The Differential Privacy Implementation

Yuanshi Li yli903@uottawa.ca

Xinfei Wang xwang646@uottawa.ca

University of Ottawa

ABSTRACT

The protection of data privacy is essential in the digital age today, where data is collected and analyzed on a massive scale. Differential privacy techniques provide a formal framework for achieving data privacy by adding noise to the original data to create fake data that preserves the privacy of individuals. This article analyzes classic differential privacy protection mechanisms (such as Gaussian and Laplacian mechanisms) and applies them to the database using Java implementation, which is a practical way to protect sensitive personal data while enabling accurate data analysis. Our motivation is to demonstrate the effectiveness of differential privacy techniques in protecting data privacy and provide a practical privacy protection of personal data. Additionally, we evaluate each mechanism's functionality and data protection while comparing its running time, memory usage, and other characteristics. The project's usability, scalability, and potential for future work are all evaluated at the end. This project is currently open source on GitHub [1].

KEYWORDS

Differential Privacy, Privacy Security, Database, Java

1. Introduction

The advancement of technology has led to an enormous amount of data being generated, and its analysis has become an essential part of our daily lives. However, the collection and analysis of data raise concerns about the privacy and security of individuals. Data privacy is the fundamental right of individuals to keep their personal information confidential and protected from unauthorized access, use, and disclosure. Protecting data privacy is essential because it helps to safeguard individuals' personal information and prevent its misuse. Data can be used for various purposes, including marketing, research, and government surveillance. However, the misuse of data can lead to discrimination, identity theft, and other harmful consequences. In addition, the unauthorized access, use, or disclosure of personal information can breach an individual's privacy, which is a fundamental human right.

Differential privacy is a mathematical framework for protecting privacy when sensitive data is being analyzed. [2] It offers a precise definition of privacy as well as approaches for measuring the privacy protections offered by different data analysis algorithms. Differential privacy aims to permit sensitive data analysis while safeguarding the privacy of specific users. Its core definition of privacy is the inability to tell whether a dataset contains an individual's data or not. With this definition of privacy, algorithms can be developed that protect user privacy while offering valuable

insights into sensitive data. It ensures that the output of an algorithm is indistinguishable between two similar datasets, thereby protecting the privacy of the individuals in the dataset. [3] Differential privacy has emerged as a key element in data analysis that protects user privacy, since it gives computer scientists and data scientists a way to add noise to data in a controlled way, preventing individual records from being identified while still allowing the extraction of valuable insights from the data. By adding noise to the original data, it is difficult to identify any specific individual's data. The application of differential privacy techniques is becoming increasingly important as data collection and analysis continue to grow. The technique has been applied to several applications, including health care, finance, and government. In health care, the technique has been used to protect the privacy of patient data while enabling research to be conducted on sensitive medical data. In finance and retails, the technique has been used to preserve the privacy of customer data while allowing the analysis of the data to improve business processes. In government, differential privacy techniques have been used to protect the privacy of individuals in public data sets while enabling researchers to conduct analysis.

The primary motivation of this project is to apply differential privacy techniques to a database using Java to protect data privacy. We used the Differential Privacy open-source building libraries provided by Google, [4] and applied them applied them to our project according to their ideas of providing Gaussian noise [5] and Laplacian noise algorithms [6], aiming to implement Gaussian and Laplacian noise addition techniques to generate fake data that preserves the privacy of the data in MongoDB. We also implemented these noise algorithms to build methods to generate protected fake data with bounded mean, bounded quantiles, bounded variance, and standard deviation etc. The sensitivity of the dataset is determined by the amount and type of personal information it contains. The dataset's sensitivity means that it requires strong privacy protection to prevent unauthorized access, use, or disclosure of personal information. The contribution of our project is to demonstrate the effectiveness of differential privacy techniques in protecting data privacy and provide a practical implementation of these techniques. The dataset we used is a synthetic data created using Python, which contains sensitive information such as the daily and weekly income of an organization (we set up the organization as a restaurant, but technically it could be set up as any type of organization, such as supermarket or hospital), to make sure that they are protected from malicious attacker or competitor.

2. Methodology

The mathematical foundations of differential privacy rely on the concept of adding random noise to the data. The noise is added in such a way that the data remains useful for analysis while the privacy of individual users is protected. This noise is calibrated based on the privacy parameter, ε, which represents the privacy loss incurred by an individual in a given data analysis. ε-differential privacy is the most widely used privacy model, and it guarantees that the privacy loss of any individual record is at most ε. [3] The smaller the value of ε, the more privacy is preserved, but the more noise is added to the data, making it less useful for analysis. ε-differential privacy refers to a

privacy model in which the privacy of an individual is guaranteed to within a certain tolerance, represented by ε. In other words, the privacy loss incurred by an individual is bounded by ε. The ε-differential privacy model provides a flexible trade-off between privacy and utility, allowing for the adjustment of the privacy loss to meet the specific needs of a given data analysis.

Our main method for implementing differential privacy across multiple algorithms is to add noise to the system. This fundamental idea of this method is to introduce random noise to the data before performing any analysis. [7] Privacy is protected because it is difficult to deduce any sensitive information about a specific data point due to the noise. The fact that adding noise doesn't require any modifications to the underlying data or the analysis process makes it a flexible and simple method to use. The addition of noise includes the addition of Gaussian, Laplace, and exponential noise among other types. [6] The specific privacy requirements and desired level of utility determine the type of noise to be used. In ε-differential privacy models, a privacy parameter that determines the trade-off between privacy and utility can be used to regulate the amount of noise added.[3] There are three kinds of methods are used in our project for implementing differential privacy.

Then main method for adding noise to our data is Gaussian noise. [5] Each data point will have a random value added to it that will be drawn from a Gaussian distribution. The variance is determined by the privacy budget, and the distribution's mean is set to zero. The variance regulates the level of privacy offered by regulating the amount of noise added to the data. Gaussian noise has a number of benefits, such as mathematical tractability, compatibility with many statistical methods, and intuitive interpretation as the addition of random noise. It is simple to adjust the amount of noise added to the data based on the desired level of privacy, which is one of the main advantages of using Gaussian noise in differential privacy. By increasing the variance of the Gaussian distribution, for instance, more noise will be added to the data, strengthening privacy guarantees. On the other hand, lowering the variance will make the data more accurate because there will be less noise. Another benefit of using Gaussian noise is that it works well with many different statistical methods, such as clustering, regression, and hypothesis testing. The privacy of the data is maintained while still enabling accurate analysis and interpretation by the addition of Gaussian noise.

