathcal{M}(T) = z]}{\Pr[\mathcal{M}(T') = z]} &= \frac{\sum_{S \subseteq T} \Pr[\lambda(T) = S] \cdot \Pr[\hat{f}(S, \tau) = z]}{\sum_{S \subseteq T} \Pr[\lambda(T) = S] \cdot \sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \Pr[\hat{f}(S \cup S_u, \tau) = z]} \\
&\leq \max_{S \subseteq T} \frac{\Pr[\hat{f}(S, \tau) = z]}{\sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \Pr[\hat{f}(S \cup S_u, \tau) = z]} \\
&= \frac{1}{\sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \frac{\Pr[\hat{f}(S^* \cup S_u, \tau) = z]}{\Pr[\hat{f}(S^*, \tau) = z]}} ,
\end{aligned}$$

where $S^* = \arg\max_{S \subseteq T} \frac{\Pr[\hat{f}(S,\tau)=z]}{\sum_{S_u \subseteq T_u} \Pr[\lambda(T_u)=S_u] \Pr[\hat{f}(S \cup S_u,\tau)=z]}$. Consider the optimal solution $\{x_t\}$ that achieves S^*, it is also feasible for $S^* \cup S_u$, thus $\bar{f}(S^*, \tau) \leq \bar{f}(S^* \cup S_u, \tau)$. On the other hand, given an optimal solution to $S^* \cup S_u$, we can obtain a feasible solution for S^* by setting $x'_t = 0$ for $t \in S_u$. In doing so, the weight reduced is $\sum_{t \in S_u} x_t \leq \min\{\tau, |S_u|\}$, since $w_t \in [0, 1]$ (see Section 2.2). Thus, we also have $\bar{f}(S^* \cup S_u, \tau) \leq \bar{f}(S^*) + \min\{\tau, |S_u|\}$. In summary, we have $|\bar{f}(S^* \cup S_u, \tau) - \bar{f}(S^*)| \leq \min\{\tau, |S_u|\}$ for any S_u.

Based on the above fact, we have

$$\frac{\Pr[\hat{f}(S^* \cup S_u, \tau) = z]}{\Pr[\hat{f}(S^*, \tau) = z]} \geq \exp\left(-\frac{\varepsilon}{\tau} \left|\bar{f}(S^* \cup S_u, \tau) - \bar{f}(S^*, \tau)\right|\right) \geq \exp\left(-\frac{\min\{\tau, |S_u|\}}{\tau}\varepsilon\right) ,$$

thus,

$$\frac{\Pr[\mathcal{M}(T) = z]}{\Pr[\mathcal{M}(T') = z]} \leq \frac{1}{\sum_{k=0}^{\Delta} p_k \cdot \exp(-\frac{\min\{\tau,k\}}{\tau}\varepsilon)} ,$$

[1]We abuse the notation of $\Pr[\cdot]$ for probability density as with convention.

where $p_k = \Pr[\mathrm{Bin}(\Delta, q) = k] = \binom{\Delta}{k} q^k (1-q)^{\Delta-k}$. Similarly,

$$\begin{aligned}\frac{\Pr[\mathcal{M}(T') = z]}{\Pr[\mathcal{M}(T) = z]} &\le \max_{S \subseteq T} \frac{\sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \Pr[\hat{f}(S \cup S_u, \tau) = z]}{\Pr[\hat{f}(S, \tau) = z]} \\ &= \sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \frac{\Pr[\hat{f}(S^\circ \cup S_u, \tau) = z]}{\Pr[\hat{f}(S^\circ, \tau) = z]} \\ &\le \sum_{k=0}^{\Delta} p_k \cdot \exp\left(\frac{\varepsilon}{\tau} \min\{\tau, k\}\right),\end{aligned}$$

where $S^\circ = \arg\max_{S \subseteq T} \frac{\sum_{S_u \subseteq T_u} \Pr[\lambda(T_u) = S_u] \Pr[\hat{f}(S \cup S_u, \tau) = z]}{\Pr[\hat{f}(S, \tau) = z]}$.

Putting it together, we get Lemma 3.1. □

Optimization with a PGF trick. Lemma 3.1 requires computing $\sum_{k=0}^{\lfloor \tau \rfloor} p_k \exp\left(\frac{k}{\tau}\varepsilon\right)$ and $\sum_{k=0}^{\lfloor \tau \rfloor} p_k \exp\left(-\frac{k}{\tau}\varepsilon\right)$, which takes $O(\tau) = O(\Delta)$ time, which can be expensive when τ is large. We show that with the help of probability-generating function (PGF), we can obtain a numerically stable result in $O(1)$ time, assuming the CDF function of a binomial distribution can be evaluated in $O(1)$.

Let $K \sim \mathrm{Bin}(\Delta, q)$, consider the PGF of K, defined as

$$G(z) := \mathbf{E}[z^K] = \sum_{k=0}^{\Delta} p_k \cdot z^k = \sum_{k=0}^{\Delta} \binom{\Delta}{k} (1-q)^{\Delta-k} (qz)^k = (1 - q + qz)^\Delta .$$

WLOG, we assume an integer τ. Then the target is to compute the sum of the first $\tau + 1$ terms in $G(z)$ for $z = \exp(\pm\frac{\varepsilon}{\tau})$. Let $A = 1 - q + qz$. We divide each summand by A^Δ. Then, the k-th term becomes

$$\binom{\Delta}{k} \left(\frac{1-q}{A}\right)^{\Delta-k} \left(\frac{qz}{A}\right)^k = \binom{\Delta}{k} \left(1 - \frac{qz}{A}\right)^{\Delta-k} \left(\frac{qz}{A}\right)^k = \Pr\left[\mathrm{Bin}(\Delta, \frac{qz}{A}) = k\right],$$

which is exactly the probability of another binomial distribution. Taking the sum is equivalent to querying its CDF. Accordingly, we present in Lemma 3.2 a faster computation of the amplification.

Lemma 3.2 (Simplified Computation). *Algorithm 1 satisfies ε'-user-level DP for*

$$\begin{aligned}\varepsilon'(\varepsilon, \tau, \Delta, q) = \max \Big\{ &(\mathrm{logcdf}(\tau, \Delta, q_1) + \Delta \cdot \log A_1) \oplus (\mathrm{logsf}(\tau, \Delta, q) + \varepsilon), \\ &- \Big((\mathrm{logcdf}(\tau, \Delta, q_2) + \Delta \cdot \log A_2) \oplus (\mathrm{logsf}(\tau, \Delta, q) - \varepsilon) \Big) \Big\},\end{aligned}$$

where $\oplus$ denotes the LogSumExp *function,* logcdf *and* logsf *are the cumulative distribution function and survival function for the binomial distribution respectively, $A_1 = 1 - q + q \cdot \exp(\varepsilon/\tau)$, $q_1 = q \cdot \exp(\varepsilon/\tau)/A_1$; $A_2 = 1 - q + q \cdot \exp(-\varepsilon/\tau)$, $q_2 = q \cdot \exp(-\varepsilon/\tau)/A_2$.*

3.2 Truncation Threshold Selection

The Sample-and-Truncate mechanism presented in Algorithm 1 requires the knowledge of the truncation threshold τ. As explained in Section 2.3, choosing an appropriate τ under user-level DP is highly non-trivial, and it is a major contribution of R2T [9]. In this subsection, we carefully combine the doubling research solution in [9] to Sample-and-Truncate, leading to the complete DP-S4S for scalar SJA query with pure DP in Algorithm 2.

