SLIM-View: Sampling and Private Publishing of Multidimensional Databases[*]

Ala Eddine Laouir and Abdessamad Imine
Université de Lorraine, CNRS and Inria
Nancy, France
firstname.lastname@loria.fr

ABSTRACT

Despite the enormous data processing capacity available in big data frameworks, obtaining appropriate and private responses to large-scale queries without revealing sensitive information is still a challenging problem. In this paper, we address the problem of combining offline sampling techniques for space efficiency in multidimensional databases and Differential Privacy (DP) to protect sensitive data. We present our framework SLIM-View, which uses a novel sampling technique relying on a bi-objective optimization to decide the best sample size and the exponential mechanism to select the best sample while ensuring privacy. Our extensive experiments demonstrate that SLIM-View outperforms existing approaches by orders of magnitude in terms of utility and scalability while ensuring the same level of privacy.

CCS CONCEPTS

• **Security and privacy → Database and storage security**; **Data anonymization and sanitization**.

KEYWORDS

Multidimensional data, Privacy, Differential Privacy, Data sampling.

ACM Reference Format:
Ala Eddine Laouir and Abdessamad Imine. 2024. SLIM-View: Sampling and Private Publishing of Multidimensional Databases. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY '24), June 19–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3626232.3653275

1 INTRODUCTION

Huge volumes of data from individuals (e.g. users or clients) constitute a primary source of information that could help companies improve and grow their businesses. OLAP (Online Analytical Processing) is one of the most widely used data analysis methods to extract this valuable information. Through the use of *aggregation queries* (such as SUM and COUNT), the data analyst is able to leverage quantities of data to create insights that aid the decision-making process. To improve the performance of these queries, the data is preprocessed and organized as multidimensional data (in tensor or tabular form) that contains precomputed aggregations on certain data attributes (see Figure 1). Despite these pre-established aggregations, the enormous volume of data can require significant storage space and slow down query response. Moreover, because the data collected is personal to each individual, a malicious data analyst can issue queries to obtain sensitive information about the individuals. The fact that the data retrieved is the result of aggregation queries does not offer any privacy guarantee.

Due to the risks on the privacy of individuals, many privacy models and techniques have been proposed in the literature. Differential Privacy (DP) [5] is increasingly considered the gold standard for data privacy, both in the research community and in real-world applications and industries (Google [20], Apple [19], US Census Bureau [2]). It formally guarantees that the data query result is not dependent on the presence/absence of a particular individual while retaining the utility. This offers the individuals the deniability needed for their privacy.

In the context of databases, there are two broad categories of solutions providing DP for OLAP tasks. The first is to apply DP to the query result, called *online query answering* under DP and in this scenario the number of queries the analyst can issue is limited by what is called the *privacy budget*. The second category is *publishing data under* DP, where the data itself is perturbed while consuming a fixed privacy budget and the analyst can query the perturbed data without any additional budget consumption. In our work, we focus on publishing data under DP.

A naive approach to producing DP-perturbed data is to add noise to all data points, but this results in poor utility because queries will accumulate a lot of noise [5]. More sophisticated mechanisms and techniques are therefore needed to balance the loss of utility and the privacy noise. In the literature, there have been several works on differentially private techniques to publish multidimensional data like data partitioning approaches [11, 23], where they focus on transforming the original data into smaller blocks and add noise them. Another promising direction is using generative models to produce synthetic differentially private data [3, 9, 10, 18, 21, 22], some focusing on image data and others on regular tabular data. The main weakness of these techniques is their sensitivity to the distribution of values in multidimensional data and therefore they fail compared to other techniques in terms of utility, especially in OLAP testing. Other solutions are workload-dependent algorithms [13–15] which fall into the category of batch answering techniques. So they do not produce a privacy-preserving view of the data but rather respond to a certain number of queries defined in a workload at once. They mainly focus on low-dimensional data due to their complexity.

All these solutions fail to meet all of the desired properties that should be satisfied by any approach addressing the multidimensional data privacy problem, namely: utility preservation, scalability

[*]This work is funded by DigiTrust (http://lue.univ-lorraine.fr/fr/article/digitrust/).

to high-dimensional data, space efficiency to reduce the storage space required to materialize the privacy-preserving view, and publishing data with or without a predefined workload. In comparison with many existing approaches in the literature, our solution SLIM-View (SampLing dIfferentially Materialized View) proposed in this paper is able to fully meet all these requirements.

Our contributions. To our knowledge, SLIM-View is the first technique that considers creating and publishing a differentially private view using sampling for space efficiency. In our proposal, we define a new sampling algorithm that preserves utility while ensuring differential privacy and uses an optimization problem to autonomously find the best sample size for space efficiency. It is worth noting that SLIM-View can work with or without a given workload, and in both cases we show experimentally that it can provide better utility than other solutions with higher scalability.

Paper organisation. The paper is structured as follows: Section 2 reviews some known works in the literature. Section 3 presents the concepts used throughout our paper. Section 4 presents the problem statement for publishing a private view based on data sampling. The detailed overview of SLIM-View is given in Section 5, and Section 6 provides an experimental evaluation of our solution. In Section 7, we discuss the limitations of our solution and we conclude in Section 8 by giving some future works.

2 RELATED WORK

In the case of multidimensional data, the naive application of DP is to add random noise using Laplace mechanism to all cells (called *Identity* solution) [5]. In [4], a heuristic is presented to manage consistency when applying *Identity* to a lattice structure of high-dimensional data. This basic solution results in poor utility, especially when exploring data using OLAP range queries due to accumulated noise added to each cell. To provide a better alternative, the issue of publishing private views of high-dimensional data has been frequently addressed in the literature. In recent years, many methods have been developed to explore and query high-dimensional data under DP. We present here the state-of-the-art relating to this problem.

Partitioning-based approaches. Instead of applying noise to all cells, [24] and [13] provide a clustering algorithm that groups cells into bins, approximates the values in each bin by the average of those cells, and uses the Laplace mechanism to add noise to the average value of each bin. These solutions are effective only in low-dimensional data (1 or 2 dimensions for small datasets). In [17, 23], the authors use multidimensional spatial data and propose a recursive partitioning algorithm which uses the quad-tree strategy to divide the tensor into smaller blocks. Considering the original tensor as the initial block, the algorithm will divide it into four blocks and proceed recursively to each new block created until a certain depth is reached or the number of cells in each block is sufficiently small. The cells in each block will be approximated by the mean of the cells in that block, and random noise is added to the mean of each block to preserve privacy. HDPView [11] can be seen as an improvement of [23], where a new heuristic was proposed to produce fewer blocks while maintaining better utility for OLAP queries.

Workload-based approaches. Using a workload (a set of queries) to determine which parts of the data to publish is a well-founded approach, and it represents a mutually defined contract between the data provider and end users. In [14], the authors introduced the matrix mechanism that optimizes the results of a workload. The work in [15] presents an extension for [14] (HDMM) which adapts the matrix mechanism to high-dimensional data and improves [13] which also falls into this category of solutions. PrivateSQL [12] offers a framework that calculates the sensitivity of a relational data based on its schema and relations . Then it creates a multidimensional view of this data using [13] or [14].

Private generative models. This class of solutions trains a generative model on sensitive data subject to privacy constraints, and the model is used to generate synthetic data that can be used for publication and used to answer user queries. In [22], the authors train a model based on a Bayesian network under DP guarantees. In [16, 21, 25] marginal tables are used to learn the distribution of the original data. Deep generative models are also a promising solution to produce private synthetic data [3, 9, 10, 18]. The authors of [8] used a generative adversarial network (GAN). According to their results, it outperforms [22] only on some classification tasks, but is outperformed by [22] on private tabular data.

The partition-based solution runs either on low-dimensional data [13] or on a particular data structure such as blocks [11, 23], which requires a specific system to fulfill the query. In addition to their high complexity, workload-based approaches use a matrix structure for queries and data, which also requires a dedicated solution to query and explore the results. For generative models, a huge volume of data is required to obtain accurate results using GANs and Bayesian network models are very sensitive to data distribution. In addition, these approaches are very complex and resource intensive for big data. In scenarios where data is frequently updated, this cost becomes very significant.

For all these reasons, the adoption of these solutions in real applications will be very difficult and will require expertise and resources. To address this limitation, we propose a new approach that is simple to deploy, resource-efficient, and scalable while ensuring data privacy and utility.

3 PRELIMINARIES

In this section, we give the notation and explain briefly notions used throughout the paper.

Database. Given a database DB with a set of tuples defined over a set of d attributes $A = \{a_1, \dots, a_d\}$, where d is called the *number of dimensions* (or attributes) of DB. Each attribute a is associated with a domain $dom(a)$ containing $|dom(a)|$ discrete and totally ordered values. The overall domain size of DB is $|dom(A)| = \prod_{a \in A} |dom(a)|$. For instance, the domain of attribute *Patient* is $\{Jack, Sophie, Alice, \dots, Bob\}$ as illustrated in *Fact table* of Figure 1. We can represent $dom(a)$ as a *range* $\tau[b_a, e_a]$ such that $b_a \leq v_a \leq e_a$ for every value $v_a \in dom(a)$. For ranges $\tau[b_a, e_a]$ and $\tau'[b'_a, e'_a]$: τ' is a *subrange* of τ when $b_a \leq b'_a$ and $e'_a \leq e_a$; τ and τ' *overlap* if (i) $b'_a \leq b_a \leq e'_a$, or (ii) $b'_a \leq e_a \leq e'_a$.