Laplace noise is another technique we used for adding noise. [6] Adding random noise to the data is the basic concept behind adding Laplace noise, and the amount of noise is dependent on the privacy budget and the sensitivity of the data. By first determining the data's sensitivity, the Laplace noise is then added to the data. This is the largest change that can be made to a function's output when its input changes by a single record. The size of the noise to be added is then decided using the privacy budget, a measurement of the amount of privacy loss that is acceptable in a specific scenario. The Laplace distribution, a continuous probability distribution distinguished by its symmetrical shape and weighted tails, is used to introduce noise into the data. Laplace noise has the benefit of being simple to calculate and use. Laplace noise can be added to the data with relatively straightforward calculations, in contrast to Gaussian noise,

which requires more intricate calculations. Furthermore, Laplace noise has a fixed variance, making it simpler to manage how much noise is added to the data. However, using Laplace noise has some drawbacks as well. The Laplace distribution's large tails can produce outliers in the data, which can skew the results. Additionally, the Laplace noise may be too large and affect the accuracy of the results, making it unsuitable for small datasets.

Comparable to Laplace noise, exponential noise was also used in our project. To introduce exponential noise, random noise from an exponential distribution is added to the sensitive data. [2] This makes it possible to obfuscate the data while still enabling accurate analysis and protecting it from privacy intrusions. A privacy budget, which is a gauge of how much privacy must be lost in a given situation, can be used to limit the amount of noise that is added. The exponential noise mechanism has the benefit of being memoryless, which means that previous calculations or data releases have no impact on how much noise is added. This may be helpful in situations where the privacy budget fluctuates. However, compared to Laplace noise, exponential noise may be more sensitive to the size of the data, which could be a problem in some circumstances. Laplace noise or exponential noise will therefore be chosen depending on the particular use case, the desired level of privacy, and the utility.

3. Project Structure

This project uses Google open-source library differential privacy [4] as the noise addition part. Based on existing algorithms, we extend them to database applications. And the calculation method of standard deviation has been improved. The supported algorithms are shown in Table 1.

TABLE 1 Supported algorithms

Algorithm	Language (Java/Kotlin)
Laplace mechanism	Supported
Gaussian mechanism	Supported
Count	Supported
Sum	Supported
Mean	Supported
Variance	Supported
Standard deviation	Supported
Quantile tree	Supported
Automatic bounds approximation	Supported
Truncated geometric thresholding	Supported
Laplace thresholding	Supported
Gaussian thresholding	Supported
Pre-thresholding	Supported

3.1 Algorithm Part Structure

Figure 1 is the class diagram of the algorithm. Noise is designed as an interface, and Gaussian [8], Laplace [9], and Discrete Laplace [10] noise are the specific implementations of Noise.

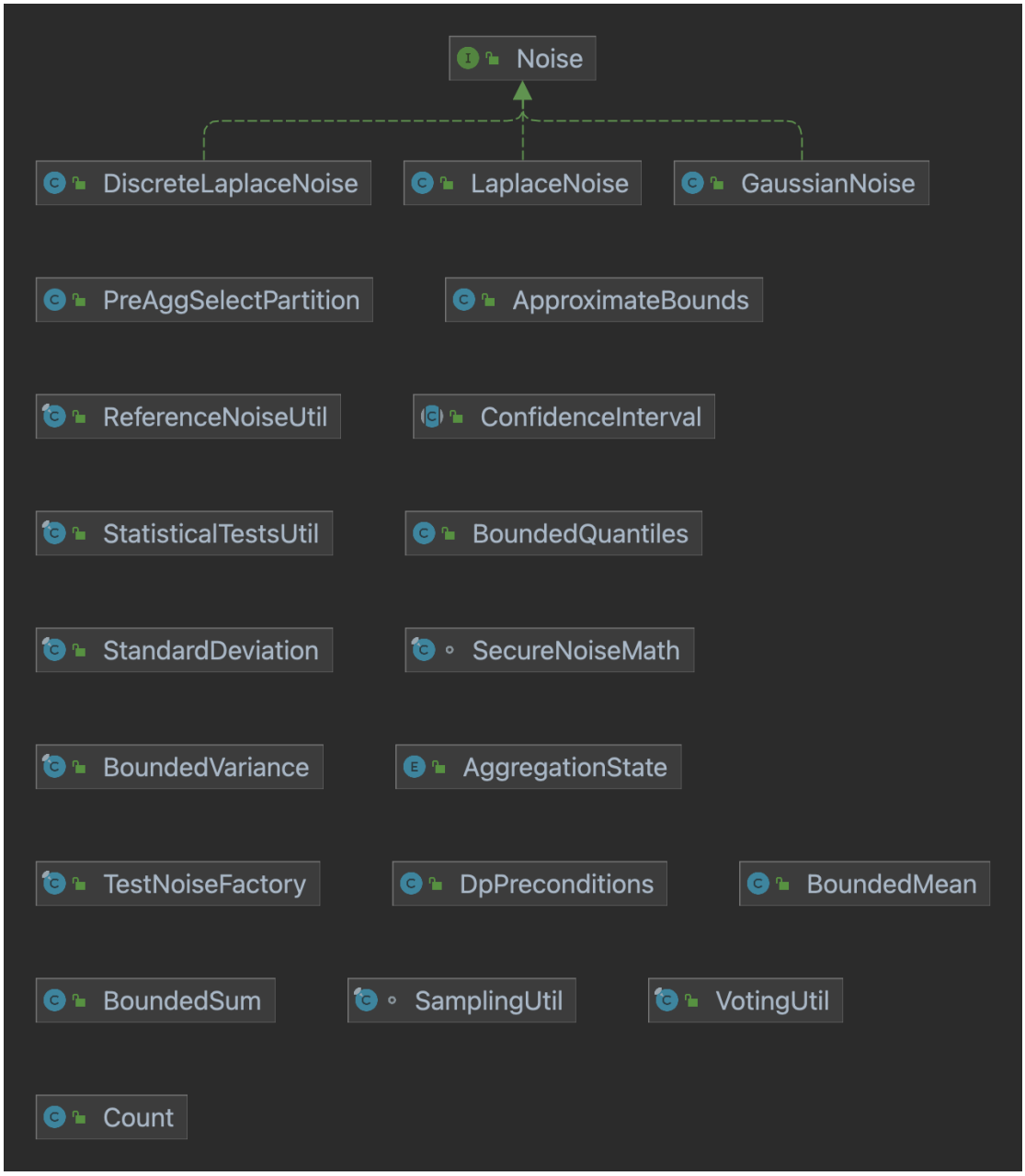

Fig 1. The class diagram of the algorithms

SamplingUtil is the function for sampling from several distribution. In the following chapter 4, we will discuss how to use this class to obtain the sample drawn from the geometric distribution of parameter in the Laplace and Discrete Laplace mechanism.