Algorithm 2 DP-S4S for scalar SJA and pure DP

Input: Join tuples T, query f, sample rate q, privacy parameter ε, tuple bound Δ, probability β
Output: Private query result $\hat{f}(T)$

1: Compute sample set S with Lines 1-3 of Algorithm 1.
2: $\varepsilon_0 \leftarrow \varepsilon$
3: $L \leftarrow \lfloor \log \Delta \rfloor + 1$
4: **for** $i \leftarrow L$ downto 1 **do**
5: $\quad \tau_i \leftarrow 2^{i-1}$, $\varepsilon_i \leftarrow \varepsilon_0 / i$
6: $\quad \bar{f}(S, \tau_i) \leftarrow$ Optimal target value to problem (1) on S, τ_i
7: $\quad \hat{f}(S, \tau_i) \leftarrow \bar{f}(S, \tau_i) + \mathrm{Lap}(\frac{\tau_i}{\varepsilon_i})$
8: $\quad \tilde{f}(S, \tau_i) \leftarrow \hat{f}(S, \tau_i) - \frac{\tau_i}{\varepsilon_i} \ln \frac{3L}{\beta}$
9: $\quad \varepsilon_i' \leftarrow \varepsilon'(\varepsilon_i, \tau_i, \Delta, q)$ by Lemma 3.2
10: $\quad \varepsilon_0 \leftarrow \varepsilon_0 - \varepsilon_i'$
11: Report $\hat{f}(T) \leftarrow \frac{1}{q} \cdot \max_i \tilde{f}(S, \tau_i)$

Recall from Lemma 3.1 that $\varepsilon' \leq \varepsilon$, and that a mechanism on the sampled join tuples preserves user-level pure DP. Therefore, a naive solution is to allocate $\varepsilon_i = \varepsilon / L$ to each iteration. However, this approach fails to exploit the amplification from combining a truncation mechanism with subsampling. Ideally, one wish to adopt some $\varepsilon^* > \varepsilon$ such that the amplified $\varepsilon'(\varepsilon^*, \tau, \Delta, q) \leq \varepsilon$. While the amplified $\varepsilon'(\varepsilon, \tau, \Delta, q)$ is easily computable for a given ε, finding the optimal ε^* involves a binary search. Observe from Lemma 3.1 that the amplification effect is more significant for large τ's. Accordingly, we adopt an *adaptive budget allocation* strategy in DP-S4S. In particular, we run the L subroutines in descending order of τ_i (Line 4 of Algorithm 2). For each τ_i, we allocate a target privacy consumption ε_i inversely proportional to the number of remaining subproblems, i.e. applying basic composition in Lemma 2.4. After obtaining the private result, we only charge its actual consumption from the remaining budgets (Lines 9-10). Note that the algorithm also takes as input a probability β (used in Line 8), which is important for its utility analysis, presented in the next subsection.

Remark. In [15], the authors also propose a faster variant of S&E for scalar queries that explicitly requires that a good truncation threshold τ be given in advance; in their experiments, τ is tuned from data, which might leak private information according to [9]. In this paper, we follow [9] to properly select τ with differential privacy, and do not consider this variant of S&E.

3.3 Utility Analysis

The following lemma establishes the correctness of DP-S4S for scalar SJA queries under user-level pure DP, as well as its result utility guarantee. The error bound covers both sampling error and noise injected to satisfy DP.

Lemma 3.3. *Algorithm 2 satisfies $(\varepsilon, 0)$-DP. Let n be the number of records, m be the number of users and q be the sample rate. With probability $1 - \beta$, its error is*

$$|\hat{f}(T) - f(T)| = \tilde{O}\left(\sqrt{\frac{n}{q}} + \frac{1}{q} + \frac{1}{\varepsilon_{i^*}}\left(\tau^*(T) + \sqrt{\frac{\tau^*(T)}{q}}\right)\right),$$

Algorithm 3 Sample-and-Truncate for Rényi DP

Input: Join tuples T, query f, truncation threshold τ, sample rate q, privacy parameters α and ρ
Output: Private query result $\hat{f}(T, \tau)$
1: Compute $\bar{f}(S, \tau)$ with Lines 1-4 in Algorithm 1
2: Release $\hat{f}(S, \tau) \leftarrow \bar{f}(S, \tau) + \mathcal{N}(0, \sigma^2)$ for $\sigma = \tau \cdot \sqrt{\alpha/2\rho}$
3: Report $\hat{f}(T, \tau) \leftarrow \frac{1}{q} \cdot \hat{f}(S, \tau)$ as in Algorithm 1

where $\tilde{O}(\cdot)$ omits constants and polylogarithmic factors in m and $1/\beta$; $\tau^*(T) := \max_{u \in U} \sum_{t \in T} \mathbf{1}[u \rightsquigarrow t]$ is the maximum number of tuples a user contribute to in T; and $\varepsilon_{i^*} \geq \varepsilon/L$ is the allocated noise-calibration budget where $L = \lfloor \log \Delta \rfloor + 1$.

The proof of Lemma 3.3 can be found in Appendix B.1 of the full version [30]. As a sketch, main ideas of the proof include (i) we apply standard Bernstein bounds for the sampling error and (ii) for DP noise, conditioned on the high probability event that $\tilde{f}$'s are all underestimates, we show that the output error depends on $\tau^*(S)$, i.e., the maximum contribution of a user in the sample. A key observation in the proof is that besides the amplification effect that reduces the DP noise scale (see Lemma 3.1), another benefit of sampling in DP-S4S is that it reduces the maximum contribution among the users. Roughly speaking, if the maximum contribution of a user is $\tau^*(T)$ for the raw join results, then we expect the maximum contribution to be $\tau^*(S) \approx q \cdot \tau^*(T)$ in the sample, subject to some lower-order terms. After scaling the results by $\frac{1}{q}$, we obtain $\tau^*(T)$, so that the DP error does not increase. Appendix C in [30] evaluates the contribution of both error sources.

Comparison with R2T. The utility guarantee of R2T is as follows.

Lemma 3.4 (Result Utility of R2T [9]). *With probability $1 - \beta$, the output of the* R2T *mechanism satisfies*

$$|\mathcal{M}_{R2T}(T) - f(T)| = \tilde{O}\left(\frac{L}{\varepsilon} \cdot \mathrm{DS}(\mathbf{I})\right).$$

Compared to R2T, DP-S4S introduces an additional error term $\sqrt{n/q}$ for $q = \Omega((\tau^*(T))^{-1})$, due to sampling error. For counting queries, $\tau^*(T) = \mathrm{DS}(\mathbf{I})$ is the downward sensitivity of query f (see Section 2.3). Thus, the sampling-based mechanism achieves the same asymptotic error for a sufficiently large sample rate. In addition, we improve the $\frac{L}{\varepsilon}$ term in the DP error to $\frac{1}{\varepsilon_{i^*}} \leq \frac{L}{\varepsilon}$, which benefits from the privacy amplification. To see why, note that the first iteration allocates exactly $\varepsilon_L = \varepsilon/L$. Meanwhile, its amplified privacy guarantee $\varepsilon'_L \leq \varepsilon_L$, meaning the remaining budget $\varepsilon_0 \geq \varepsilon - \varepsilon/L = \varepsilon \cdot (1 - 1/L)$. By induction, we have in each iteration the allocated ε_i is at least as large as ε/L.

Comparison with S&E. Fang and Yi [15] do not include a closed-form error bound for S&E under pure DP. In the following, we show that its error can be large even when the exact (i.e., non-private) best truncation threshold is used.

Specifically, the error analysis in [15] for running S&E with R2T focuses on approximate DP. It is not hard to see that when each of the k iterations is allocated with a privacy budget of $\varepsilon_i = O(\varepsilon/k)$, the mechanism satisfies pure DP, according to basic composition (Lemma 2.4). When ε_i is small, the amplified ε'_i for each iteration can be approximated as

$$\varepsilon'_i = \ln\left(1 + \frac{m}{C}\left(e^{\varepsilon_i} - 1\right)\right) = O\left(\frac{m\varepsilon}{kC}\right),$$

where m is the inverse of the sample rate. The final estimator scales the sample average by m. Thus, its variance from DP noise is

$$\frac{m^2}{k^2} \cdot \left(\sum_{i=1}^{k} \frac{2\tau^2}{(\varepsilon_i')^2} \right) = O\left(kC^2 \cdot \frac{\tau^2}{\varepsilon^2} \right).$$

Thus, the injected error to satisfy pure DP grows with $\sqrt{k}$, which becomes high when we set a large k, which is important for reducing sampling error. More importantly, its DP error is also linear to C, the maximum number of collaborators for each user (see Section 2.3), which is data-dependent and can be large, as our experiments show in Section 5. The proposed approach DP-S4S does not suffer from either of these issues, according to Lemma 3.3.

Finally, as discussed in Appendix D of the full version [30], the sampling error of S&E is also a factor of l larger compared to our DP-S4S, since the former takes *all* tuples associated with the sampled user, whereas the latter performs sampling on aggregation units, which automatically takes into account the correlation between users.