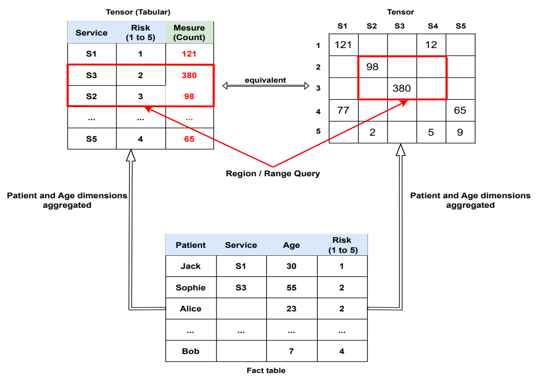
Count tensor. We transform a database DB with d dimensions into a *count tensor* (i.e., *multidimensional data*) $\mathcal{R}$ over ranges $\tau_1^{\mathcal{R}}, \dots, \tau_d^{\mathcal{R}}$

such that $\mathcal{R}[v_1, \ldots, v_d]$ is a *count cell* storing the number of tuples in *DB* that have the same values $v_1, \ldots, v_d$. A database can also be transformed into a count tensor over a subset of dimensions, where non-appearing attributes will simply be aggregated. For simplicity, we use "tensor" instead of "count tensor".

In *Fact table* of Figure 1, the dimensions *Patient* and *Age* have been aggregated to create a tensor, which can be a *multidimensional array* or *tabular* form. The first representation visualizes (and stores) the entire domain, while the other only stores non-empty cells in the domain. Our work is based on the tabular representation, but we take the entire domain into account when constructing a private view. Furthermore, several database management systems, such as Postgres [1], allow tensor manipulation for OLAP tasks.

We call *region* any subtensor R of $\mathcal{R}$ such that the dimensions of R are covered by subranges of those of $\mathcal{R}$. The number of non-empty cells is denoted by $|R|$, and the domain of a region (subtensor) is denoted by $||R||$ which is computed in similar way to the domain of the original *tensor*. The intersection between two regions R_1 and R_2 (i.e. $R_1 \cap R_2$) is the set of cells defined by the overlap between all the ranges of their dimensions. R_1 is a *sample* of R_2, denoted by $R_1 \subset R_2$, if (i) R_1 and R_2 have the same ranges on each dimension, and (ii) the non-empty cells of R_1 are a subset of those of R_2 (i.e. $|R_1| < |R_2|$).

In Figure 1, highlighted in red (in both representations) is the region defined by the following ranges: $R = \{\ Service \in [S2, S3], Risk \in [2, 3]\}$ and $|R| = 2$ is the number of non-empty cells (rows in tabular representation) within R.


Tensor (Tabular)
Tensor
equivalent
Region / Range Query
Patient and Age dimensions aggregated
Patient and Age dimensions aggregated
Fact table

Service	Risk (1 to 5)	Mesure (Count)
S1	1	121
S3	2	380
S2	3	98
...	...	...
S5	4	65

	S1	S2	S3	S4	S5
1	121			12	
2		98			
3			380		
4	77				65
5		2		5	9

Patient	Service	Age	Risk (1 to 5)
Jack	S1	30	1
Sophie	S3	55	2
Alice		23	2
...	...	...	...
Bob		7	4

Figure 1: Example of count tensor.

Count range queries. We consider a special class of queries, namely *count range queries*. Given a tensor $\mathcal{R}$ with d dimensions. A count range query Q is defined on k dimensions ($k \leq d$) with subranges $\tau_1^Q, \ldots, \tau_k^Q$ and $Q(\mathcal{R})$ targets a region of $\mathcal{R}$ to return the sum of its count cells. It is clear that Q targets a region to compute the sum of its cells. In Figure 1, highlighted in red (in both representations) is the query Q defined by the following ranges: $Q = \{\ Service \in [S2, S3]\ , Risk \in [2, 3]\}$ and it returns: $Q(\mathcal{R}) = 478$.

Workload. A *workload* is a set of count range queries used by the data publisher and the end user as a contract that defines the view to be published (via regions targeted by count range queries) instead of the whole data tensor [12]. We consider here that the tensor publication is driven by a workload. Accordingly, we describe our tensor as a set of disjoint (or non-overlapping) regions $\mathcal{R} = \{R_1, \ldots, R_n\}$ (i.e., for all $R_i, R_j \in \mathcal{R}$, $R_i \cap R_j = \emptyset$). For instance, we can consider the following tensor with two regions (see Figure 1): $\mathcal{R} = \{\{Service \in [S1, S2]\ , Risk \in [2, 3]\}_1, \{Service \in [S1, S5]\ , Risk \in [4, 5]\}_2\}$.

Differential Privacy. In this work, we use a method, called *Differential Privacy* (DP), which quantitatively assesses the privacy of data publication [5]. Instead of data, DP is intended for algorithms.

Given two databases DB and DB' with their corresponding tensors $\mathcal{R}$ and $\mathcal{R}'$ respectively. First, we define the concept of *neighbouring* databases as follows:

Definition 3.1 (Neighbouring databases [6]). Two databases DB and DB' are *neighbouring* if we can obtain one of them by inserting a tuple (or a single individual) into the other.

The impact of this neighborhood on $\mathcal{R}$ and $\mathcal{R}'$ constructed respectively from DB and DB' is either (i) a cell whose value is incremented/decremented, or (ii) an empty cell (with value 0) becomes non-empty with value 1 or vice versa.

Let M be a data publishing algorithm that takes as input a tensor $\mathcal{R}$ and publishes information about the data it contains. To consider M as *differentially private*, it must be insensitive to the presence or absence of an individual in $\mathcal{R}$.

Definition 3.2 (ϵ-Differential privacy [6]). An algorithm M satisfies *ϵ-differential privacy* (or *ϵ-DP*) if for all neighbouring tensors $\mathcal{R}$ and $\mathcal{R}'$, and all possible sets of outputs S:

$$\frac{\Pr\left[M(\mathcal{R}) \in S\right]}{\Pr\left[M(\mathcal{R}') \in S\right]} \leq exp(\epsilon)$$

The parameter ϵ is called the *privacy budget*.

In practice, M is a *randomized* algorithm, which has many possible outputs under the same input. It is well known that DP is used to answer specific queries on databases. Let f be a query on a tensor $\mathcal{R}$ whose its answer $f(\mathcal{R})$ returns a number. The *sensitivity* of f is the amount by which the output of f changes for all neighbouring tensors.

Definition 3.3 (Sensitivity [6]). The sensitivity of a query f for any two neighboring tensors $\mathcal{R}, \mathcal{R}'$ is:

$$\Delta_f = \max_{\mathcal{R},\mathcal{R}'} \left\| f(\mathcal{R}) - f(\mathcal{R}') \right\|_1$$

where $||.||_1$ is the L_1 norm.

For instance, if f is a count range query then Δ_f is 1.

To achieve DP for a query f, we need to add random noise to its answer $f(\mathcal{R})$. To inject adequate noise, some basic mechanisms have been developed for DP. One of them is the *Laplace Mechanism*.

Definition 3.4 (Laplace Mechanism [6]). The *Laplace mechanism* adds noise to $f(\mathcal{R})$ as:

$$f(\mathcal{R}) + Lap\left(\frac{\Delta_f}{\epsilon}\right)$$

where Δ_f is the *sensitivity* of f, and $Lap(\alpha)$ denotes sampling from the Laplace distribution with center 0 and scale α.

For instance, if f is a count range query then the noise is sampled from the distribution $Lap(1/\epsilon)$.

Unlike the Laplace Mechanism (based on noisy numerical answers), the *Exponential mechanism* allows for selecting the "best" element (according to a scoring function) in a set while preserving DP [6].

Definition 3.5 (Exponential Mechanism [6]). Given a set of values V extracted from a tensor $\mathcal{R}$ and a real scoring function u to select values in V. The *Exponential Mechanism* randomly samples v from V with probability proportional to:

$$\exp\left(\frac{\epsilon \times u(\mathcal{R}, v)}{2 \times \Delta_u}\right)$$

where Δ_u is the sensitivity of u.

The Exponential Mechanism satisfies DP by approximately maximizing the score of the value it returns, while sometimes returning a value from the set that does not have the highest score.

Combining several DP mechanisms is possible, and the privacy accounting is managed using the sequential and the parallel composition properties of DP. Let $M_1, ..., M_n$ be mechanisms satisfying $\epsilon_1, ..., \epsilon_n$ -DP.

THEOREM 3.6 (SEQUENTIAL COMPOSITION [6]). *An algorithm sequentially applying $M_1, ..., M_n$ satisfies $\left(\sum_{j=1}^{n} \epsilon_j\right)$-DP.*

THEOREM 3.7 (PARALLEL COMPOSITION [6]). *An algorithm applying $M_1, \ldots, M_n$ to n disjoint datasets in parallel satisfies ($max_{j=1}^{n} \epsilon_j$)-DP.*

THEOREM 3.8 (POST-PROCESSING [6]). *For any ϵ-DP mechanism $M(x)$ and any function f, $f(M(x))$ satisfies ϵ-DP.*

Datasets can be transformed to produce new ones (e.g., using relational algebra operators such as selection or projection). It is therefore possible to combine such data transformations with DP mechanisms, and the privacy budget is calculated based on the stability of the transformation [7]. Let T be a data transformation (i.e., a function from tensors to tensors) and M an ϵ-DP mechanism.

THEOREM 3.9 (TRANSFORMATION STABILITY [7]). *Transformation T is said c-stable if for any two neighbouring tensors $\mathcal{R}$ and $\mathcal{R}'$, the size of the symmetric difference between $T(\mathcal{R})$ and $T(\mathcal{R}')$ is bounded by a positive integer c.*

Composing a function f with a c-stable transformation has the sensitivity of $c \times \Delta_f$. Accordingly, the composition also satisfies DP [7].

THEOREM 3.10 (TRANSFORMATION COMPOSITION [7]). *The composite computation $M \circ T$ provides $\epsilon \times c$-differential privacy.*

4 PROBLEM STATEMENT

4.1 Compression and Perturbation

Given a tensor $\mathcal{R} = \{R_1, \ldots, R_n\}$, we aim to find its differentially private view $\tilde{\mathcal{R}} = \{\tilde{R_1}, \ldots, \tilde{R_n}\}$. We define the construction of this view as an optimization problem.