ConfidenceInterval class is to capture the scale of the noise they add to a metric during the anonymization process [11]. Given a noised metric M and a confidence level $1-\alpha$, the mechanisms return confidence intervals $[L, R]$ containing the raw metric m (where $\boldsymbol{m}$ is the value after contribution bounding, but before applying noise) with a probability of at least $1-\alpha$, i.e., Pr $[L \leq m \leq R] \geq 1-\alpha$.

A particular confidence interval is purely based on M and non-personal parameters of the respective mechanism, such as epsilon, delta, sensitivities and contribution bounds [12]. In particular, its computation does not access the raw metric m and

consequently it also does not consume any privacy budget.

The confidence intervals here do not account for discrepancy due to contribution bounding. Because the boundaries of statistical data are provided by the user, there is no need to calculate the boundaries of the data to improve performance.

ReferenceNoiseUtil, *StatisticalTestsUtil*, and *VotingUtils* classes contribute to unit tests [13]. The detailed implementation of other algorithms is demonstrated in chapter 4.

DpPreconditions class is designed as utilities class which validate the correctness of DP parameters. This class is used to check whether the number and type of algorithm input parameters are correct. For example, whether the correct δ is entered when using the Gaussian mechanism is in the range of $[0, 1]$.

To solve the differential privacy violation problem identified by Mironov [14], *SecureNoiseMath* is the mathematical utilities for generating secure differential privacy noise [15]. Under the provisions of IEEE, floating point numbers have an exponent and a significand. This works like scientific notation: the significand is multiplied by $2^{exponent}$ to get the represent value. Double-precision floating point numbers, which are assumed have 52 bits of significand and 11 bits of exponent; this means that they can represent the first 2^{52} multiples of any power of two between 2^{-1022} and 2^{1023} [16]. *SecureNoiseMath* class is to limit the numeric noise to the range of this secure field.

3.2 Application Part Structure

Figure 2 is the class diagram of the project application. The algorithm part was introduced into the project as an external library. This project can be applied to MongoDB and local CSV files. There are six software packages ***App***, ***Entity***, ***Exception***, ***Scenarios***, ***Schedule***, and ***Utils*** dedicated to handling different transactions. The project is built using Gradle automation, and the use of GradleWapper enables it to run even on machines without Gradle deployment.

DBApp is the only class in the package ***App***. This class operates the data in the MongoDB database using the *DBConnector* class in the package ***Utils***. There are two ways to manipulate data in the database, one is to use the Document and Bson [17, 18] data structures to directly use the built-in methods of MongoDB for differential privacy algorithm operations. The second method is to extract data from the database and save it as a static local file, such as using the CSVWriter method of opevcsv [19] to save data from databases such as MongoDB, and then applying the differential privacy algorithm. By testing the efficiency of these two methods, for the test data of this project (10000 and 40042, respectively), the second method has better efficiency in terms of memory usage and running speed (Figure 3 shows the running time under the same scenarios). Therefore, the second approach is the final choice in the project.

Visit and *VisitsForWeek* corresponds to the entities of the data. The package ***Entity*** is designed for the users to create their own data entities.

The package ***Exception*** is used to handle unexpected situations, such as database connection errors, non-existent collections in the MongoDB database, etc., to prevent program crashes.