3.4 DP-S4S for Approximate DP

So far, our description of DP-S4S aims at enforcing pure DP, i.e., $\delta = 0$. For approximate DP, i.e., $\delta > 0$, we adapt DP-S4S to employ the Gaussian mechanism (Lemma 2.9) to satisfy Rényi-DP, and subsequently convert Rényi DP to (ε, δ)-DP using Lemma 2.7. Algorithm 3 adapts Algorithm 1 to enforce Rényi DP.

Under Rényi-DP, we derive privacy amplification as follows.

Lemma 3.5 (RDP Amplification). *Algorithm 3 satisfies (α, ρ')-user-level RDP for*

$$\rho'(\alpha, \rho, \tau, \Delta, q) = \frac{1}{\alpha - 1} \ln \left(\sum_{k=0}^{\lfloor \tau \rfloor} p_k \cdot \exp\left((\alpha - 1) \frac{k^2}{\tau^2} \rho \right) + \Pr[\mathrm{Bin}(\Delta, q) > \tau] \cdot \exp((\alpha - 1)\rho) \right),$$

where $p_k = \Pr[\mathrm{Bin}(\Delta, q) = k] = \binom{\Delta}{k} q^k (1-q)^{\Delta - k}$.

The proof can be found in Appendix B.2 of the full version [30], which is similar to that for Lemma 3.1. The difference is that we apply properties of Rényi divergence instead of analyzing the ratio between probabilities. We also have $\rho' \le \frac{1}{\alpha - 1} \ln(\sum_{k=0}^{\Delta} p_k \cdot \exp((\alpha - 1)\rho)) = \rho$.[2]

Using Lemma 3.5 as a building block, we can arrive at a variant of Algorithm 2 that achieves (α, ρ)-RDP, which implies (ε, δ)-DP: we only need to change the noise and noise bound to Gaussian, and allocate ρ_i instead of ε_i during each iteration. For the utility analysis, note that the sampling error is independent of the noise, and we also have the same bound for the user contribution in the sample. Thus, its utility is given as follows.

Corollary 3.6 (RDP-S4S for scalar SJA). *There is an (α, ρ)-RDP version of Algorithm 2. Let n be the number of records, m be the number of users, q be the sample rate and $\tau^*(T)$ be the maximum contribution of a user. With probability $1 - \beta$, its error is*

$$|\hat{f}(T) - f(T)| = \tilde{O}\left(\sqrt{\frac{n}{q}} + \frac{1}{q} + \sqrt{\frac{\alpha}{\rho_{i^*}}} \left(\tau^*(T) + \sqrt{\frac{\tau^*(T)}{q}} \right) \right),$$

where $\rho_{i^*} \ge \rho / (\lfloor \log \Delta \rfloor + 1)$ is the amplified privacy budget.

[2]While we show that sampling join tuples amplifies user-level DP under both pure and Rényi DP, the same proof does not easily extend to (ε, δ)-DP mechanisms outside the Rényi-DP class. It is left open whether an arbitrary approximate-DP mechanism on such a sample preserves the same level of user-level privacy.

In particular, this indicates an (ε, δ)-DP variant by taking $\rho = \varepsilon - \frac{\ln(1/\delta)}{\alpha-1}$. Now, the DP error scales with $\sqrt{\frac{\alpha(\alpha-1)}{\varepsilon(\alpha-1)-\ln(1/\delta)}}$, and the order α can be optimized for given ε and δ.

4 DP-S4S for Vector Queries

This section focuses on DP-S4S for *vector* queries that return $d > 1$ scalars (e.g., a group-by count query). Observe that a vector query F can be viewed as d scalar queries $F = \{f_1, \ldots, f_d\}$.

As discussed in Section 2.3, a naive idea for answering a vector SJA queries with user-level DP is to process each component scalar query separately, each with an allocated privacy budget according to Lemma 2.4. The problem of this approach, however, is that its error depends on the sensitivity of each scalar, rather than the sensitivity of the whole vector, which can be significantly smaller. In particular, the L_2 sensivitity for a d-dimensional vector query is smaller than that of its component scalar queries by a factor of $\sqrt{d}$ [10]. PMSJA [10] address this issue by considering the following quadratically constrained quadratic program (QCQP):

$$\begin{aligned}
\underset{y_u, z_{f,t}}{\text{maximize}} \quad & \sum_{u \in U} y_u && && (2)\\
\text{subject to} \quad & \sum_{f \in F} \left(\sum_{t \in T_f : u \rightsquigarrow t} w_{f,t} \cdot z_{f,t} \right)^2 \le \tau^2, && \forall u \in U \\
& \sum_{u : u \rightsquigarrow t} (y_u - 1) \le z_{f,t} - 1, && \forall f \in F, t \in T_f \\
& 0 \le y_u \le 1, && \forall u \in U \\
& 0 \le z_{f,t} \le 1, && \forall f \in F, t \in T_f
\end{aligned}$$

For given τ, let $y^*_u, z^*_{f,t}$ be the optimal solution to Program (2) (with ties broken arbitrarily). The truncated query for $f \in F$ is then

$$\bar{f}(T_f) := \textstyle\sum_{t \in T_f} w_{f,t} \cdot z^*_{f,t}. \tag{3}$$

Unlike in R2T, the global sensitivity of the truncated queries $\bar{F}$ is still large. To address this issue, PMSJA considers the local sensitivity of $\bar{F}$ (explained in Section 2.1). Analysis shows that the L_2 local sensitivity satisfies:

$$\mathrm{LS}_{\bar{F}}(\mathbf{I}) \le 2\tau \cdot (E(\mathbf{I}) + 1), \tag{4}$$

where $E(\mathbf{I}) := \sum_{u \in U}(1 - y^*_u)$ is the number of truncated users. Intuitively, $E(\mathbf{I})$ has global sensitivity 1, since the optimal solution on $\mathbf{I}$ can be transferred into a feasible solution on $\mathbf{I}'$ which contains one more user, by additionally truncating the extra user and all his/her tuples. The formal proof can be found in [10]. PMSJA then invokes an (ε, δ)-DP algorithm to release $\bar{F}$. For choosing the threshold τ, it runs the sparse-vector-technique (SVT) mechanism [13] to privately find the smallest power of 2 such that only $E(\mathbf{I}) = \tilde{O}(1)$ users are truncated.

To achieve scalability, the main idea of DP-S4S is to combine join-tuple sampling described in Section 3 with PMSJA. Note that PMSJA enforces approximate DP; hence, we adopt the approach in Section 3.4 that processes the query under Rényi-DP, and subsequently converts the privacy guarantee to the standard (ε, δ)-DP.

Challenges. There are two major challenges in realizing this idea. First, as mentioned above, PMSJA involves the use of local sensitivity, which cannot be directly used to calibrate DP noise as described in Section 2.1. For (ε, δ)-DP, we can inject noise according to the smooth sensitivity (Lemma 2.10), which provides a viable middle ground that satisfies differential privacy while

reducing the noise scale compared to global sensitivity. To our knowledge, however, there is no existing fundamental mechanism that utilizes smooth sensitivities to achieve Rényi DP, which is more strict than (ε, δ)-DP. Second, even with the necessary tools in place to enforce Rényi DP with smooth sensitivity, it is still highly non-trivial to derive a similar sample-and-truncate algorithm with privacy amplification for PMSJA, which involves solving a far more complex QCQP than the linear program in Section 3.1.

In the following, Sections 4.1 and 4.2 address these two challenges, respectively.

4.1 Smooth Sensitivity Mechanism for Rényi DP

In this subsection, we build a new fundamental Rényi DP mechanism for the scenario that the local sensitivity is far smaller than the global sensitivity, i.e., $\mathrm{LS}_{L_2} \ll \mathrm{GS}_{L_2}$. This is common in practical applications [9, 10, 13, 15, 33]. Utilizing Lemma 2.8, we have the following result.