We consider a *compression mapping* ρ to compress a count tensor $\mathcal{R} = \{R_1, \ldots, R_n\}$ into $\mathcal{R}' = \{R'_1, \ldots, R'_n\}$ such that $R'_i \subset R_i$ (or R'_i is a sample of R_i) for each $R_i \in \mathcal{R}$ and $R'_i = \rho(R_i)$. By compression, we seek to reduce the size of each region through the *suppression of cells*. This transformation ensures space efficiency in a big data context. The *compression variation*, with respect to non-empty cells, can be measured for each region R_i as follows:

$$\mathrm{CV}_\rho(R_i) := |R_i| - |\rho(R_i)| \tag{1}$$

We denote the sum over R_i as $S_i = \sum_{c \in R_i} c$ (where c is the count cell) and its perturbed output $\tilde{S_i} = S'_i + \eta_i$ where S'_i is the sum over the compressed region R'_i and η_i is the noise sampled from the Laplace Mechanism $Lap(1/\epsilon)$ and applied over R_i.

How to apply this noise? Two extreme solutions are possible: sample either a noise for each cell (non-empty or empty) or a single noise for the entire region. Likewise *Identity* [5], the first solution is more expensive and destroys the data utility due to the amount of errors generated. As for the last solution, it is not safe because any malicious user can guess the noise value from some known values. In this work, we propose a minimal and safe noise coverage over the regions. To do that, we split each region R_i into m_i subregions $r_{i,1}, \ldots, r_{i,m_i}$ where each subregion $r_{i,l}$ has a sampled noise $\eta_{i,l}$ with $l \in 1, \ldots, m_i$. We denote by $\Delta(R_i)$ the set of subregions associated to R_i, such that its corresponding compressed region R'_i has the same subregions (i.e. $\Delta(R_i) = \Delta(R'_i)$).

This noise coverage requires a data structure to handle the perturbation of the results returned by queries applied on the tensor. In this case, the size of this data structure must not cause the gain obtained by region compression to be lost. Therefore, the maximal number of subregions m_i for each region R_i must satisfy the following condition:

$$m_i \ < \ |R_i| - |\rho(R_i)| \tag{2}$$

To protect the selection of subregions for each region, we choose randomly m_i between 1 and $k = |R_i| - |\rho(R_i)| - 1$.

Compression and perturbation incur two types of errors: *Suppression Error* (SE) and *Perturbation Error* (PE), as defined as follows. Given a compression mapping ρ. For any region $R_i \in \mathcal{R}$ the absolute error AE_ρ between R_i and its perturbed output $\tilde{R_i}$ is:

$$\begin{aligned} \mathrm{AE}_\rho(R_i) := |S_i - \tilde{S_i}| \quad &\leq |S_i - S'_i| + |\eta_i| \\ &\leq \mathrm{SE}_\rho(R_i) + \mathrm{PE}_\rho(R_i) \end{aligned} \tag{3}$$

$$\text{Compression Error is } \mathrm{SE}_\rho(R_i) := |S_i - S'_i| \tag{4}$$

$$\text{Perturbation Error is } \mathrm{PE}_\rho(R_i) := |\eta_i| = |P(m_i) \cdot Lap(1/\epsilon)| \tag{5}$$

Now η_i is based on two independent distribution functions $P(m_i)$ and $Lap(1/\epsilon)$ that sample a number of subregions and a noise from Laplace Mechanism, respectively. As the noise η_i is sampled from two distributions, the expectation of $\mathrm{AE}_\rho(R_i)$ is:

$$\begin{aligned} \mathbb{E}\left[\sum_{R_i \in \mathcal{R}} \mathrm{AE}_\rho(R_i)\right] &\leq \mathbb{E}\left[\sum_{R_i \in \mathcal{R}} \mathrm{SE}_\rho(R_i)\right] + \mathbb{E}\left[\sum_{R_i \in \mathcal{R}} \mathrm{PE}_\rho(R_i)\right] \\ &= \sum_{R_i \in \mathcal{R}} \mathrm{SE}_\rho(R_i) + \sum_{R_i \in \mathcal{R}} \mathbb{E}[\mathrm{PE}_\rho(R_i)] \\ &= \sum_{R_i \in \mathcal{R}} \mathrm{SE}_\rho(R_i) + \frac{1}{2\epsilon} \sum_{R_i \in \mathcal{R}} |R_i| - |\rho(R_i)| \end{aligned} \tag{6}$$

To find the optimal compression mapping ρ we need to maximize Eq. (1) and minimize Eq. (6). Accordingly, we define a bi-objective optimization problem:

$$\begin{aligned}
&\underset{\rho}{\textbf{maximize}} \sum_{R_i \in \mathcal{R}} \mathrm{CV}_{\rho}(R_i) \\
&\underset{\rho}{\textbf{minimize}} \sum_{R_i \in \mathcal{R}} \mathrm{SE}_{\rho}(R_i) + \frac{|R_i| - |\rho(R_i)|}{2\epsilon} \\
&\text{where } \rho(R_i) \neq \emptyset \text{ and } \rho(R_i) \subset R_i
\end{aligned} \tag{7}$$

It is very hard to find the optimal compression mapping ρ. Our problem is very similar to the Knapsack problem which is known to be NP-complete. The vast amount of data generated due to the data explosion (big data) is making it even harder to explore all possible solutions. Accordingly, we present in this work a heuristic solution providing a good tradeoff between utility (i.e., smaller errors due to compression and noise) and privacy.

4.2 Querying our DP view

From a tensor $\mathcal{R} = \{R_1, \ldots, R_n\}$ we build a DP view $\tilde{\mathcal{R}} = \{\tilde{R_1}, \ldots, \tilde{R_n}\}$ where each region R_i has d dimensions with ranges $\tau_1^{R_i}, \ldots, \tau_d^{R_i}$. Region R_i corresponds to a compressed and perturbed region $\tilde{R_i}$.

The user can query our view $\tilde{\mathcal{R}}$ via count range queries. A range query Q is formulated by a simple SQL query as follows:

```
SELECT Sum(count)
FROM R
WHERE
   dim_1 between (s_1, e_1)
   and
   ...
   and
   dim_d between (s_d, e_d)
```

In the clause WHERE, each attribute dim_l targets range $[s_l, e_l]$ with $l \in 1, \ldots, d$. The answer of Q consists of the sum of all cells c resulting from the intersection between the ranges of Q and the regions of $\tilde{\mathcal{R}}$:

$$\begin{aligned}
&Q(\tilde{\mathcal{R}}) = \sum_{\tilde{R_i} \in \tilde{\mathcal{R}}} \left(\sum_{c \in Q \cap \tilde{R_i}} c + \eta_i \right) \\
&\text{where } Q \cap \tilde{R_i} = \prod_{l \in 1, \cdots, d} \left([s_l, e_l] \cap \tau_l^{\tilde{R_i}} \right), \\
&\tau_l^{\tilde{R_i}} \text{ is range of dimension } l \text{ for region } \tilde{R_i},\ \eta_i = \sum_{\eta_{i,j} \in N} \eta_{i,j} \text{ and} \\
&N = \{\text{Noise } \eta_{i,j} \text{ of subregion } r_{i,j} \in \Delta(\tilde{R_i}) \mid Q \cap r_{i,j} \text{ is nonempty}\}
\end{aligned} \tag{8}$$

The total noise η_i is added based on the noises in the subregions $\Delta(\tilde{R_i})$ associated to each region $\tilde{R_i}$ and overlapping with Q. Of course the efficiency of our method depends on the amount of noise added inside each compressed region. Instead of injecting random noise into each count cell, we show in Section 5 how we construct a DP view with minimal random noise that does not depend on the size of each region. The complexity of our query processing is $O(nd)$ where n is the number of regions and d is the number of dimensions.

5 OUR METHOD: SLIM-VIEW

In this section, we present our method SLIM-View, which constructs a differentially private view for a tensor, and the user can query this view without any privacy risk while maintaining the the utility of the query results.

5.1 Overview

Figure 2 shows the main execution flow of our solution. The tensor publication is driven by a workload that reflects the regions targeted by user queries (i.e. count range queries). In the absence of such workload, SLIM-View will split the tensor uniformly into a set of region and use this set as workload for remaining steps of the solution. Our private data release involves three main steps: (i) finding the best space (or size) allocation to compress each region; (ii) creating a view by randomly sampling the values to keep in each compressed region; and (iii) building the association table to manage the user query processing by injecting minimal noise into query responses.

The overall algorithm of SLIM-View is described in Algorithm 1. In the first step (Line 7), it searches for the best compression that provides efficient allocation space while maintaining utility (see Subsection 5.2). Once all spaces have been allocated to the regions, SLIM-View iteratively constructs the private view (Line 10), where each new region $\tilde{R_i}$ consists of a compressed region randomly constructed by an Exponential Mechanism (see Subsection 5.3). In order to minimize the noise to be injected, an association table AT is constructed iteratively (Line 11). It randomly divides R_i into subregions and samples minimal and random noise for each subregion. The minimal number of subregion is controlled by the hyperparameter λ (see Subsection 5.4).

To ensure that the created view does not depend on the presence or absence of an individual in the database, we need to introduce some randomness. To do that, well-known DP mechanisms are integrated into all steps of our solution. The privacy budget ϵ is distributed proportionally across the space allocation, sampling and association table phases, using two proportions (i.e., *alloc_prop* and *sampl_prop*) whose sum is strictly less than 1 (Lines 4-6). Furthermore, we will show in Subsection 5.5 how privacy guarantees are ensured at each step of SLIM-View.