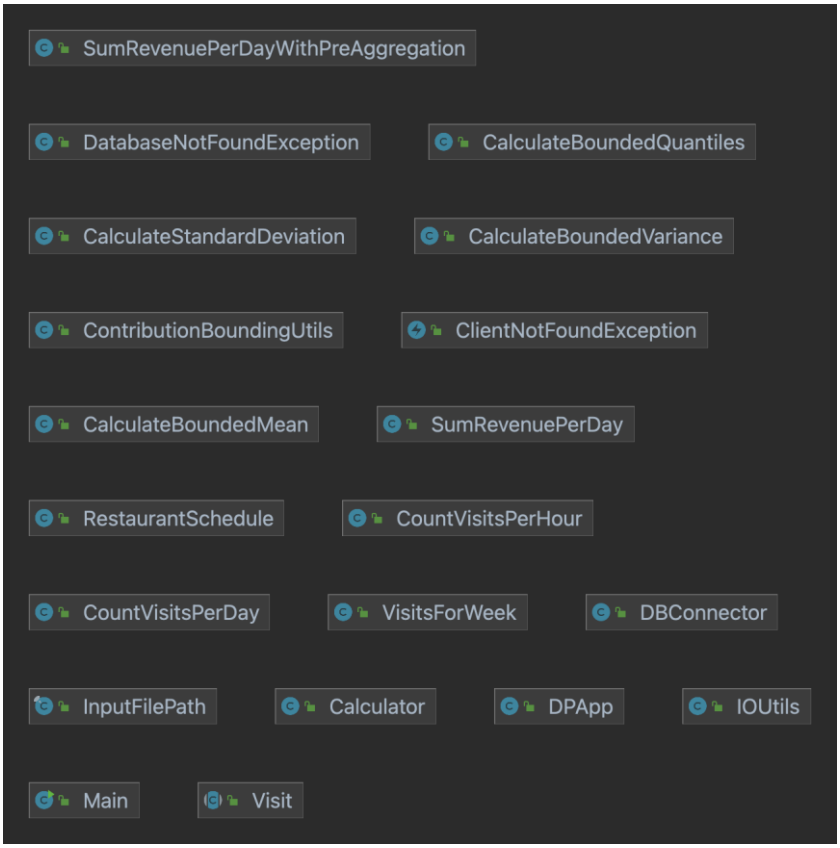

Fig 2. The class diagram of the application

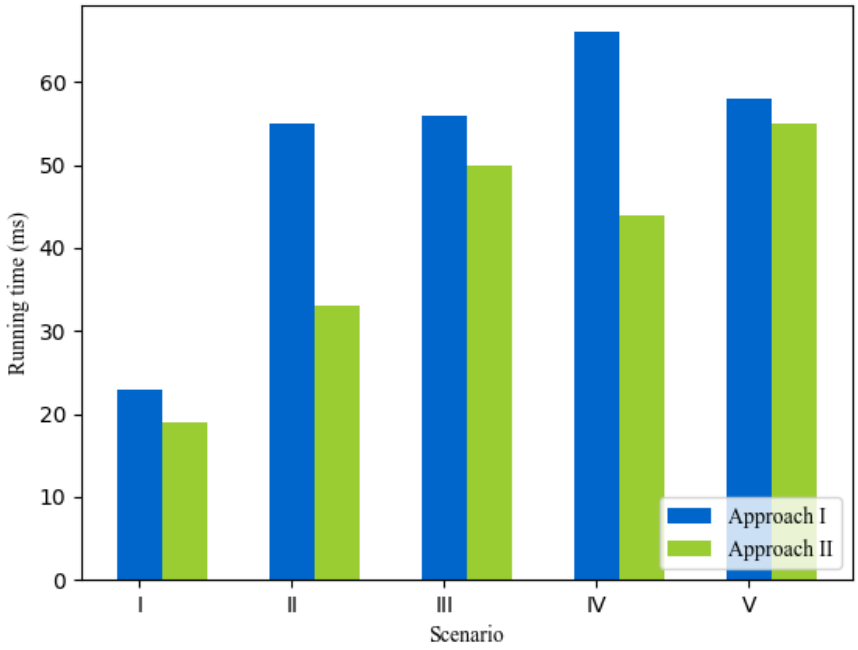

Fig 3. Running time under the same conditions of the two data processing approaches (I: CountVisitsPerHour, II: CountVisitsPerDay, III: BoundedMean, IV: SumRevenuePerDay, V: BoundedQuantiles)

The *RestaurantSchedule* class in Package ***Schedule*** is used to declare the

relationships in the data. In the test case, class *RestaurantSchedule* specifies the restaurant's operating hours from 9 A.M. to 10 P.M. o'clock and when customers can dine. Users can customize the constraints between different data types in this package. Package ***Scenarios*** is the specific implementation of the algorithms applied to the data. Users specify corresponding DP mechanisms based on different requirements. In class *Calculator* of package ***Utils***, "exposed" and "potentially attacked" operations are provided for comparison with query results that have been applied with differential privacy. The specific scenario and experimental analysis results will be explained in the following chapters.

File read and write, entity conversion, and some basic computing tools are included in package ***Utils***. Class ContributionBoundingUtils contains the restaurant visits where the number of days contributed by a single visitor is limited to the boundary which is given by users. Users can customize the boundary values of partition contributions in this class and apply them to various scenarios.

4. Implementation Details

The algorithm part is built using Bazel automated construction tool, packaged as a jar file and provided as an external library for project use. Junit 5 is used to perform unit testing on the methods of each class, and simulate specific usage scenarios using the *ReferenceNoiseUtil*, *StatisticalTestsUtil*, and *VotingUtils* classes mentioned earlier, followed by use case testing.

The application part adopts the popular Gradle automated construction tool, [20] and uses Aspect Oriented Programming (AOP) approach [21] to make the project structure clear, with strong scalability, making it convenient for users to customize data and create application scenarios.

Experiments are conducted in batches through scripts, and comprehensive comparisons are made in terms of algorithm selection, scenario operation, and operational performance.

4.1 Algorithm Details

Both class *LaplaceNoise* and *GaussianNoise* are specific implementations of interface *Noise*. Taking class *GaussianNoise* as an example, this article specifically introduces the implementation of noise.

Figure 4 shows the relationship between class *GaussianNoise* and interface *Noise*. This approach is robust against unintentional privacy leaks due to artifacts of floating-point arithmetic, because of the design of *SecureNoiseMath* classes.

The function *getSigma* returns the standard deviation of the Gaussian noise necessary to obtain $\{\varepsilon, \delta\}$-differential privacy for the given l2 sensitivity. The result will deviate from the tightest possible value *sigma_tight* by at most $GAUSSIAN\_SIGMA\_ACCURACY * sigma\_tight$. This implementation uses a binary search. Its runtime is rougly $\log(GAUSSIAN\_SIGMA\_ACCURACY) + \log\left(\max\left\{\frac{sigma\_tight}{l2\_sensitivity}, \frac{l2\_sensitivity}{sigma\_tight}\right\}\right)$. The calculation is based on Balle and Wang's research [22]. The paper states that the lower bound on sigma from the original analysis