Lemma 4.1 (RDP Smooth Sensitivity Mechanism). *Given $F : \mathcal{I} \to \mathbb{R}^d$ and its smooth-sensitivity function $\mathrm{SS}_F : \mathcal{I} \times \mathbb{R} \to \mathbb{R}$ as in Lemma 2.10, the RDP-SS mechanism defined as*

$$\mathcal{M}_{\text{RDP-SS}}(\mathbf{I}) := F(\mathbf{I}) + \sigma(\mathbf{I}) \cdot \mathcal{N}(0, \mathbf{1}_d)$$

satisfies (α, ρ)-RDP for

$$\sigma(\mathbf{I}) = \frac{\mathrm{SS}_F(\mathbf{I}, \gamma)}{1 - \alpha + \alpha e^{-2\gamma}} \cdot \sqrt{\frac{\alpha}{\rho}},$$

where $\gamma := \frac{1}{2} \ln \min\{t_1^{-1}, t_2\}$, and $0 < t_1 < 1 < t_2$ are solutions to

$$h(t) := t^\alpha - \alpha e^{\frac{(\alpha-1)\rho}{d}} t + (\alpha - 1) e^{\frac{(\alpha-1)\rho}{d}} = 0\,.$$

The proof can be found in Appendix B.3 of the full version [30]. The intuition is that to bound the Rényi-divergence between two Gaussian distributions, there are two constraints in Lemma 2.8: the noise scales must be large enough to bound the first term, which depends on the difference between their means. In addition, the scales must also be multiplicatively close to ensure the second term is bounded: when $\sigma_1 = \sigma_2$, the second term takes the minimum value of 0; when the variances differ, we must bound their ratio, which can be achieved by properly setting the smoothness factor γ. After deciding γ (see the proof for details), we calibrate the noise scale according to both γ and the γ-smooth sensitivity SS_F.

Implementation. A major challenge in implementing smooth-sensitivity mechanisms is that the SS_F function is hard to compute in general [28]. In the following, we make an assumption that there exists an upper bound B on the L_2 local sensitivity LS_F such that $B(\mathbf{I}) \geq \mathrm{LS}_F(\mathbf{I})$ and $|B(\mathbf{I}) - B(\mathbf{I}')| \leq 1$ for neighboring datasets. Under this assumption, we can compute SS_F in constant time. We first argue that this is a feasible assumption, which was also made by existing work [21, 22, 33]. A trivial bound that satisfies this constraint is a constant function $B(\cdot) \equiv \mathrm{GS}$. But it does not utilize the fact that $\mathrm{LS} \ll \mathrm{GS}$. To reduce noise, the goal is to set B as close to LS as possible. Note that due to normalization, requiring $|B(\mathbf{I}) - B(\mathbf{I}')| \leq 1$ is equivalent to the more general form of $|B(\mathbf{I}) - B(\mathbf{I}')| \leq \tau$. This is exactly how we will apply this lemma in our mechanism for vector queries in Section 4.2, following Equation (4).

Algorithm 4 implements the $\mathcal{M}_{\text{RDP-SS}}$ mechanism. Here, the equation $h(t) = 0$ is solved using Brent's method [4] (Line 1). Under the assumption for B, for any $\mathbf{I}' \in \mathcal{I}$,

$$\mathrm{LS}(\mathbf{I}') \leq B(\mathbf{I}') \leq B(\mathbf{I}) + \mathrm{dist}(\mathbf{I}, \mathbf{I}')\,.$$

Therefore, $G(\mathbf{I}) := \max_{k \in \mathbb{N}_{\geq 0}} (k + B(\mathbf{I})) \cdot e^{-\gamma k}$ is the γ-smooth sensitivity of F by Lemma 2.10. In addition, to compute $G(I)$, we only need to evaluate two values of k, as follows. Consider $g(k) = (k + b) \cdot e^{-\gamma k}$, we have its derivative $g'(k) = (1 - \gamma k - \gamma b) e^{-\gamma k}$, thus the maximum is taken

Algorithm 4 RDP Smooth Sensitivity Mechanism $\mathcal{M}_{\text{RDP-SS}}$

Input: Instance $\mathbf{I}$, vector query $F : \mathcal{I} \to \mathbb{R}^d$, upper bound B on $\text{LS}_{L_2}(F)$ with $|B(\mathbf{I}) - B(\mathbf{I}')| \leq 1$, privacy parameters α, ρ

Output: Private estimate $\tilde{F}(\mathbf{I})$

1: $t_1, t_2 \leftarrow \text{brentq}(t^\alpha - \alpha e^{(\alpha-1)\rho/d} \cdot t + (\alpha - 1)e^{(\alpha-1)\rho/d} = 0)$
2: $\gamma \leftarrow \frac{1}{2} \ln \min\{t_1^{-1}, t_2\}$
3: $k \leftarrow \lfloor \frac{1}{\gamma} - B(\mathbf{I}) \rfloor$, $G(\mathbf{I}) \leftarrow (k + B(\mathbf{I})) \cdot e^{-\gamma k}$
4: $k \leftarrow k + 1$, $G(\mathbf{I}) \leftarrow \max\{G(\mathbf{I}), (k + B(\mathbf{I})) \cdot e^{-\gamma k}\}$
5: $\sigma \leftarrow \frac{G(\mathbf{I})}{1-\alpha+\alpha e^{-2\gamma}} \cdot \sqrt{\frac{\alpha}{\rho}}$
6: **return** $F(\mathbf{I}) + \sigma \cdot \mathcal{N}(0, \mathbf{1}_d)$

at the closest integers to $k^* = \frac{1}{\gamma} - b$. Therefore, each $G(\mathbf{I})$ can be obtained by computing two values at most (Lines 4 and 6) and is bounded by

$$G(\mathbf{I}) \leq \frac{1}{\gamma} \cdot e^{\gamma \cdot B(\mathbf{I}) - 1}.$$

We compare RDP-SS with the smooth sensitivity mechanism in Appendix E of the full version [30].

4.2 DP-S4S for Vector Queries

Equipped with the smooth sensitivity mechanism $\mathcal{M}_{\text{RDP-SS}}$ in Lemma 4.1, we are now ready to construct our DP-S4S mechanism for answering vector SJA queries under user-level DP, presented in Algorithm 5. Similar to Algorithm 2 for scalar queries, Algorithm 5 consists of three stages, for computing the sample set S (Lines 2-7), choosing the truncation threshold τ (Lines 9-16), and releasing the result of the truncated query (Lines 18-22), respectively.

Specifically, in the first stage, for each scalar query $f \in F$, DP-S4S samples every join tuple with probability q, obtaining the sample set S_f for f. The combined sample set S is then the union of the samples S_f for all f (Line 7). The second stage then follows the PMSJA framework, which applies the SVT mechanism to choose τ, solving QCQPs on the sample set S in each iteration with a τ that is a power of 2, as mentioned at the beginning of Section 4.

It remains to clarify Stage 3, which receives from Stage 2 the chosen truncation threshold τ, as well as the truncated vector query $\bar{F}$ (defined in Eq. (3)). The goal is to release $\bar{F}$ under Rényi DP so that the overall mechanism attains (ε, δ)-DP via Lemma 2.7. Stage 3 first selects the Rényi order α (discussed below), and then the divergence parameter ρ such that the resulting (α, ρ)-RDP guarantee can be converted to (ε, δ)-DP. It then defines the L_2 local-sensitivity bound B for function $\bar{F}/(2\tau)$ according to Inequality (4), and invokes mechanism $\mathcal{M}_{\text{RDP-SS}}$ (Algorithm 4) to compute a noisy version of $\hat{F}(S)$ under DP, before reporting the scaled version of $\hat{F}(S)$ as the final result (Line 21).

Finally, we clarify the choice of the order α in Rényi divergence (Line 17). While our mechanism is private for any $\alpha > 1 + \frac{\ln(1/\delta)}{\varepsilon_2}$, the error depends on the choice of α. To choose an appropriate α, we perform a doubling search: from the lower bound of α (which leads to an infinite error), we repeatedly double it until we find some α that the data-independent error term $\sqrt{\alpha/\rho}/(1-\alpha+\alpha e^{-2\gamma})$ is smaller compared to choosing $\alpha/2$ or 2α. Then we iterate through possible choices within the interval $(\alpha/2, 2\alpha)$ to locate an α that minimizes the term, and set $\rho = \varepsilon_2 - \frac{\ln(1/\delta)}{\alpha-1}$ accordingly.