Algorithm 1 SLIM-View

```
1:  Inputs: tensor $\mathcal{R} = \{R_1, \ldots, R_n\}$, privacy budget $\epsilon$, privacy proportion for space allocation $alloc\_prop$, privacy proportion for sampling $sampl\_prop$, hyperparameter $\lambda$.
2:  Outputs: private view $\tilde{\mathcal{R}} = \{\tilde{R_1}, \ldots, \tilde{R_n}\}$, association table $AT$.
3:  $\tilde{\mathcal{R}}, AT \leftarrow [\,], [\,]$
4:  $\epsilon_o \leftarrow \epsilon * alloc\_prop$            ▷ Privacy budget for space allocation
5:  $\epsilon_s \leftarrow \epsilon * sampl\_prop$            ▷ Privacy budget for sampling
6:  $\epsilon_n \leftarrow \epsilon - \epsilon_o - \epsilon_s$            ▷ Privacy budget for association table
7:  $space\_allocations \leftarrow optimize\_allocation(\mathcal{R}, \epsilon_o)$
8:  for $R_i \in \mathcal{R}$ do
9:      $b \leftarrow space\_allocations[R_i]$
10:     $\tilde{R_i} \leftarrow Exponentiel\_Mechanism\_Sampling(R_i, b, \epsilon_s)$
11:     $AT_i \leftarrow Association\_Table(R_i, b, \epsilon_n, \lambda)$
12:     $\tilde{\mathcal{R}}, AT \leftarrow \tilde{\mathcal{R}} \cup \tilde{R_i}, AT \oplus AT_i$
13: end for
14: return $\{\tilde{\mathcal{R}}, AT\}$
```

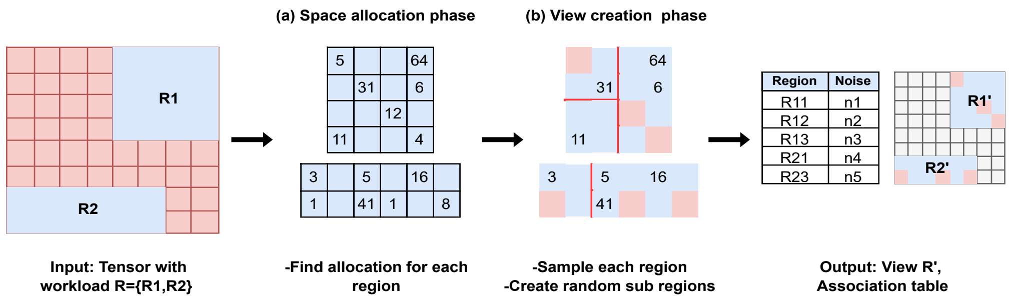

(a) Space allocation phase
(b) View creation phase
R1
R2
5
64
31
6
12
11
4
3
5
16
1
41
1
8
Region
Noise
R11
n1
R12
n2
R13
n3
R21
n4
R23
n5
R1'
R2'
Input: Tensor with workload R={R1,R2}
-Find allocation for each region
-Sample each region
-Create random sub regions
Output: View R', Association table

Figure 2: Our tensor publication.

5.2 Space Allocation Phase

In our bi-objective optimization problem Eq. (7) (see Section 4), each region $R_i \in \mathcal{R}$ undergoes two transformations: compression and perturbation. For the compression, there are two main issues: (i) how many non-empty cells to be included in the view for best utility?, and (ii) how to select the cells to be preserved in the view without any privacy threat? Addressing both issues simultaneously can be computationally expensive, due to the large volume of a data tensor. In this subsection, we will focus only on the first problem.

The objective is to determine the adequate allocation space (i.e., the number of cells to keep after compression) for each region in order to maintain utility. Given a compression mapping ρ, and the space allocated to region R_i for compression $b_i = |\rho(R_i)|$. We can therefore simplify the first objective function in Eq. (7) as follows:

$$\textbf{minimize} \quad \sum_{i=1}^{n} b_i \tag{9}$$

As for the second objective function in Eq. (7), to ensure utility, each b_i is chosen to minimize the SE sampling error which is directly due to the removal of non-empty cells. Note that $\text{SE}(R_i) = |S_i - S'_i|$ where S_i and S'_i are sums over R_i and its compression R'_i, respectively. To avoid manipulating the cells of R'_i, we can estimate the value of each cell in R'_i by averaging the cells of R_i, where $Avg(R_i) = S_i/|R_i|$. With this simplification, we can then estimate that S'_i is $b_i \times Avg(R_i)$. Therefore, the second objective function in Eq. (7) is approximated as follows:

$$\begin{aligned} &\textbf{minimize} \quad \sum_{i=1}^{n} |S_i - b_i \times Avg(R_i)| \text{ is rewritten into} \\ &\textbf{maximize} \quad \sum_{i=1}^{n} b_i \times Avg(R_i) \end{aligned} \tag{10}$$

The problem is therefore approximated in such a way that there is only one decision variable b_i, namely:

$$\begin{aligned} &\textbf{minimize} \quad \sum_{i=1}^{n} b_i \\ &\textbf{maximize} \quad \sum_{i=1}^{n} b_i \times Avg(R_i) \end{aligned} \tag{11}$$

Using both objective functions of Eq. (11), we can find a balance between compression and accuracy. However, a question remains: is it safe to compute these allocations without any privacy consideration, even if they are not public?

Privacy Risk with $\mathbf{Avg(R_i)}$. If we consider a case where regions in the tensor contain cells with big-scale values (e.g. 10^3) due to a lot of aggregations (see Section 3 on how to produce a tensor), the noise sampled from the Laplace distribution can hide the presence or the absence of a certain individual. However, it is unable to hide the information if a cell is empty or not. Since there is no restriction on the number of queries sent by the user, an attacker can infer the number of non-empty cells (i.e., allocation b_i) in each region of the published tensor. Using this knowledge of the database, the attacker can solve the allocation phase and compare the results he obtained with the allocation observed from the tensor. Since the allocation is calculated from the average of cells in each region and the presence/absence of an individual affects this value by modifying the sum of cells (or the sum and number of cells), this comparison will expose the the targeted individual.

Due to this privacy issue, we cannot guarantee DP if we use the optimization problem as defined Eq. (11), and we need to find a safe solution to the optimization problem. Two solutions are possible. The first is to create an algorithm to solve the multi-objective optimization while ensuring DP, which is outside the scope of our work. The second solution is to consider the optimization as a function and either perturb the input and consider it as a post-process to differentially private data, or perturb the output (the set of $b_1, \ldots, b_n$) of the optimization.

To consider perturbing the output of a function, it is necessary to compute its sensitivity in the case of neighboring databases. For the multi-objective optimization, we found that the global sensitivity was not bounded (the sensitivity analysis can be found in Appendix A 2.2 in the full technical report[1]) which makes it impossible to apply noise-based mechanisms like Laplace to its output.

In our case, we therefore only have the first option left: perturbing the input. Our optimization takes as input: the $Avg(R_i)$ of each region R_i and its size $|R_i|$. Instead of using the $Avg(R_i)$, we create a DP version $\overline{Avg(R_i)}$ of this value where: $\overline{Avg(R_i)} = Avg(R_i) + Lap(\frac{\Delta Avg}{\epsilon_o})$, with ϵ_o is the privacy budget and ΔAvg is the sensitivity of the average function.

How to define the sensitivity the average function? It is well-known that the sensitivity of the average is not bounded, but we can circumvent it and compute $\overline{Avg(R_i)}$. As $Avg(R_i) = \frac{Sum(R_i)}{|R_i|}$, we can obtain $\overline{|R_i|}$ using the Laplace mechanism with a sensitivity equal to 1 since it is a count. Similarly for $Sum(R_i)$, $\overline{Sum(R_i)}$ can be obtained with the Laplace mechanism with a sensitivity equal to 1 since it is a sum of counts. We can now get an ϵ_o-DP of $Avg(R_i)$, as follows: $\overline{Avg(R_i)} = \frac{Sum(R_i)+Lap(\frac{1}{\epsilon_o/2})}{|R_i|+Lap(\frac{1}{\epsilon_o/2})}$. Since $\overline{Sum(R_i)}$ and $\overline{|R_i|}$

[1]https://github.com/AlaEddineLaouir/SLIM-View

are $\frac{\epsilon_o}{2}$-DP. By sequential composition of DP (see Theorem 3.6) we can show that $\overline{Avg(R_i)}$ is ϵ_o-DP.

Thus, our optimization problem to find the necessary space allocation for each region is defined as follows:

$$\begin{aligned} &\textbf{minimize} && \textstyle\sum_{i=1}^{m} b_i \\ &\textbf{maximize} && \textstyle\sum_{i=1}^{m} b_i * \overline{Avg(R_i)} \\ &\textbf{where :} && 0 < b_i < Max\left(\overline{|R_i|}, 2\right) \end{aligned} \tag{12}$$

In Eq. (12), in addition to $\overline{Avg(R_i)}$, we use $\overline{|R_i|}$ to bound the maximum space that can be allocated to region R_i because using the original $|R_i|$ may leak information to the attacker about individuals in the database. To understand this risk, consider the same case as before, where the attacker can identify non-empty cells. We assume that the attacker knows all individuals in region R_i as well as its size $|R_i|$ and that the maximum allocation to this region for compression is $|R_i'| = |R_i| - 1$. If a new individual is added to the database and leads to the creation (from empty to full) of a new cell. In this case, the new size $|R_i'|$ is equal to the previous one (known to the attacker) $|R_i'| + 1$. For another compression, the maximum allocation for that region may change and the attacker would notice that the maximum allocation has changed, which simply means a new individual is added to the database.

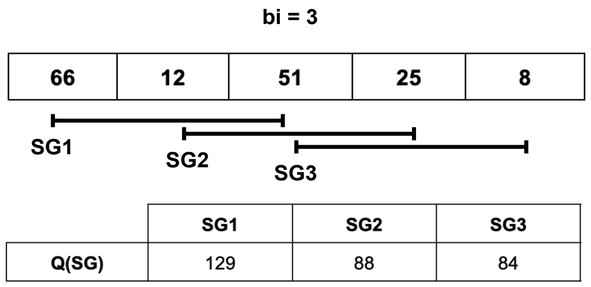
5.3 Sampling by Exponential Mechanism

After computing for each region R_i its allocation b_i (the number of cells to keep after compression), SLIM-View starts building the private view of each region. Based on our definition of utility Eq. (7), the best solution will be obtained by selecting the Top_{b_i} cells which minimize the difference between the sums S_i and S_i' obtained respectively before and after compression.