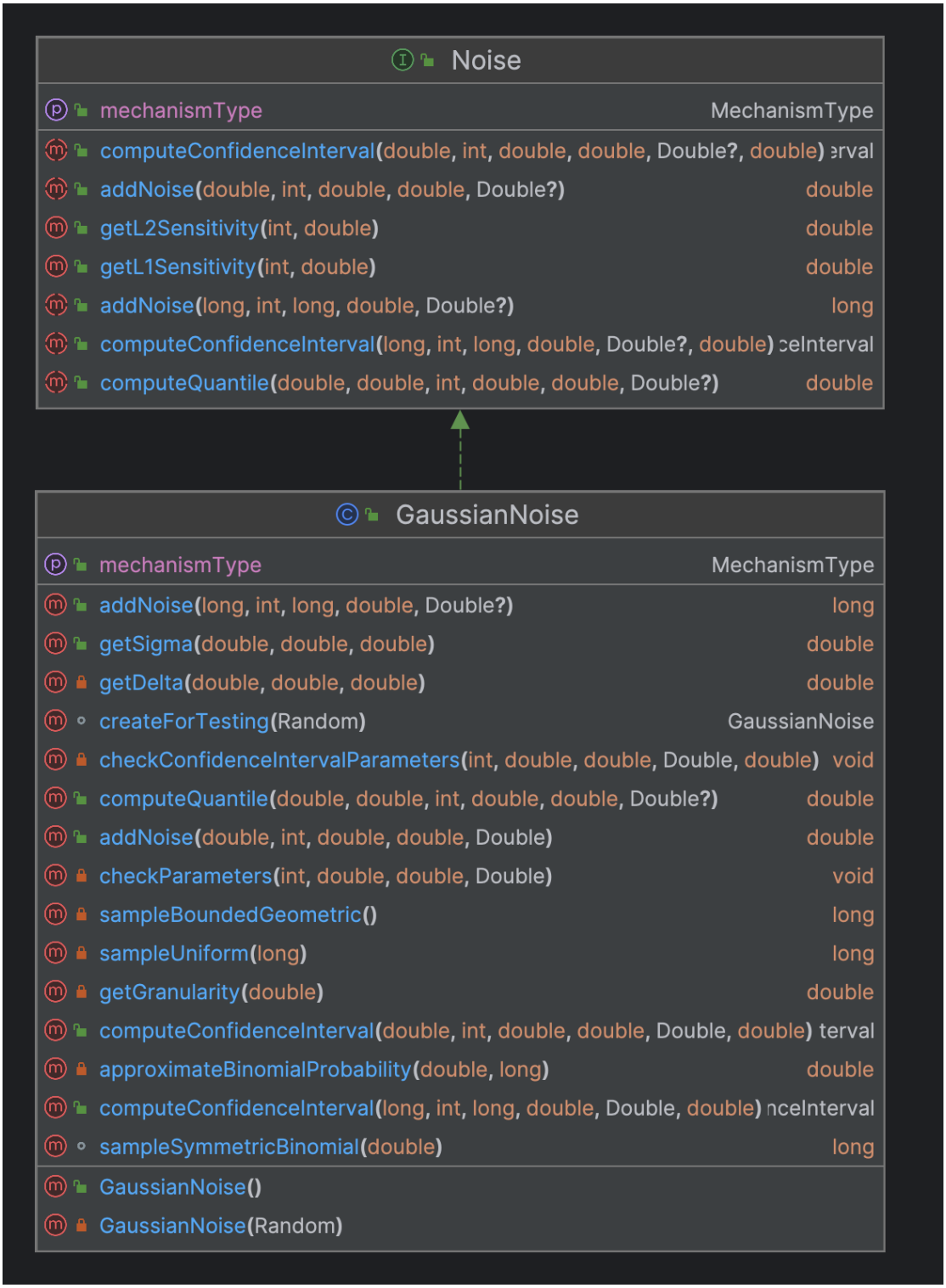

Fig 4. Class diagram of *GaussianNoise* and the relationship between interface *Noise*

of the Gaussian mechanism $\{\sigma \geq \sqrt{2 * l2\_sensitivity^2 * \log\left(\frac{1.25}{\delta}\right)/\varepsilon^2}\}$ is far from tight and binary search can give it a better lower bound.

The function *getDelta* returns the smallest delta such that the Gaussian mechanism with standard deviation obtains $\{\varepsilon, \delta\}$-differential privacy with respect to the provided *l2_sensitivity*.

The function *computeQuantile* is to computes the quantile z satisfying $\Pr[Y \leq z] = rank$ for a Gaussian random variable Y whose distribution is given by applying the Gaussian mechanism to the raw value x using the specified privacy parameters ε, δ, *l0_sensitivity*, and *lInf_sensitivity*.

The function *getGranularity* determines the granularity of the output of the function *addNoise* based on the σ of the Gaussian noise.

The function *sampleSymmetricBinomial* returns a random sample m where $m + n/2$ is drawn from a binomial distribution of n Bernoulli trials that have a success probability of $1/2$ each. The sampling technique is based on rejection sampling approach proposed by Bringmann et al [23]. The square root of n must be at least 10^6. This is to ensure an accurate approximation of a Gaussian distribution.

The function *sampleBoundedGeometric* returns a sample drawn from the geometric distribution with success probability $1/2$, i.e., the number of unsuccessful Bernoulli trials until the first success. The sample is capped should it exceed the geometric bound.

The function *sampleUniform* draws an integer greater or equal to 0 and strictly less than n uniformly at random. This custom implementation is necessary because the approach *java.security.SecureRandom* provides such functionality only for type *int* but not for the type *long*.

The function *approximateBinomialProbability* approximates the probability of a random sample $m + n/2$ drawn from a binomial distribution of n Bernoulli trials that have a success probability of $1/2$ each. The approximation is taken from Lemma 7 of the noise generation documentation from google [24].

Table 1 shows the currently supported algorithms, using class *BoundedVariance* as an example to illustrate the implementation of the algorithm and the points that need to be noted.

Class *BoundedVariance* calculates differentially private variance for a collection of values. Figure 5 is the class diagram of *BoundedVariance*.

The variance is a biased estimate and is computed as difference between the noisy variance of squares and square of the noisy variance. To improve utility, all entries are normalized by setting them to the difference between their actual value and the middle of the input range before summation [25]. Supports contributions from a single privacy unit to multiple partitions as well as multiple contributions from a single privacy unit to a given partition. The user can provide a *Noise* instance which will be used to generate the noise. If no instance is specified, *LaplaceNoise* is applied. The reason for using *LaplaceNoise* instead of *GaussianNoise* by default is that by comparing the results obtained from using two noise addition mechanisms on our data, *LaplaceNoise* has better protection, which will be discussed in subsequent chapters.

The function *addEntry* and *addEntries* clamp the input values and adds them to the variance.

The function *mergeWith* merges the output of *getSerializableSummary* from a different instance of *BoundedVariance* with this instance. Intended to be used in the context of distributed computation.