The following lemma formalizes the privacy guarantee of Algorithm 5, whose proof appears in Appendix B.4 of the full version [30]. For this lemma, amplification only accounts for the probability that no record is sampled from the witness user. Note that since $q' \leq 1$, we have $\varepsilon' \leq \frac{\varepsilon}{5} + \frac{4\varepsilon}{5} = \varepsilon$.

Algorithm 5 DP-S4S for vector SJA and approximate DP

Input: Join tuples $T_1, \ldots, T_d$, queries $F = \{f_1, \ldots, f_d\}$, privacy parameters ε, δ, sample rate q, probability β

Output: Private query results $\hat{F}(\mathbf{I})$

// *Stage 1: compute the sample set S*
$S \leftarrow \emptyset$
for each $f \in F$ **do**
  $S_f \leftarrow \emptyset$
  for each $t \in T_f$ **do**
    Sample t into S_f with probability q
  $S \leftarrow S \cup \{S_f\}$
// *Stage 2: choose truncation threshold τ with SVT [10, 13]*
$\varepsilon_1 \leftarrow \varepsilon/5$, $\varepsilon_2 \leftarrow 4\varepsilon/5$, $\theta \leftarrow \frac{6}{\varepsilon_1} \ln \frac{6}{\beta} + \mathrm{Lap}(\frac{2}{\varepsilon_1})$
for each $\tau \leftarrow 1, 2, 4, 8, \ldots$ **do**
  $y_u^*, z_{f,t}^* \leftarrow$ Optimal solution to problem (2) on S, τ
  $E(S) \leftarrow \sum_u (1 - y_u^*)$
  if $E(S) + \mathrm{Lap}(\frac{4}{\varepsilon_1}) \le \theta$ **then**
    $\tilde{F}(S) \leftarrow \{\tilde{f}(S) := \sum_{t \in S_f} w_{f,t} \cdot z_{f,t} \mid f \in F\}$
    break
// *Stage 3: release query result via Algorithm 4*
$\alpha \leftarrow \mathrm{find\_alpha}(\varepsilon_2, \delta, d)$
$\rho \leftarrow \varepsilon_2 - \frac{\ln(1/\delta)}{\alpha - 1}$
Define LS upper bound $B(S) := E(S) + 1$
$\tilde{F}(S) \leftarrow 2\tau \cdot \mathcal{M}_{\text{RDP-SS}}(S, \tilde{F}, B, \alpha, \rho)$
Report $\hat{F}(\mathbf{I}) := \frac{1}{q} \cdot \tilde{F}(S)$

Lemma 4.2. *For any chosen α, algorithm 5 satisfies (ε', δ)-DP for*

$$\varepsilon' = \ln(1 + q'(e^{\varepsilon/5} - 1)) + \frac{1}{\alpha - 1}\left(\ln\left(1 + q'\left(e^{4(\alpha-1)\varepsilon/5} \cdot \delta - 1\right)\right) + \ln\frac{1}{\delta}\right),$$

where $q' = 1 - (1 - q)^{\Delta}$.

Discussion. Unlike the case for scalar queries, where we derive a strong result for sampling amplification in Lemma 3.1, here the amplification effect is limited for a large Δ, i.e., when each user may contribute to many join tuples. The reason is that it is unclear how the optimal solution to Program (2) changes even when only 1 tuple is sampled from the witness user. Note that this is the first mechanism that achieves user-level RDP with sampling of join tuples. In particular, it remains unclear whether the original PMSJA mechanism under (ε, δ)-DP achieves the same privacy guarantee when run on sample join tuples. We further evaluate the error of Algorithm 5 in Appendix F of the full version [30].

5 Experiments

This section presents extensive experiment evaluations comparing DP-S4S with the state-of-the-art algorithms on real benchmark datasets, for both relational queries and graphlet counting ones.

5.1 Setup

Mechanisms. We consider the following mechanisms: DP-S4S, R2T [9], PMSJA [10], and S&E [15]. As mentioned in Section 3.2, we do not consider the variant of S&E that requires a pre-known good truncation threshold τ (used in the experiments in [15]), which might leak private information and, thus, would be unfair to other mechanisms. However, we also observe that if we simply allocate a privacy budget in each of the k independent iterations of S&E for truncation threshold computation (e.g., by invoking R2T), its overall noise becomes overwhelming that S&E's error reaches almost 100% error for scalar queries across multiple datasets. To make S&E competitive, we improve the mechanism so that it samples multiple users at the same time, which avoids allocating a privacy budget in each iteration, leading to empirical performance compared to the implementation based on their original description. This is formally described in Corollary D.2 in Appendix D [30]. In addition, we adopt an assumption favorable for S&E that the number of users m is public.

Since all the mechanisms work on precomputed join results, we separate the time for computing the join using PostgreSQL from the time consumed by the DP mechanisms. We implement all algorithms in a single-threaded setting to enable a fair evaluation of CPU time. Each mechanism is run 100 times independently, and we report the average error and running time, dropping the 20 largest and 20 smallest ones.

Datasets. We use the same datasets as existing work [9, 10, 15] for evaluation. For graph pattern counting queries, we use 4 real datasets: *Deezer* is a friendship network. *Amazon* is a co-purchasing graph. Answer-to-question (*A2Q*) and comment-to-answer (*C2A*) are the interactions between users in the Stack Overflow network[3], with edges labeled by date. All of them can be obtained from SNAP [24]. Their statistics are summarized in Table 4 in [30].

For relational database SQL queries, we use the TPC-H benchmark dataset. There are 8 relations: `Region(RK)`, `Nation(NK,RK)`, `Customer(CK,NK)`, `Orders(OK,CK)`, `Supplier(SK,NK)`, `Part(PK)`, `PartSupp(PSID,PK,SK)`, `Lineitem(LID,OK,PSID)`. By default, we use data scale factor 4, where the largest table (`Lineitem`) contains roughly 24 million records.

For the definition of user-level DP, we regard each vertex as a user for the graph datasets. In TPC-H, depending on the query, we may treat each customer (`CK`), supplier (`SK`), part-supp (`PID`), or order (`OK`) as a user, where a join result is contributed by a user if it is related to these primary keys through foreign-key references.

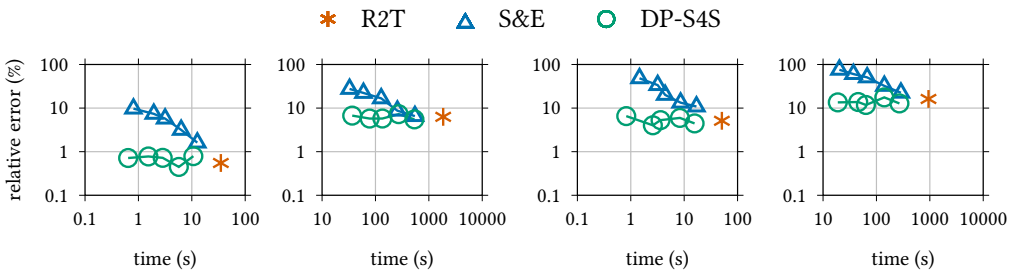
Queries. For graph datasets, we consider common graphlet counting queries, including edges Q_{1-}, line-2 paths Q_{2-}, line-3 paths Q_{3-}, triangles $Q_{\triangle}$ and rectangles $Q_{\square}$. For directed graphs, we also consider the fanout-2 pattern $Q_{<}$, which differs from Q_{2-}. For all these queries, an aggregation unit is contributed by $l \geq 2$ users.

We evaluate the vector query mechanisms on the TPC-H benchmark using the same queries as in [10]. All the queries are derived from the original $Q5$ or $Q7$ in TPC-H, with different aggregation functions {`count(1)`, `sum(l_extendedprice*(1-l_discount))`}, different group by attributes {`n_name`, `l_shipdate`, `extract(year from l_shipdate)`}, and different definitions of private users {(`SK`,`CK`), (`PSID`,`CK`), (`PSID`,`OK`)}.

Parameters and environment. Unless otherwise specified, we consider pure DP with $\varepsilon = 1$ for a scalar query, and approximate DP with $\varepsilon = 4$ and $\delta = 10^{-7}$ for vector queries. The default failure probability β is set to 0.1. For S&E, the number of sampled users is chosen as $k = \lfloor q \cdot m \rfloor$, so that the expected number of records is the same as our mechanism.