But in terms of differential privacy, we need to ensure that any transformation applied is random or stable to ensure that the view created does not leak any information about a particular individual. However, the stability analysis reveals that the Top_b algorithm is 2-*stable* (see Stability analysis of deterministic sampling in Appendix A 2.1 in the full technical report[2]). This means (see Theorem 3.10) that the behavior of the algorithm is sensitive to the presence/absence of individuals, which implies that the noise to be added at the end must further increase to mask this sensitivity of the Top_b algorithm but at the expense of utility.

To avoid adding too much noise, we propose a new sampling technique that not only ensures DP but also tries to maximize utility like the Top_b algorithm. Our solution, given in Algorithm 2, uses the Exponential Mechanism and consumes a dedicated privacy budget ϵ_s to select a sample of b cells from region R, which preserves utility without any privacy risk.

As the number of possible subgroups (or samples) of size b that can be constructed from region R is very large (i.e., $\binom{|R|}{b}$), Algorithm 2 lists cells of the region in random order (Line 3). Then it defines subgroups of size b as consecutive overlapping cells, where each subgroup SG_j has a particular starting position $starting\_index_j$ (see Figure 3). This representation amounts to sampling $|R| - b + 1$ random subgroups from which the algorithm can choose. For the



	SG1	SG2	SG3
Q(SG)	129	88	84

Figure 3: Exponential mechanism sampling.

quality metric, Algorithm 2 uses the sum of b consecutive and overlapping cells to compute the quality of each subgroup SG_j:

$$Q(SG_j) = Sum(R, starting\_index_j, b) \tag{13}$$

For the sensitivity Δ_Q of the quality metric Q, it is equal to 1 since Q is just a count and the maximum effect of an individual on a count is 1.

Algorithm 2 *Exponential_Mechanism_Sampling*

```
1: Inputs: region $R$, allocation size $b$, privacy budget $\epsilon_s$
2: Output: sampled view $R'$
3: $R \leftarrow shuffle\_values(R)$
4: $starting\_indexes, scores, subgroups\_prob \leftarrow [], [], []$
5: $starting\_indexes \leftarrow get\_possible\_starting\_indexes(R, b)$
6: for $i \in \{1, \ldots, |starting\_indexes|\}$ do
7:     $scores[i] \leftarrow Sum(R_i, starting\_indexes[i], b)$
8: end for
9: for $j \in \{1, \ldots, |scores|\}$ do
10:     $subgroups\_prob[j] \leftarrow exp\left(\frac{\epsilon_s * scores[j]}{2}\right)$
11: end for
12: $index \leftarrow random\_choice(starting\_indexes, subgroups\_prob)$
13: $R' \leftarrow extract\_cells(R, b, index)$
14: return $R'$
```

Each subgroup is assigned a probability of being retained (Lines 9-11) based on the score obtained from the quality metric function (Lines 6-8): the higher the score, the higher the probability. Then, a subgroup with the highest score is randomly selected (Line 12) with higher probability. Finally, the *extract_cells* function (Line 13) creates the region R' using the cells from the randomly selected subgroup.

5.4 Association Table

To publish the private view of the tensor, the data must be perturbed before answering the queries. A basic technique for perturbing a tensor is to add noise to each cell, which will yield poor utility due to the accumulated noise of each cell [5]. So, rather than injecting noise into each cell, SLIM-View adds noise to random groups of cells and therefore less noise is accumulated for each query. The noise of each group of cells is stored with the coordinates (for the ranges) of the group in a tensor. To define these groups, we can use

[2]https://github.com/AlaEddineLaouir/SLIM-View

the workload regions as groups, but since the workload is known, this can help estimate the added noise.

Privacy risk related to noise estimation. We assume that the attacker knows a part of a region is empty. If all cells in that region have the same noise coming from the Laplace Mechanism and one cell has a different value than those around it, the attacker can conclude that a certain individual has been added to the database.

Algorithm 3 is part of SLIM-View to create groups of cells (called *sub_regions*) for each region. Since these *sub_regions* will be stored (only their coordinates and noise), their size should not affect the compression achieved with sampling.

Algorithm 3 *Association_Table*

```
1:  Inputs: region $R$ , allocation size $b$, privacy budget $\epsilon_n$, hyper-
    parameter $\lambda$.
2:  Output: association table $AT_R$.
3:  $subregions \leftarrow [R]$
4:  $nb\_subregions \leftarrow get\_random\_between(0, (|R| - b) * \lambda)$
5:  while $nb\_subregions > 0$ do
6:      $random\_region \leftarrow select\_random(subregions)$
7:      $left\_sub, rigth\_sub \leftarrow split\_randomly(random\_region)$
                    ▷ Replace the selected sub_region by the new ones
8:      $subregions \leftarrow remove(subregions, random\_region)$
9:      $subregions \leftarrow add(subregions, left\_sub)$
10:     $subregions \leftarrow add(subregions, rigth\_sub)$
11:     $nb\_subregions \leftarrow nb\_subregions - 1$
12: end while
13: $AT_R \leftarrow []$
14: for $subregion \in subregions$ do
15:     $noise \leftarrow sample\_from\_Laplace(\epsilon_n)$
16:     $AT_R \leftarrow add(AT_R, \{subregion, noise\})$
17: end for
18: return $AT_R$
```

Algorithm 3 tries to determine the number *nb_subregions* of *sub_regions* to be created from a given region R (Line 4). Note that *nb_subregions* is defined randomly but bounded by the space gained ($|R| - b$) in that region R. Since the process computing the allocation space b for R is DP, the algorithm dividing R into subregions does not need to consume any privacy budget. The hyperparameter λ is used to set the portion of the space gained from the previous SLIM-View step that will be allocated for the association table AT_R. As long as the space allocated to AT_R is not consumed (Line 5), Algorithm 3 divides a randomly selected region (*random_region*) of *sub_regions* into two subregions *left_sub* and *rigth_sub* (Line 7). Each point in the domain of *random_region* has the same probability p of being the division point, which is equal to $p = \frac{1}{||random\_region||}$ where $||random\_region||$ is the domain size of region R (see in Section 3 how the domain size is computed). This random choice of split point helps SLIM-View eliminate **the risk of noise estimation**.

Since a cell can be perturbed individually with probability $p_c = d * \frac{1}{||random\_region||}$ where d is the number of dimensions (or attributes) used in R, there is uncertainty in inferring whether or not an individual is in the database. The new sub-regions *left_sub* and *rigth_sub* will replace *random_region* (Lines 9 − 11). If the allocated space is consumed, the creation of *sub_region* will stop and the next step is to create the association table AT_R. Each *sub_region* in *sub_regions* will be stored with random noise from the Laplace Mechanism with budget ϵ_n (Lines 15-17).

Note that if the algorithm fails to split a region, it will simply ignore that region and choose another (if it exists) or stop and move to Line 14. This processing has been omitted for Algorithm 3 for the sake of simplicity.

5.5 Privacy Accounting

Since each region $R \in \mathcal{R}$ is sampled and perturbed independently, to measure the privacy budget consumed by SLIM-View to create the private view $\tilde{\mathcal{R}}$, we can track the consumption necessary for a region R. Using the parallel composition property of DP (see Theorem 3.7), we obtain the total budget consumption of SLIM-View.

Each region R goes through three main steps: space allocation, cell sampling and association table. In the first step, each region perturbs its average (Avg) using the Laplace Mechanism while consuming ϵ_o before passing it to the solver in order to find a solution to our optimization problem Eq.(12). Given two neighbouring tensors $\mathcal{R}$ and $\mathcal{R}'$. Let b be the space allocated to region R. According to the *post-processing* property of DP [5], the following equation holds:

$$\left|\frac{\Pr[b|\mathcal{R}]}{\Pr[b|\mathcal{R}']}\right| = \left|\frac{\Pr[Avg(R)|\mathcal{R}]}{\Pr[Avg(R)|\mathcal{R}']}\right| \leq exp(\epsilon_o) \tag{14}$$

where $\Pr[b|\mathcal{R}]$ is the probability that a region R from $\mathcal{R}$ gets space allocation b.

In the sampling phase, SLIM-View uses the Exponential Mechanism to select a subgroup (or a sample) of cells while consuming the privacy budget ϵ_s. For a subgroup SG of cells to be selected, the following equation holds:

$$\left|\frac{\Pr[SG(R)|\mathcal{R}]}{\Pr[SG(R)|\mathcal{R}']}\right| \leq exp(\epsilon_s) \tag{15}$$

Lastly, to create the association table (AT_R) for a region R, SLIM-View needs to determine the number of subregions *nb_subregions* based on the allocation space b. Using the *post-processing* property of DP [5], the following equation holds for *nb_subregions* and no additional privacy budget needs to be consumed:

$$\left|\frac{\Pr[nb\_subregions|\mathcal{R}]}{\Pr[nb\_subregions|\mathcal{R}']}\right| = \left|\frac{\Pr[b|\mathcal{R}]}{\Pr[b|\mathcal{R}']}\right| \leq exp(\epsilon_o) \tag{16}$$

The creation of subregions is random and data-independent, so no privacy budget is consumed/needed at this stage. For noise injection into each subgroup, a random noise is sampled using the Laplace mechanism and the same privacy budget ϵ_n is consumed for each subgroup.