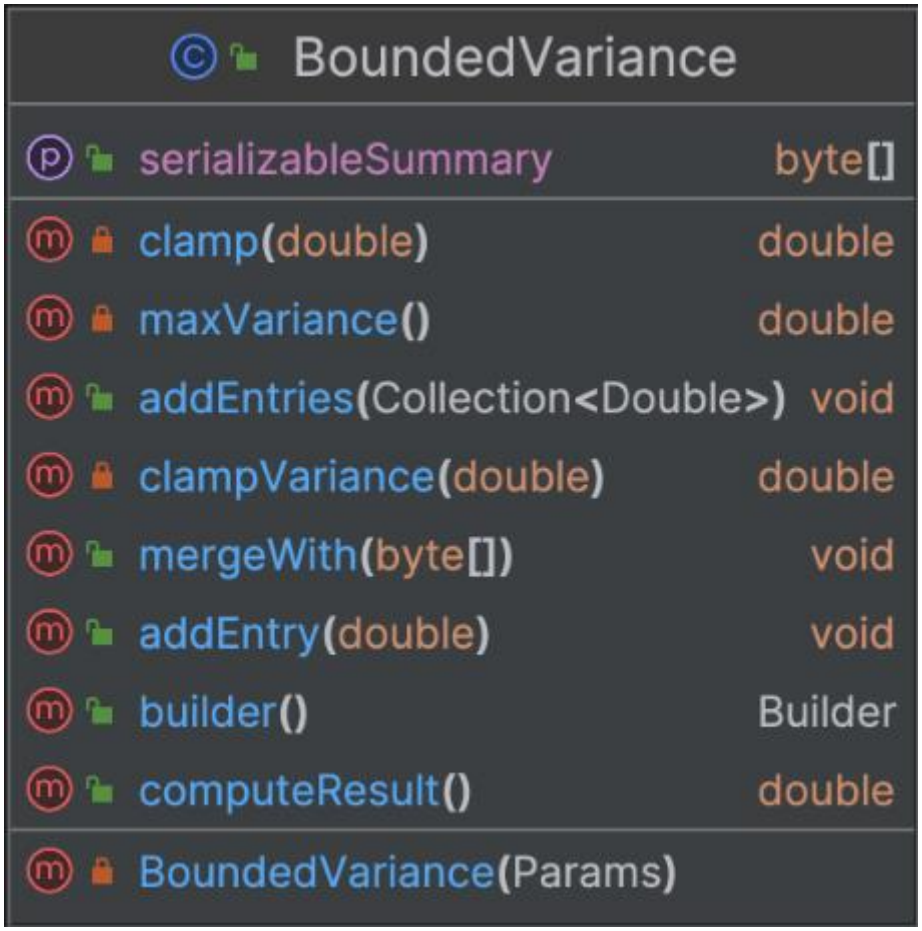

Fig 5. The class diagram with key methods of the *BoundedVariance*

The function *getSerializableSummary* returns a serializable summary of the current state of this *BoundedVariance* instance and its parameters. The summary can be used to merge this instance with another instance of *BoundedVariance*. This method cannot be invoked if the variance has already been queried, i.e., the function *computeResult* has been called. Moreover, after this instance of *BoundedVariance* has been serialized once, further modification and queries are not possible anymore.

By using Google's open API AutoValue [26] to automatically generate constructor *builder*. All parameters are included in *builder*.

The function *computeResult* calculates and returns differentially private variance of elements added using *addEntry* and *addEntries*. The method can be called only once for a given instance of this class. All subsequent calls will result in throwing an exception.

In addition to the above algorithms, Google has recently updated the *PreAggSelectPartition* method, which is used to calculate an $\{\varepsilon, \delta\}$-differential privacy decision of whether to materialize a partition.

Many differential privacy mechanisms work by performing an aggregation and adding noise. They achieve $\{\varepsilon\_m, \delta\_m\}$-differential privacy under the assumption that partitions are chosen in advance. In other words, they assume that even if no data is associated with a partition, noise is added to the empty aggregation, and the noisy result is materialized. However, when only partitions containing data are materialized, such

mechanisms fail to protect privacy for partitions containing data from a single privacy unit ID (e.g., user) [27]. To fix this, partitions with small numbers of privacy unit IDs must sometimes be dropped in order to maintain privacy. This process of partition selection incurs an additional $\{\varepsilon, \delta\}$-differential privacy budget resulting in a total differential privacy budget of $(\varepsilon + \varepsilon\_m, \delta + \delta\_m)$ being used for the aggregation with partition selection.

Depending on the *maxPartitionsContributed*, the *PreAggSelectPartition* uses one of two differentially private partition selection algorithms. The partition selection was given by Damien D, James V et al [28]. The selection is related to factors such as *maxPartitionsContributed* and *l0_sensitivity*.

4.2 Application Details

In addition to the file read and write, exception detection, and database connection modules mentioned earlier, the scenario module is a module that provides users with privacy protection. In this chapter, we focus on describing the implementation of the scenario section.

Figure 6 shows a specific scenario using the *BoundedQuantiles* algorithm, where the numerical value is found based on the ranking of customer expenses (For example, rank=0.5 means finding the median in the set of data).

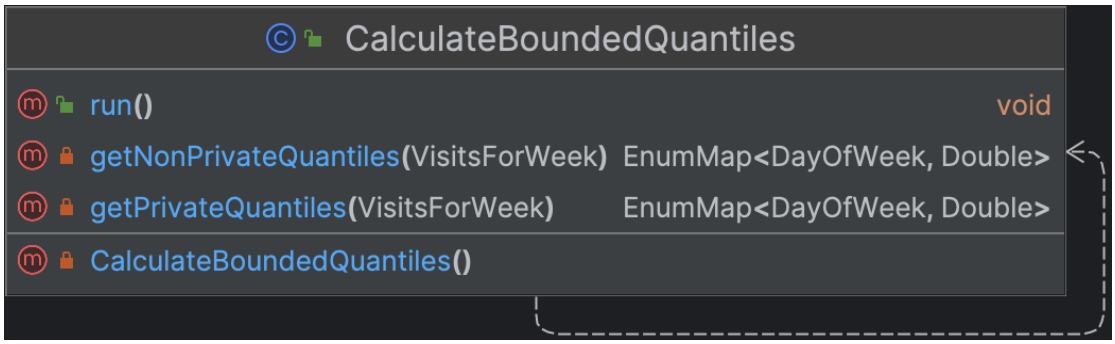

Fig 6. The class diagram of *CalculateBoundedQuantiles*

The function *getNonPrivateQuantiles* calculates the result without the use of differential privacy and these data are the ones we need to protect for privacy. Function *getPrivateQuantiles* uses the quantile tree mechanism for differential privacy protection to obtain the "confused" data. The obtained results are saved as static csv files by calling the file writing method in class *IOUtils*, and can then be imported into the database using MongoDB's import/export tool MongoDB Shell to achieve differential privacy protection.

The rest of the scene designs are modeled after this structure, which uses the class *Calculator* to calculate "real data" and then uses the differential privacy algorithm library to calculate the data protected for privacy. The experimental results will be analyzed in detail in the next chapter.

5. Experimental Results and Analysis

We mainly tested the algorithm on three aspects: computational results, application performance (such as the selection of file read and write methods mentioned in Figure 3, algorithm runtime), and protection performance (simulating differential privacy

attacks). In addition, the project has passed the unit testing of Junit 5 to ensure the fine-grained usability of the project.

5.1 Computational results

For the algorithm count, the result of counting the number of the customers of each hour shows in Figure 7.