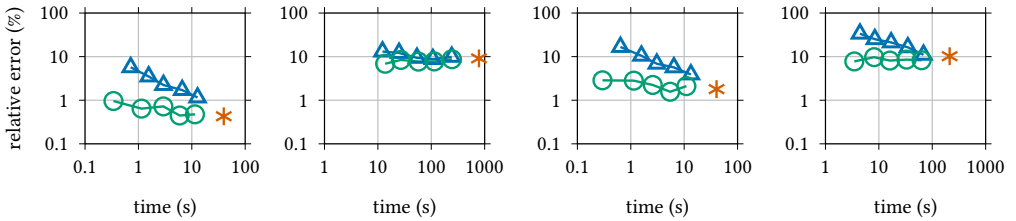
Following prior works, we use data-independent degree upper bounds D for graph datasets as shown in Table 4 in [30]. The global sensitivity is computed accordingly. For example, $\mathrm{GS}_{1-} = D$,

[3]We perform a similar preprocessing as in [10].



(a) Q_{1-} (base query time: 0.18s). (b) Q_{2-} (base query time: 57.4s). (c) $Q_{\triangle}$ (base query time: 8.87s). (d) $Q_{\square}$ (base query time: 296s).

Fig. 2. Query error vs. time on Deezer with $\varepsilon = 1$. (Sampling rates $q \in \{\frac{1}{64}, \frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}\}$)



(a) Q_{1-} (base query time: 0.21s). (b) Q_{2-} (base query time: 35.1s). (c) $Q_{\triangle}$ (base query time: 7.27s). (d) $Q_{\square}$ (base query time: 36.2s).

Fig. 3. Query error vs. time on Amazon with $\varepsilon = 1$. (Sampling rates $q \in \{\frac{1}{64}, \frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}\}$)

$\mathrm{GS}_{2-} = \mathrm{GS}_{\triangle} = D^2$, $\mathrm{GS}_{\square} = D^3$, etc. The upper bound on the number of collaborators C for S&E can also be computed from D. To be more precise, for edge counting and triangle counting, we have $C = D$ since two collaborators must share an edge. For line-2 paths and rectangles, $C = D + D^2$ since a collaborator is either a neighbor or a distance-2 neighbor. In the experiments, we always use $C = D = 1024$ for S&E for simplicity, which is favorable to S&E.

For TPC-H, none of the mechanisms requires the global sensitivity GS as input. S&E still requires setting C. Following their paper [15], we still set $C = 1024$ (again in its favor), though this can be insufficient, especially for vector queries.

All experiments were conducted on a Linux server equipped with an Intel(R) Xeon(R) Gold 6252N 2.30GHz CPU with 96 cores and 1.5TB of memory. Each experiment was run in a separate Docker container. For scalar queries, we allocated 32GB of memory; for vector queries, we allocated 256GB of memory. The experiment code can be found at https://github.com/QiuTedYuan/DP-S4S/.

5.2 Comparison on Scalar Queries

Figures 2 and 3 compare the empirical performance of R2T, S&E, the proposed DP-S4S over graphlet counting queries, with varying sample rate from 1/64 to 1/4 for the latter two. Horizontal axes correspond to execution times, while vertical axes correspond to error (in percentage) relative to the actual query result, both in log scale. Each data point in the plot corresponds to the result for a given sample rate; a high sample rate leads to longer execution time. In the figure captions, *base query time* refers to the join computation time measured by PostgreSQL.

Comparing DP-S4S with R2T, we observe that DP-S4S achieves a similar error on almost every dataset at moderate sample rates. As analyzed in Section 3.3, there are two sources of error in DP-S4S: sampling error and DP noise. While it is typical for sampling error to increase when lowering the sample rate, the amplification through Lemma 1 also becomes more effective, leading to a larger ε_i^* and, correspondingly, lower noise to satisfy DP. The overall error depends on which

is dominating. Recall that a lower sample rate also reduces the maximum contribution of a user to the sample, which offsets the increase in the scaling-back factor. Therefore, despite having an additional sampling error, it is still possible for DP-S4S to be slightly more accurate than R2T under certain sample rates thanks to the privacy amplification. The optimal sample rate, however, depends on both the data size (which affects the sampling error) and the downward sensitivity of the original query (which impacts the DP error). Under differential privacy, we cannot use the exact best value for q that balances both. On the other hand, the experimental results show that the accuracy of DP-S4S is stable across various datasets and a range of sample rates, suggesting that q can be chosen in a data-inpdenent manner. Notably, on the largest set of join tuples (Deezer Q_{2-}), DP-S4S reduces the running time of R2T from 1855 seconds (or 30 minutes) to 36.7 seconds at sample rate $q = 1/64$, with only a mild error increase from 6.30% to 6.72%.

Compared with S&E, DP-S4S is consistently more accurate given the same execution time. Their accuracy gap is larger for small q, and is more than 10x for $q = 1/64$. This is because DP-S4S benefits from having a better sample amplification when q is small, whereas S&E has no amplification effect unless $q \leq 1/(C+1) \approx 10^{-3}$ as per Corollary D.2 in [30]. Also, as shown in Section 3.3, the error of S&E grows with the sample rate (and, thus, the number of iterations k) under pure DP. Even when relaxed to approximate DP, the overhead due to DP noise, $C\sqrt{\log(1/\delta)}$, causes its error to be more than 6000x larger than in the non-sampling setting.

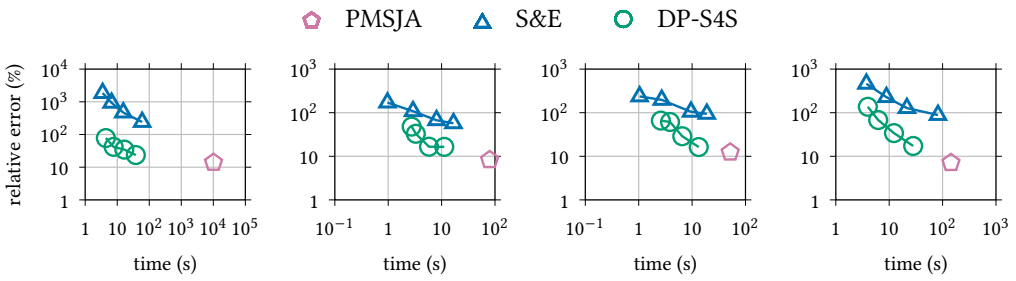
We also observe that S&E has a slightly higher running time than our mechanism when given the same sample rate. This is mainly because of the variance in its sample size. While both mechanisms have the same expected sample size, DS-S4S uses Poisson sampling, whose size is stable. For S&E, the sampled tuples are correlated since all the records from the sampled users are taken. When high-degree nodes are sampled, more join tuples are produced, leading to a higher running time.

5.3 Comparison on Vector Queries

Relational SJA queries. We adopt the same query set as [10]: $Q5$ aggregates the revenue grouped by nation names, where each customer or supplier is a private user. In the three variants of $Q7$, both $7a$ and $7b$ compute group record counts where each customer or partsupp is a private user: $7a$ groups the join results by nation names, while $7b$ uses the year of the shipdate as an additional group-by attribute. $7c$ computes the revenue grouped by shipdate, where each user is a partsupp or an order.

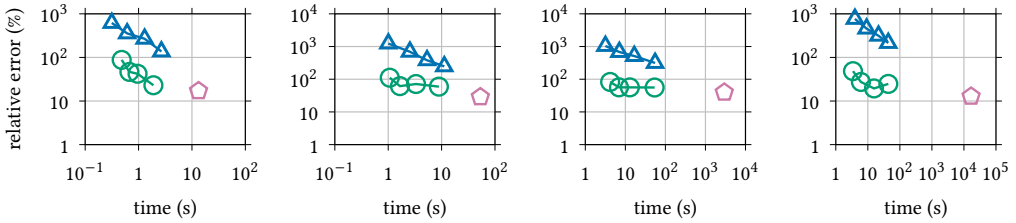
Figure 4 presents the results on a TPC-H benchmark. Clearly PMSJA consumes a lot of memory and CPU time on such datasets. In particular, PMSJA took more than $10,000$ seconds (about 2.7 hours) to produce an output for $Q5$. We have also experimented with $Q7$ by considering customers and suppliers as private users, and PMSJA ran out of memory even after we increased its RAM limit to 512GB. Note that the CPU time and memory consumption of a QCQP solver are super-linear: a typical solver such as MOSEK [1] requires over $O(n^3)$ time and $O(n^2)$ space. Accordingly, doubling the data size may cause the mechanism to be slower by more than 8x, and may increase the memory usage by 4x.