For simplicity, we consider $\epsilon_{sn} = \epsilon_s + \epsilon_n$ the budget used by the algorithms sampling and noise injection, in order to create $\{R', AT_R\}$, the compressed region of R with its association table. Based on Equations (14, 15 and 16) and the sequential composition property of DP (see Theorem 3.6), for a region R, SLIM-View uses $\epsilon = \epsilon_o + \epsilon_{sn}$ to create a DP view $\{R', AT_R\}$ from allocation to publication. Therefore, the following equation holds:

$$\left|\frac{\Pr\left[b|\mathcal{R}\right]}{\Pr\left[b|\mathcal{R}'\right]}\right| \cdot \left|\frac{\Pr\left[\{R', AT\_R\}|\mathcal{R}\right]}{\Pr\left[\{R', AT\_R\}|\mathcal{R}'\right]}\right| \leq exp(\epsilon_o).exp(\epsilon_{sn}) = exp(\epsilon) \tag{17}$$

where $\Pr\left[\{R', AT_R\}|\mathcal{R}\right]$ is the probability that a R' and AT_R are created from the tensor $\mathcal{R}$. This ensures that each published region guarantees ϵ-DP, and since each region consumes the same budget and based on the parallel composition property of DP, the private view of the tensor $\mathcal{R}$ also satisfies ϵ-DP .

6 EXPERIMENTS

In this section, we describe the performance evaluation of SLIM-View on different datasets and workloads by comparing with existing solutions in the literature.

6.1 Experiments Setup

Datasets and workloads. In our experiments (code[3]), we used five well-known datasets: (i) Adult[4] is a benchmark dataset. (ii) Numerical Adult (nume-Adult) is a projection only on the numeric attributes of Adult, containing demographic information about individuals and their revenues. (iii) Bitcoin[5] contains records of transactions carried out within the Bitcoin blockchain. (iv) Traffic[6] is a dataset of metro traffic volume in terms of passengers. (v) Electricity[7] contains data on an electricity market transfer. This dataset is also used in our experiments to ensure that our tests are performed on a wide variety of data with different statistical properties.

Datasets	#Dimensions	#Workloads	#Queries	Domain
Adult	15	8	8×100	8.9×10^{18}
nume-Adult	8	119	5×3000	2.3×10^{11}
Trafic	8	238	5×3000	1.3×10^{14}
Bitcoin	9	456	5×3000	3.4×10^{11}
Electricity	8	238	5×3000	9.6×10^{13}

Table 1: Dataset and workload statistics.

Evaluation metrics. In our evaluation, we focused on three key aspects: utility, scalability, and space efficiency. We used the RMSE (Root-Mean-Square Error) to measure utility by calculating the average distance between the original responses and those in the perturbed view of the workload. RMSE $=\sqrt{\frac{1}{n}\sum_{Q\in W}(Q(C)-Q(C'))^2}$ where W is the workload and C, C' are the original, private view of the tensor (regardless of the method used to create C'). For the scalability aspect, we measured the time necessary to create the private view, Execution time = $Time_{end} - Time_{start}$, which allows us to capture the resources needed from each solution to create the private view. As for spatial efficiency, we measured the size difference between the original tensor and the created private view, called compression ratio, where: compression ratio = $\frac{|C|-|C'|}{|C|}$.

[3],https://github.com/AlaEddineLaouir/SLIM-View

[4]http://archive.ics.uci.edu/ml/datasets/Adult

[5]https://archive.ics.uci.edu/dataset/526/bitcoinheistransomwareaddressdataset

[6]http://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume

[7]https://www.openml.org/d/151

Experimental workloads. For a given dataset and a specified number of dimensions, numerous random workloads, along with their corresponding tensors, are generated. Refer to table 1 for detailed statistics on datasets and workloads. Each workload, along with its respective tensor, is then submitted to each DP solution for the view creation and evaluation processes. Workload-dependent solutions use workload not only for evaluating views, but also for creating them.Each evaluation is run ten times and the final results are averaged.

SLIM-View and Competitors. SLIM-View can create a DP view with or without a given workload. We also consider in our experiments the case of SLIM-View Without-Workload (SLIM-View-WW) as a separate solution for creating views. Instead of having a workload as an input, SLIM-View-WW will divide the tensor uniformly and consider the set of region obtained as a workload for view creation. For the sake of these experiments, we set the number of uniform regions created by SLIM-View-WW to 2^d where d is the number of dimensions of the tensor. For the allocation phase, SLIM-View uses the standard Pymoo solver[8]. We compared SLIM-View and SLIM-View-WW with many recent solutions in the literature (see Section 2), namely: (i) HDPView [11] and PrivTree [23] for partitioning approaches; (ii) HDMM [15] for workload-dependent approaches; (iii) PrivBayes [22] and P3GM [18] for generative models creating DP datasets.

Hyperparameters. In all experiments, we set $\epsilon = 0.1$. For SLIM-View (and SLIM-View-WW) we set $alloc\_prop = 0.5$, $sampl\_prop = 0.25$, $\lambda = 0.05$ and we provided also in **Appendix A.1 in the full technical report**[3] a complete analysis of the effect of each of these parameters on the performance of SLIM-View. For the other solutions, we used the same hyperparameter values as those used in their original experiments.

Hardware configuration. To run the experiments, we used a cluster of servers with the following specifications for each server: 2 CPUs Intel Xeon E5-2620 v3 6 cores/CPU x86_64 with 64 GiB of RAM and 2 x Nvidia GTX 980. Each solution is assigned a separate server.

6.2 Workload-independent solutions

For each number of dimensions between 2 and 6, we generated 3000 queries divided into random workloads based on their set of dimensions, used to compare with [11, 18, 22, 23] .

Based on the results presented in Figure 4, SLIM-View and SLIM-View-WW are able to provide smaller RMSE (better utility) than the closer competitor [18] by up to several orders of magnitude on 6-dimensional tensors. SLIM-View-WW is able to outperform SLIM-View on low dimension tensors, which makes sense since SLIM-View is based on limited workloads to find the best sampling solution. With higher number of dimensions (less aggregation), SLIM-View has more data points to manage, therefore more chance of finding the best sampling solution. We also notice in Figure 4 that the higher the dimensions, the better the other solutions gets. To investigate this further, we conducted a second experiment using Adult dataset with dimensions between 7 and 14 with a query workload of 100 (and a tensor) for each set of dimensions.

[8]https://pymoo.org

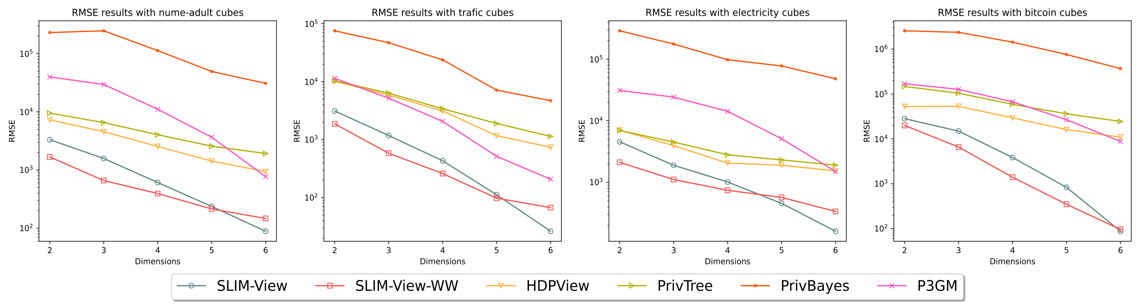


Figure 4: RMSE for workload-independent solutions.

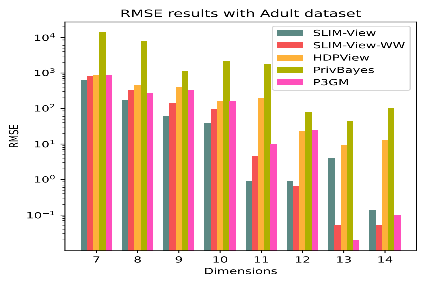


Figure 5: RMSE for workload-independent solutions using Adult dataset.

The results in Figure 5 show that SLIM-View and SLIM-View-WW still outperform all other solutions in terms of utility, only in 13 and 14 dimensions that P3GM was able to catch up with. Since the Adult dataset has dimensions of 15 (Table 1), few aggregations are performed in the 13- and 14-dimensional tensors. Thus, the cell values are mostly 0 or 1 and P3GM was able to catch up and only then produce a view comparable to SLIM-View and SLIM-View-WW.

Regarding the time needed to construct the private view which indicates the scalability of the solution, we compared SLIM-View and SLIM-View-WW with HDPView [11] and PrivTree [23]. P3GM was not considered because it requires training a deep learning model which requires a lot of time and resources. For PrivBayes, it is coded in C++ and called as an external process from a Python script, which makes the comparison unfair since all others are written directly in Python. But it should be noted that PrivBayes was fast compared to P3GM.

Based on Figure 6, all algorithms take insignificant and comparable time to create the view based on a low-dimensional tensor. As the number of dimensions increases, HDPView [11] and PrivTree [23] take longer to generate a view. On 6-dimensional tensors, SLIM-View takes less than 1 second on average while HDPView [11] is at least x10 times and PrivTree [23] takes x7 times longer on average to create the private view. Even though [11, 23] are meant to be scalable, they are strongly affected by the size of the tensor domain: the larger the size the more partitioning is required. The same goes for SLIM-View-WW: the higher the dimensions, the longer it takes to handle all tensor regions. The results in Figure 6 highlight the importance of creating workload-based views for scalability, as this allows the algorithm to focus on only a small portion of the tensor.

Regarding space efficiency, only SLIM-View, SLIM-View-WW, HDPView and PrivTree that have this property. Based on the results of Figure 7, HDPView [11] is consistently better and it is able to generate fewer blocks than PrivTree [23]. Compared to our approaches SLIM-View and SLIM-View-WW, both [11] [23] outperform them in terms of compression rate and space efficiency. SLIM-View and SLIM-View-WW are less space efficient due to the possible range of values they give for each b_i in the allocation phase (Eq. 12). This allows the solver to allocate as much space as necessary to preserve *utility*, as shown in figure 4. Given the proposed setup, the best result we obtained is a space saving of slightly more than 35%, but it can be improved by setting an upper limit. In section 7 we will discuss this point in more detail.