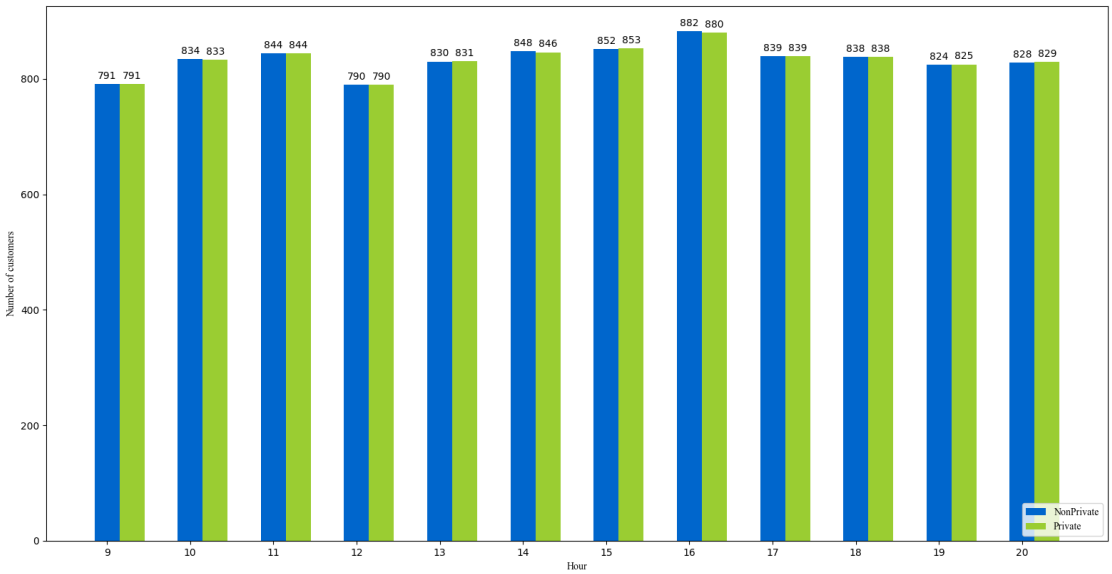

Fig 7. Result of counting algorithm for one day

Figure 8 shows the result of counting the number of visits in a week. Unlike the former scenario, the visitors can contribute to multiple partitions.

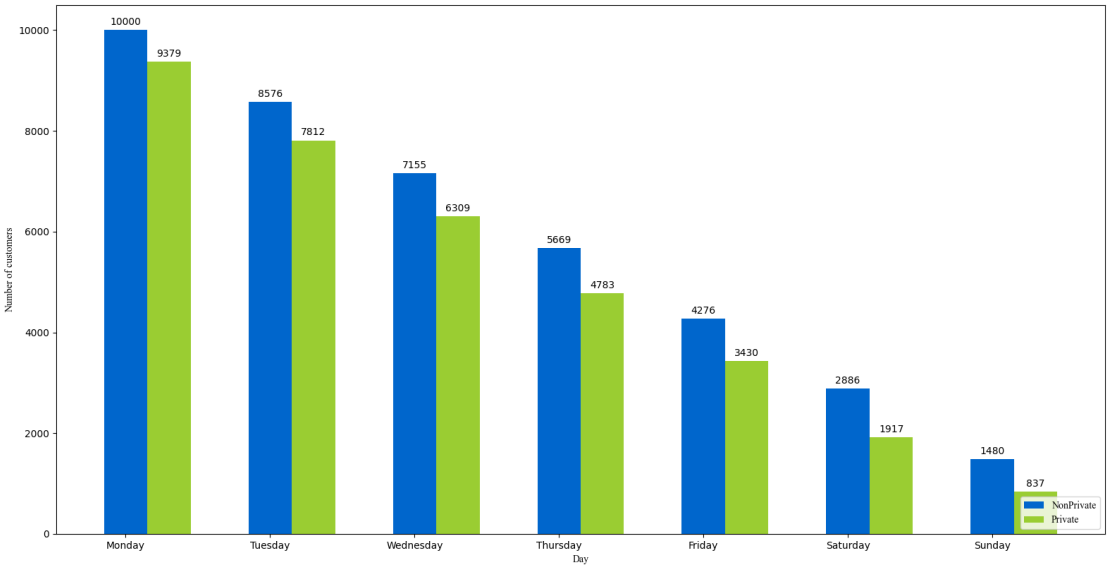

Fig 8. Result of counting algorithm for one week

Next scenario is calculating the sum-up revenue per day of the week. In this scenario, users can visit the restaurant multiple times and consume within each day of the week. For some aggregation (e.g., *BoundedMean*) support for multiple contributions per partition is as simple as configuring the *maxContributionsPerPartitions* parameter. However, if contributions have very different ranges, the library can add much more noise than necessary. For example, assume we calculate *BoundedSum* of customers' daily spendings. A customer typically

enters the restaurant twice a day and pays 10 euros for breakfast and 350 euros for dinner. The lower and upper bounds should be set to 10 and 350 while *maxContributionsPerPartitions* equals 7. If we let the library calculate the sensitivity of sum (i.e., how much a single customer can impact the result), this will result in sensitivity of 390 while the actual sensitivity is 350 (the higher the sensitivity is - the more noise is added). In such a case, it makes sense to pre-aggregate all visits, and provide a single contribution with adjusted lower and upper bounds (in our dataset, sum-up 10 and 350 and set the upper bound to 360). Figure 9 shows the result of this scenario.