Following the same logic, sampling provides a significant efficiency boost for large datasets. For $Q5$, even with a relatively high sample rate of $1/4$, we can reduce the running time to $38.11s$, which speeds up PMSJA by more than 260x. The error, on the other hand, increases from 13.9% to 23.5%. This is mainly due to the fact that the amplification effect in Lemma 4.2 is small, meaning that the theoretical result of DP-S4S only guarantees the sampled instance is at least as private with respect to the original instance, but it does not allow using a significantly larger ρ (in Rényi DP) or ε (in (ε, δ)-DP) when running the mechanism on sub-sampled data, and reducing the sample rate will increase the scale-up factor $1/q$. It is an interesting future direction to investigate better



(a) Q_5 (base query time: 14.2s). (b) Q_{7a} (base query time: 6.46s). (c) Q_{7b} (base query time: 14.2s). (d) Q_{7c} (base query time: 10.3s).

Fig. 4. Query error vs. time on TPC-H SF4 with $\varepsilon = 4$, $\delta = 10^{-7}$. (Sampling rates $q \in \left\{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}\right\}$)



(a) Q_{1-} (base query time: 1.46s). (b) Q_{2-} (base query time: 1.65s). (c) Q_{3-} (base query time: 1.91s). (d) $Q_{<}$ (base query time: 1.28s).

Fig. 5. Query error vs. time on C2A with $\varepsilon = 4$, $\delta = 10^{-7}$. (Sampling rates $q \in \left\{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}\right\}$)

amplification bounds for combining sampling with the PMSJA mechanism. Nevertheless, DP-S4S helps reduce the noise level to achieve consistently and significantly improved efficiency-utility trade-offs compared to S&E, whose error exceeds 100% even at a sample rate of 1/4 on $Q5$.

Graph queries. In addition to the queries in [10], we include another query that counts the fanout-2 pattern $a \to b \wedge a \to c$, which is different from a length-2 path on a directed graph. The results are presented in Figure 5. Similar to the performance on TPC-H, we observe that DP-S4S is consistently more accurate than S&E by an order of magnitude. The actual improvement depends on both the sample rate and the characteristics of the data.

For the fanout-2 query on C2A, we have to allocate PMSJA with 512GB memory since under the default setup, the QCQP solver MOSEK runs out of memory. In addition to high memory consumption, PMSJA is also slow: it took more than 4 hours to finish a single iteration. Our sampling mechanism reduces the running time to within a minute at a sample rate around 10%, without severely affecting the error, whereas S&E offers the same level of efficiency improvement at the cost of rather low result utility.

5.4 Parameter Study

Varying privacy parameter ε. Figure 6 presents the effect of varying the privacy parameter ε on all the mechanisms over scalar queries. Observe that even under a relatively small sample rate of 1%, DP-S4S still matches the error of R2T, whereas S&E has an overhead that can reach 14x. In all the scenarios other than DP-S4S on Q_{1-}, DP noise dominates sampling error; thus, the accuracy of all mechanisms improves with increasing ε. DP-S4S clearly achieves a better privacy-utility trade-off compared to S&E in all settings. The privacy amplification effect may also help DP-S4S to outperform R2T when DP noise dominates (as for Q_{2-} and $Q_{\square}$), in addition to the benefit of reducing the query time by over 100x.

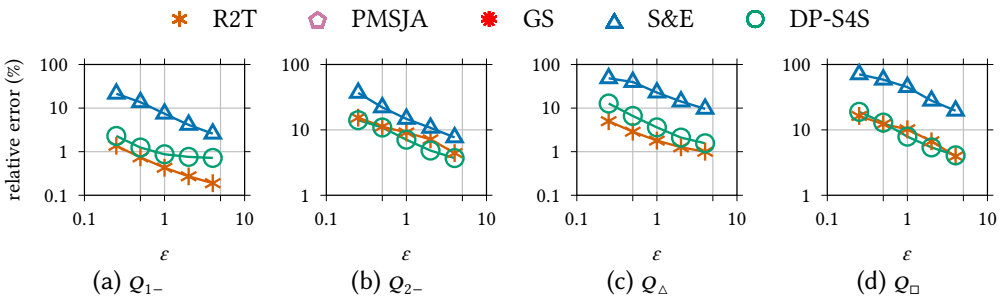


Fig. 6. Query error vs. ε on Amazon with $q = 1\%$ for sampling mechanisms.

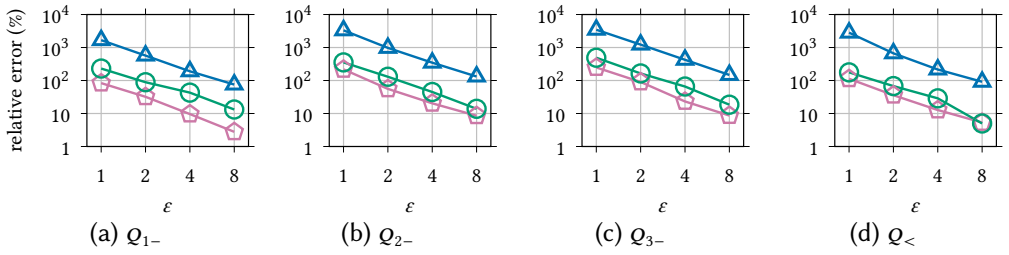


Fig. 7. Query error vs. ε on C2A with $\delta = 10^{-5}$ and $q = 10\%$ for sampling mechanisms.

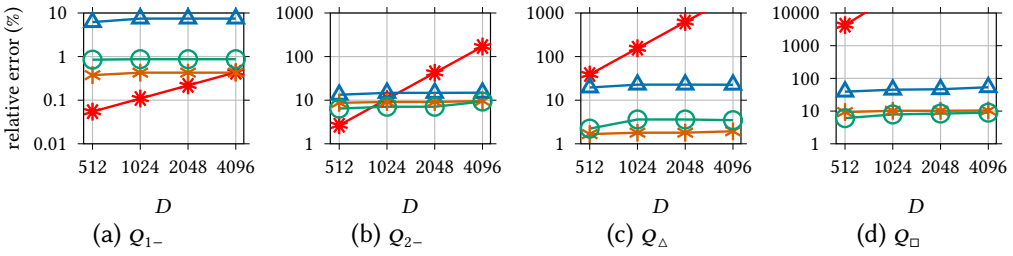


Fig. 8. Query error vs. D on Amazon with $\varepsilon = 1$ and $q = 1\%$ for sampling mechanisms.

Figure 7 shows the results for scalar queries. Since the amplification from Lemma 4.2 is limited, we run the sampling mechanisms with $q = 10\%$ to avoid large scale-back factors. We observe a 2x-3x error increase for DP-S4S, compared to a more than 15x increase in error for S&E. Nevertheless, since QCQPs are far more expensive to solve than LPs, even with a sample rate of 10%, we achieve 2-3 orders of magnitude improvement in running time, according to Figure 5. This leads to a favorable privacy-accuracy trade-off, and DP-S4S consistently outperforms S&E.

Degree upper bound D (equivalently C and Δ). All the mechanisms considered above achieve instance-specific error, which is practical for queries with a large global sensitivity. For graph queries, there are two reasons that account for a large GS: on the one hand, the degree upper bound D should be a data-independent worst-case upper bound. Under-estimating D can lead to DP violations. In practice, this can be obtained from domain knowledge, e.g., Facebook limits each user to have up to 5,000 friends. On the other hand, for queries involving joins, we need to adopt worst-case join size bounds [2], whereas in practice the join size can be much smaller. The same logic applies to the number of collaborators bound C in S&E, and the number of tuples bound Δ in our mechanism. This has been studied in existing work [9, 10].