6.3 Workload-based solutions

Since HDMM [15] does not offer any space efficiency, the comparison will focus on utility and scalability based on execution time and RMSE. In this analysis, we were unable to extend the experiments to high dimensions as done previously in Section 6.2. This limitation is due to the algorithm's complexity, specifically HDMM's computational complexity of $O(n^3)$ where n is the data domain. Comparing the results presented in Figure 8 with respect to execution time, all algorithms completed their runs in less than a second on two-dimensional tensors. However, with higher dimensions, such as four-dimensional tensors based on Traffic dataset, SLIM-View outperforms HDMM [15] by a factor of x40, as expected given its lower complexity. Note that the time measured for HDMM only concerns the optimization part, and we have not taken into account the transformation of the data and the workload into "*Implicit workload representation*" and return to normal, both of which are costly. For this reason, it was very difficult to scale this algorithm to all of our experimental data and we were limited to running it only on low-dimensional tensors.

For the utility, and according to the results in Figure 8, the performance of HDMM [15] in terms of RMSE degrades as the tensor grows. This is due to the huge number of empty cells to which HDMM [15] added noise. This noise will subsequently be accumulated by the queries. From Figure 8, HDMM [15] outperforms SLIM-View when applied on 2-dimensional tensors, but for 3- and 4-dimensional tensors, SLIM-View and SLIM-View-WW have better performance results in terms of RMSE.

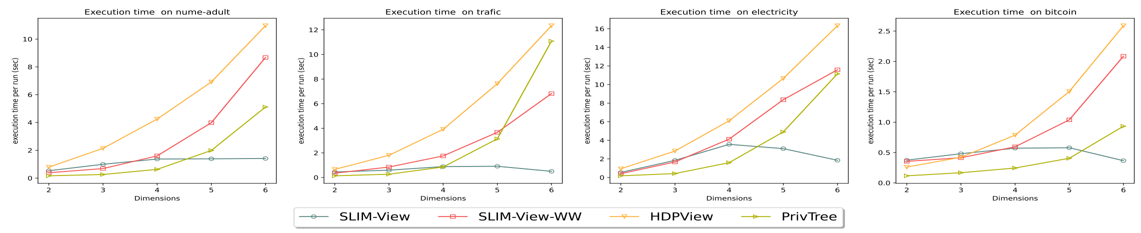


Figure 6: Execution time workload-independent solutions.

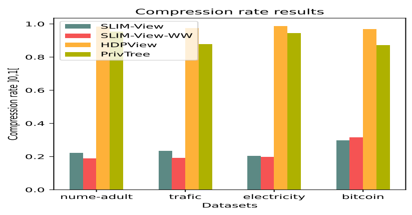


Figure 7: Space efficiency for workload-independent solutions

7 DISCUSSION

In this section, we expand on some discussion points that can be considered a limitation or advantage for SLIM-View. According to the results presented in Section 6, SLIM-View offers better utility and scalability than any other solution. But for compression, [11] [23] partition techniques offer more space efficiency than SLIM-View. This is a direct consequence of the allocation phase, in which the solver (i.e. to find a solution to the optimization problem defined in Eq.(12)) leverages between utility and spatial efficiency and is given a wide range of values to select the best b_i for each region R_i where $0 < b_i < Max\left(\overline{|R_i|}, 2\right)$. This can be adjusted by setting a lower upper bound for each b_i as follows. Let us say we want to guarantee a compression ratio of 50%. We can then compute the new upper bound $|R_i|'$ by, $|R_i|' = 0.5 * |R_i|$ and make it differentially private $\overline{|R_i|}'$ as shown in Section 5.2, then give it to the solver in order to limit the allocation for each b_i with $0 < b_i < Max\left(\overline{|R_i|'}, 2\right)$. Depending on the application's needs, the database administrator can adjust the upper bound, but this will certainly come at a cost in terms of utility.

We have also focused in this work on the applicability aspect of SLIM-View (and SLIM-View-WW) in the real world, and in many scenarios, where the data is updated frequently (e.g., daily new data is added to the data warehouse from application servers). Based on our experiment results, we find that all existing approaches do not scale well in terms of resource consumption for each execution, which makes it difficult to adopt them. As such, SLIM-View appears to be best at this scalability.

8 CONCLUSION

In this work, we addressed the issue of publishing multidimensional data under differential privacy. We proposed SLIM-View, a new scalable approach that deals with this problem by creating a space-efficient view using sampling while maintaining high utility levels. The results of our experiments demonstrated that SLIM-View was able to fulfill all desired properties and outperform existing techniques by several orders of magnitude in different scenarios and on different datasets. In future work, we plan to explore other aspects of our method in distributed and federated databases and adapt it to graph-structured data.

REFERENCES

[1] [n. d.]. Postgres cube — a multi-dimensional cube data type. https://www.postgresql.org/docs/current/cube.html. Accessed: 2023-12-30.
[2] John M Abowd. 2018. The US Census Bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2867–2867.
[3] Gergely Acs, Luca Melis, Claude Castelluccia, and Emiliano De Cristofaro. 2018. Differentially private mixture of generative neural networks. *IEEE Transactions on Knowledge and Data Engineering* 31, 6 (2018), 1109–1121.
[4] Bolin Ding, Marianne Winslett, Jiawei Han, and Zhenhui Li. 2011. Differentially private data cubes: optimizing noise sources and consistency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 217–228.
[5] Cynthia Dwork. 2006. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*. Springer, 1–12.
[6] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
[7] Hamid Ebadi, Thibaud Antignac, and David Sands. 2016. Sampling and partitioning for differential privacy. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 664–673.
[8] Ju Fan, Tongyu Liu, Guoliang Li, Junyou Chen, Yuwei Shen, and Xiaoyong Du. 2020. Relational data synthesis using generative adversarial networks: A design space exploration. *arXiv preprint arXiv:2008.12763* (2020).
[9] Frederik Harder, Kamil Adamczewski, and Mijung Park. 2021. Dp-merf: Differentially private mean embeddings with randomfeatures for practical privacy-preserving data generation. In *International conference on artificial intelligence and statistics*. PMLR, 1819–1827.
[10] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. 2019. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*.
[11] Fumiyuki Kato, Tsubasa Takahashi, Shun Takagi, Yang Cao, Seng Pei Liew, and Masatoshi Yoshikawa. 2022. HDPView: Differentially Private Materialized View for Exploring High Dimensional Relational Data. *Proc. VLDB Endow.* 15, 9 (2022), 1766–1778. https://www.vldb.org/pvldb/vol15/p1766-kato.pdf
[12] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.
[13] Chao Li, Michael Hay, Gerome Miklau, and Yue Wang. 2014. A data-and workload-aware algorithm for range queries under differential privacy. *arXiv preprint arXiv:1410.0265* (2014).
[14] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. 2015. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB journal* 24 (2015), 757–781.
[15] Ryan McKenna, Gerome Miklau, Michael Hay, and Ashwin Machanavajjhala. 2018. Optimizing error of high-dimensional statistical queries under differential privacy. *arXiv preprint arXiv:1808.03537* (2018).
[16] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2014. Priview: practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1435–1446.

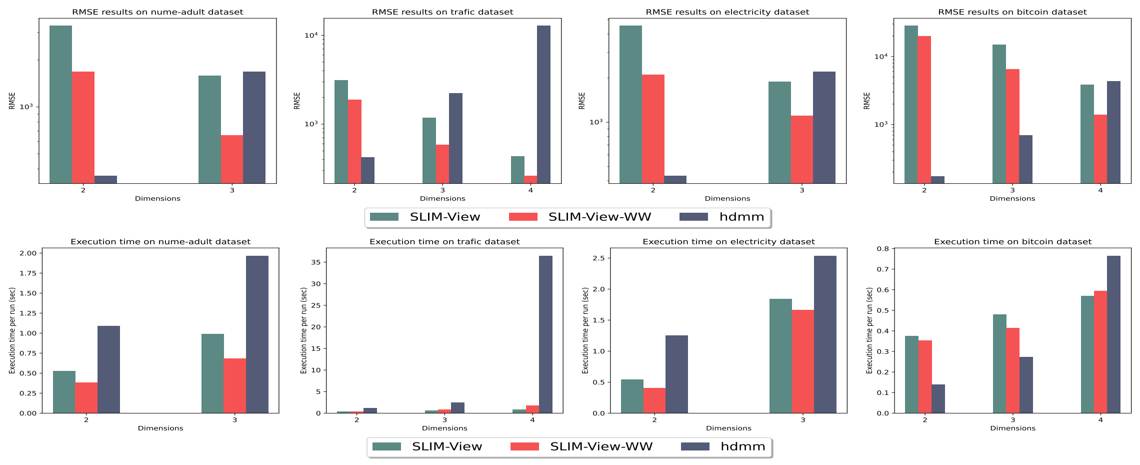


Figure 8: Comparison between SLIM-View, SLIM-View-WW and HDMM in terms of RMSE and execution time.

[17] Graham Cormode Cecilia Procopiuc Divesh Srivastava, Entong Shen, and Ting Yu. [n. d.]. Differentially Private Spatial Decompositions. ([n. d.]).

[18] Shun Takagi, Tsubasa Takahashi, Yang Cao, and Masatoshi Yoshikawa. 2021. P3GM: Private high-dimensional data release via privacy preserving phased generative model. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 169–180.

[19] Apple Team. [n. d.]. learning-with-privacy-at-scale. ([n. d.]). https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf

[20] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. 2020. Differentially private SQL with bounded user contribution. *Proceedings on privacy enhancing technologies* 2020, 2 (2020), 230–250.

[21] Chugui Xu, Ju Ren, Yaoxue Zhang, Zhan Qin, and Kui Ren. 2017. DPPro: Differentially private high-dimensional data release via random projection. *IEEE Transactions on Information Forensics and Security* 12, 12 (2017), 3081–3093.

[22] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2017. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)* 42, 4 (2017), 1–41.

[23] Jun Zhang, Xiaokui Xiao, and Xing Xie. 2016. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 international conference on management of data*. 155–170.

[24] Xiaojian Zhang, Rui Chen, Jianliang Xu, Xiaofeng Meng, and Yingtao Xie. 2014. Towards accurate histogram publication under differential privacy. In *Proceedings of the 2014 SIAM international conference on data mining*. SIAM, 587–595.