Figure 8 shows the results with varying D and correspondingly C and Δ. For all queries, we vary the degree bound D from 512 to 4096. Note that the maximum degree in the dataset is $549 > 512$; therefore, setting $D = 512$ actually violates the privacy constraint. As expected, GS mechanism scales linearly with Δ, which is D for Q_{1-} and $\Omega(D^2)$ for the other queries. Instance-specific mechanisms like R2T is better when GS is large, as the dependency is only $O(\log D)$. For sampling mechanisms, besides the overhead introduced by R2T, a larger D also leads to a larger C for S&E and a larger Δ for DP-S4S, weakening the privacy amplification effects in both. Accordingly, the result utility of DP-S4S is slightly worse than R2T, and consistently better than S&E.

For vector queries, both PMSJA and DP-S4S show performance independent of the choice of D. Note that D (or Δ) is not an input to Algorithm 5, as do not apply privacy amplification in this case.

6 Conclusion

We have designed and implemented a novel sampling-based mechanism DP-S4S for efficiently and accurately answering both scalar and vector SJA queries under user-level differential privacy. The main ideas are to (i) sample join tuples and (ii) employ Rényi DP as an intermediate step to reach the required (ε, δ)-DP. For scalar queries, we have derived a strong sampling amplification result, leading to DP-S4S matching (and sometimes even exceeding) the accuracy of the non-sampling solution R2T, at a fraction of the latter's cost. For vector queries, we have additionally built the necessary infrastructure: the first RDP smooth sensitivity mechanism, which can be of independent interest in tasks such as releasing graph statistics or computing gradient vectors in machine learning. Extensive experiments confirm the consistent and significant (often 10x) accuracy improvement over the previous state of the art S&E.

For future work, we plan to investigate whether the amplification from sampling aggregation units extends to arbitrary (ε, δ)-DP mechanisms beyond Rényi DP. In addition, enforcing DP on streaming data [6–8, 12] via dynamic sampling is another exciting future topic.

Acknowledgments

Yuan Qiu is supported by the Start-up Research Fund of Southeast University, under grant RF1028625150. Xiaokui Xiao is supported by the Ministry of Education, Singapore, under an AcRF Tier-2 Grant (MOE-000761-01), and by Google under grant 00006186. Yin Yang is supported by the Qatar National Research Fund, a member of the Qatar Foundation (No. PPM 05-0506-210017). Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the authors and do not reflect the views of the funding agencies. We thank the anonymous reviewers for their constructive comments. We also thank Yangfan Jiang for discussions on privacy amplification, and Yiwei Chen for discussions on smooth sensitivity.

References

[1] MOSEK ApS. 2025. *The MOSEK Python Fusion API manual. Version 11.0.* https://docs.mosek.com/latest/pythonfusion/index.html

[2] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size Bounds and Query Plans for Relational Joins. In *FOCS*. IEEE Computer Society, 739–748.

[3] Borja Balle, Gilles Barthe, and Marco Gaboardi. 2018. Privacy Amplification by Subsampling: Tight Analyses via Couplings and Divergences. In *NeurIPS*. 6280–6290.

[4] Richard P Brent. 2013. *Algorithms for minimization without derivatives.* Courier Corporation.

[5] Mark Bun and Thomas Steinke. 2016. Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds. In *TCC (B1) (Lecture Notes in Computer Science, Vol. 9985)*. 635–658.

[6] Xuyang Cao, Yang Cao, Primal Pappachan, Atsuyoshi Nakamura, and Masatoshi Yoshikawa. 2023. Differentially Private Streaming Data Release Under Temporal Correlations via Post-processing. In *DBSec (Lecture Notes in Computer Science, Vol. 13942)*. Springer, 184–200.

[7] T.-H. Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. 2012. Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams. In *Privacy Enhancing Technologies (Lecture Notes in Computer Science, Vol. 7384)*. Springer, 140–159.
[8] Wei Dong, Zijun Chen, Qiyao Luo, Elaine Shi, and Ke Yi. 2024. Continual Observation of Joins under Differential Privacy. *Proc. ACM Manag. Data* 2, 3 (2024), 128.
[9] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *SIGMOD Conference*. ACM, 759–772.
[10] Wei Dong, Dajun Sun, and Ke Yi. 2023. Better than Composition: How to Answer Multiple Relational Queries under Differential Privacy. *Proc. ACM Manag. Data* 1, 2 (2023), 123:1–123:26.
[11] Cynthia Dwork. 2006. Differential Privacy. In *ICALP (2) (Lecture Notes in Computer Science, Vol. 4052)*. Springer, 1–12.
[12] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *STOC*. ACM, 715–724.
[13] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.
[14] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. 2010. Boosting and Differential Privacy. In *FOCS*. IEEE Computer Society, 51–60.
[15] Juanru Fang and Ke Yi. 2024. Privacy Amplification by Sampling under User-level Differential Privacy. *Proc. ACM Manag. Data* 2, 1 (2024), 34:1–34:26.
[16] Chang Ge, Xi He, Ihab F. Ilyas, and Ashwin Machanavajjhala. 2019. APEx: Accuracy-Aware Differentially Private Data Exploration. In *SIGMOD Conference*. ACM, 177–194.
[17] Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, Raghu Meka, and Chiyuan Zhang. 2023. User-Level Differential Privacy With Few Examples Per User. In *NeurIPS*, Vol. 36. 19263–19290.
[18] Manuel Gil, Fady Alajaji, and Tamás Linder. 2013. Rényi divergence measures for commonly used univariate continuous distributions. *Inf. Sci.* 249 (2013), 124–131.
[19] Hong Guan, Lei Yu, Lixi Zhou, Li Xiong, Kanchan Chowdhury, Lulu Xie, Xusheng Xiao, and Jia Zou. 2025. Privacy and Accuracy-Aware AI/ML Model Deduplication. *Proc. ACM Manag. Data* 3, 3 (2025), 203:1–203:28.
[20] Yangfan Jiang, Xinjian Luo, Yin Yang, and Xiaokui Xiao. 2024. Calibrating Noise for Group Privacy in Subsampled Mechanisms. *Proc. VLDB Endow.* 18, 2 (2024), 322–334.
[21] Vishesh Karwa, Sofya Raskhodnikova, Adam D. Smith, and Grigory Yaroslavtsev. 2014. Private Analysis of Graph Structure. *ACM Trans. Database Syst.* 39, 3 (2014), 22:1–22:33.
[22] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. 2013. Analyzing Graphs with Node Differential Privacy. In *TCC (Lecture Notes in Computer Science, Vol. 7785)*. Springer, 457–476.
[23] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: A Differentially Private SQL Query Engine. *Proc. VLDB Endow.* 12, 11 (2019), 1371–1384.
[24] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.
[25] Daniel Levy, Ziteng Sun, Kareem Amin, Satyen Kale, Alex Kulesza, Mehryar Mohri, and Ananda Theertha Suresh. 2021. Learning with User-Level Privacy. In *NeurIPS*. 12466–12479.
[26] Ilya Mironov. 2017. Rényi Differential Privacy. In *CSF*. IEEE Computer Society, 263–275.
[27] Yurii Nesterov and Arkadii Nemirovskii. 1994. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, Philadelphia. https://doi.org/10.1137/1.9781611970791
[28] Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. 2007. Smooth sensitivity and sampling in private data analysis. In *STOC*. ACM, 75–84.
[29] Panos M. Pardalos and Stephen A. Vavasis. 1991. Quadratic Programming with One Negative Eigenvalue is NP-hard. *Journal of Global Optimization* 1, 1 (1991), 15–22. https://doi.org/10.1007/BF00120662
[30] Yuan Qiu, Xiaokui Xiao, and Yin Yang. 2026. *DP-S4S: Accurate and Scalable Select-Join-Aggregate Query Processing with User-Level Differential Privacy.* https://arxiv.org/abs/2603.14994
[31] Yuchao Tao, Amir Gilad, Ashwin Machanavajjhala, and Sudeepa Roy. 2025. Differentially private explanations for aggregate query answers. *VLDB J.* 34, 2 (2025), 20.
[32] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. 2011. The Anatomy of the Facebook Social Graph. *arXiv preprint arXiv:1111.4503* (2011). https://arxiv.org/abs/1111.4503
[33] Salil P. Vadhan. 2017. The Complexity of Differential Privacy. In *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 347–450.
[34] Shufan Zhang and Xi He. 2023. DProvDB: Differentially Private Query Processing with Multi-Analyst Provenance. *Proc. ACM Manag. Data* 1, 4 (2023), 267:1–267:27.

Received October 2025; revised January 2026; accepted February 2026