[25] Zhikun Zhang, Tianhao Wang, Jean Honorio, Ninghui Li, Michael Backes, Shibo He, Jiming Chen, and Yang Zhang. 2021. Privsyn: Differentially private data synthesis. (2021).

A APPENDIX

A.1 Sensitivity analysis of Hyperparameters

In this section, we show the behaviour of SLIM-View based on the values given for the hyperparameters. In each experiment, we vary one parameter while the rest keep their default values, and we measure the performance of SLIM-View based on the RMSE. The hyperparameters and their default values are: $\epsilon = 0.1$, $alloc_{prop} = 0.5$, $sampl\_prop = 0.25$, $\lambda = 0.05$.

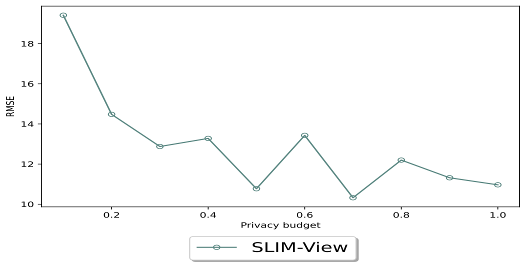


Figure 9: Impact of ϵ on SLIM-View performance in terms of RMSE.

epsilon ϵ: Here, we measured the performance of SLIM-View based on the RMSE obtained by changing the privacy budget ϵ. The smaller value of ϵ means a higher privacy budget, which leads to more noise (randomness) in the result. From Figure 9, we can see that SLIM-View behaves accordingly to the characteristics of DP.

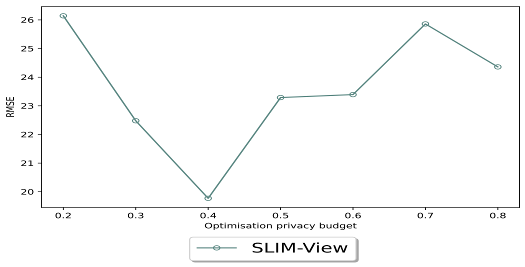


Figure 10: Impact of $alloc_{prop}$ on SLIM-View performance in terms of RMSE.

alloc$_{\mathbf{prop}}$: We varied here only the value of $alloc_{prop}$ (and $sampl_{prop} = (1 - alloc_{prop})/2$), and the bigger the ratio, the less privacy budget is consumed for the allocation phase. Adding too much noise using the Laplace mechanism to perturb the *Avg* of each region results in less than optimal allocations. Therefore, giving less budget allows for better allocation. On the other hand, the sampling and noise injection steps of SLIM-View will have a larger privacy budget. From the results in Figure 10, we see that SLIM-View performs better when the budget is split evenly between the allocation (taking half) and the remaining steps.

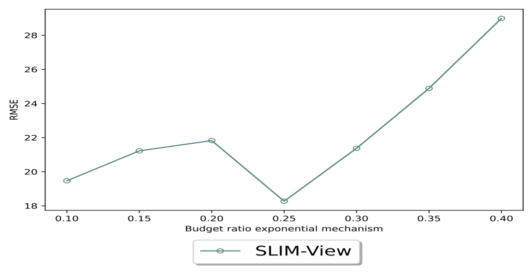


Figure 11: Impact of $sampl_{prop}$ on SLIM-View performance in terms of RMSE.

$\mathbf{sampl_{prop}}$: We tested the impact of $sampl_{prop}$ on SLIM-View performance using exponential mechanism sampling, where the smaller the ratio, the higher the privacy budget for the exponential mechanism and less for the Laplace mechanism to add noise. From Figure 11, we can see that when a larger privacy budget is given, the exponential mechanism of SLIM-View provides better results because it maintains a bias towards the better sample. Meanwhile, when a larger privacy budget is given to the Laplace mechanism, we see that the RMSE value increases.

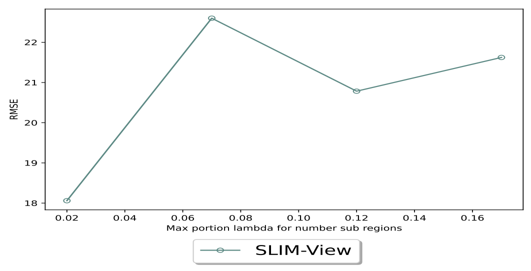


Figure 12: Impact of λ on SLIM-View performance in terms of RMSE.

lambda λ: Since there is a direct link between the compression ratio and λ, we can easily deduce the maximum effect of λ on compression. In this experience, we show the effect of λ on the performance of SLIM-View based on the RMSE. The larger λ lambda, the bigger the probability of generating numerous *sub_regions*, and for each, a noise from the Laplace mechanism is sampled. The probability of generating significant random noise that negatively affects the utility is important. We can see this phenomenon with the results provided by SLIM-View in Figure 12.

A.2 Stability computation of deterministic sampling

A.2.1 Deterministic sampling. According to our Utility metric, we obtain the best results if the algorithm is able to preserve the cells with the highest values (we consider only the non empty cells). Let's consider a sampling algorithm Top_b that takes as input a region R and b the allocation given to it and returns a region R" which contains only the biggest b cells of R:

Algorithm 4 Top_b

```
1: Inputs: $R$ - region, $b$ - allocation
2: Output: $R''$ - sampled view of $R$
3:
4: $sorted\_cells \leftarrow bubble\_sorting\_cells\_in\_ascending\_order(R)$
5: $top\_b\_cells \leftarrow get\_last\_cells(sorted\_cells, b)$
6: $R" \leftarrow region(top\_b\_cells)$
7: return $R''$
```

Let $R = [c_1, c_2, c_3, c_4]$ be a region where $c_1 = c_2 = c_3 = c_4$. In this case if we take $b = 2, Top_b$ will pick that last two elements in R since all the cells have equal values, and $Top_b(R, 2)$ gives $Top_b(R, 2) = [0, 0, c_3, c_4]$. If we consider neighbouring region $R' = [c'_1, c_2, c_3, c_4]$ where $c'_1 = c_1 + 1$, in this case $Top_b(R', 2)$ gives $Top_b(R', 2) = [c'_1, 0, 0, c_4]$. Based on these results, $Top_b(R, 2) \, \Delta \, Top_b(R', 2) = 2$ which means that Top_b is at least 2-*stable*.

A.2.2 Deterministic allocation phase. To analysis the sensitivity of the allocation phase, we consider a deterministic solution DA to solve optimisation problem given as follows:

$$\begin{aligned} &\textbf{minimize} && \textstyle\sum_{i=1}^{m} b_i \\ &\textbf{maximize} && \textstyle\sum_{i=1}^{m} b_i * Avg(R_i) \\ &\textbf{where :} && 0 < b_i < |R_i| \end{aligned} \tag{18}$$

This solver, DA, works as follows :

(1) First computes all possible values PB for B where : $B = \sum_{i=1}^{m} b_i$.
(2) For each region compute a weight $w_i = \frac{|R_i|}{\sum_{i=1}^{m} |R_i|}$ that will be used to compute the allocation of the i_{th} region
(3) For each value B in PB:
    (a) Compute b_i of each region where : $\left\lfloor \overline{b_i} \right\rfloor \leq b_i \leq \left\lceil \overline{b_i} \right\rceil$ and $\overline{b_i} = w_i * B$ and $B = \sum_{i=1}^{m} b_i$.
    (b) Compute $S = \sum_{i=1}^{m} b_i * Avg(R_i)$.
(4) Return the solution(s) with minimum B and maximum S by measuring the minimum distance from both objectifs : $dist = (\overline{S} - S) + (B - m)$ where m is the minimum allocation (each region gets 1) and $\overline{S} = \sum_{i=1}^{m} (|R_i| - 1) * Avg(R_i)$.

Based on this solver, we will analysis the sensitivity of computing regions allocations. Let's consider a dataset with two regions $D = \{R_1, R_2\}$, where all cells are equal to 1. In this case, for any B chosen in step 3 of the solver $S = \sum_{i=1}^{2} b_i * 1 = B \implies S = B \implies dist = (\overline{S} - S) + (B - 2) = (\overline{S} - B) + (B - 2)$. Which implies that $dist = \overline{S} - 2$ is constant for all possible solutions B, and all the solutions have the same quality based on $dist$ metric.

If we consider $D' = \{R'_1, R_2\}$, where $|D \Delta D'| = 1$ and $|R_1 \Delta R'_1| = 1$ and $Sum(R'_1) = Sum(R_1) + 1$ and $|R'_1| = |R_1|$. In this case of neighbouring, $Avg(R'_1) = \frac{Sum(R'_1)}{|R'_1|} = \frac{Sum(R_1)+1}{|R_1|} = \frac{Sum(R_1)}{|R_1|} + \frac{1}{|R_1|} =$

$1+\frac{1}{|R_1|}$. So for any possible solutions in step 3, $S = b'_1(1+\frac{1}{|R_1|})+b'_2 = B + \frac{b'_1}{|R_1|} \implies S = B + \frac{b'_1}{|R_1|} \implies dist = (\overline{S} - S) + (B - 2) = (\overline{S} - B + \frac{b'_1}{|R_1|}) + (B-2)$. Which implies that $dist = \overline{S} - 2 - \frac{b'_1}{|R_1|}$, so for each solutions a different *dist* value. We can notice that the more allocation is given to the first region, b'_1, the less *dist* gets. So the best solution B_{best} is obtained when $b'_1 = |R_1| - 1$, which implies $b'_2 = B_{best} - |R_1| + 1$.

So based on this case of neighbouring, $\{b_1, b_2\} \ \Delta \ \{b'_1, b'_2\} = |b'_1 - b_1| + |b'_2 - b_2| = ||R_1| - 1 - b_1| + |B_{best} - |R_1| + 1 - b_2|$. This result indicate that the sensitivity of this multi-objectifs solver in this case of neighbouring depends on the size of regions taken as input, which makes it unbounded.