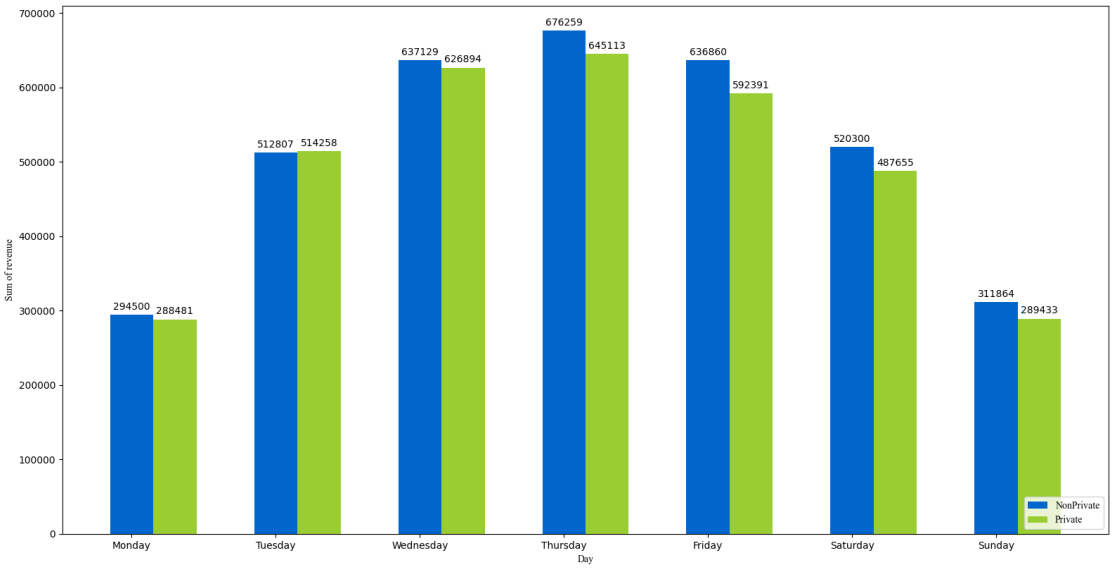

Fig 9. Result of the sum-up revenue per day of the week

Figure 10 shows the result of calculating the bounded standard deviation of the sum-up revenue. In this scenario, we set the *MAX_CONTRIBUTED_DAY* to 5 which means that the customer can visit the restaurant at most 5 times a week. We set *MAX_CONTRIBUTED_TIMES_PER_DAY* to 3, and this parameter means a customer can visit the restaurant at most 3 times one day.

The quantile tree mechanism is an important method for differential privacy protection of partitioned data especially in machine learning [29]. We set the rank to 0.2 which means that the result should be ranked 20% in the dataset. Figure 11 shows the result.

In this data, we used a trident tree with a tree height of 4 and a maximum number of branches of 3. The calculation results show a balance between protection and availability from the figure 11. We conducted multiple experiments by changing the height of the tree, the number of branches, and the size of the dataset. Firstly, the height of the quantile tree and the number of branches is related to the specific algorithm. After 100 experimental tests, we have obtained the locally optimal parameters for our dataset (which can be further refined through more experiments). Table 2 shows the relationship between tree height, number of branches, and specific algorithms.

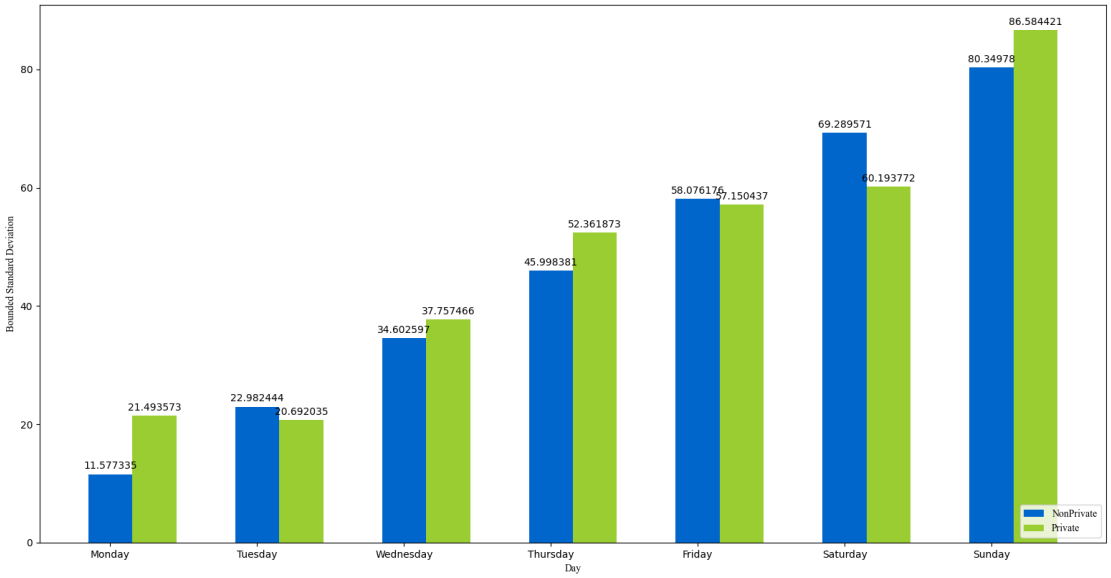

Fig 10. Result of calculating the bounded standard deviation

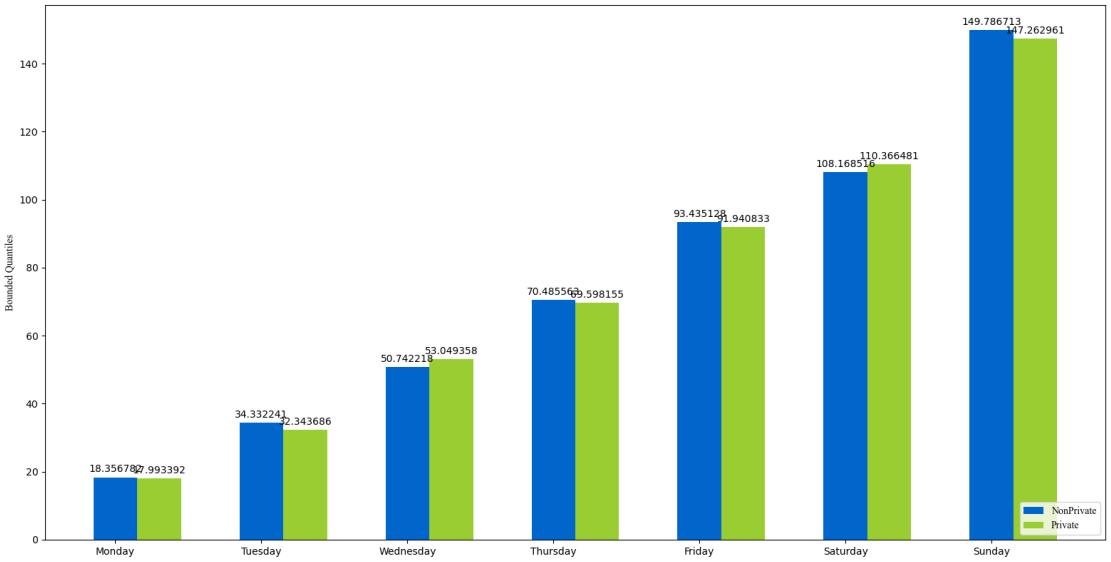

Fig 11. Result of calculating the bounded quantile

TABLE 2 Relationship between Number of branches, height, and algorithms

Algorithm	**Tree Height**	**Number of branches**
Count	2	32
Bounded Sum	4	6
Bounded Mean	4	6
Bounded Variance	3	8
Standard Deviation	2	32

5.2 Application performance

The testing device is the machine equipped with Apple Silicon M1 chip and MacOS12 operating system. Figure 12 shows the running time of different algorithms using

Laplace and Gaussian mechanisms.

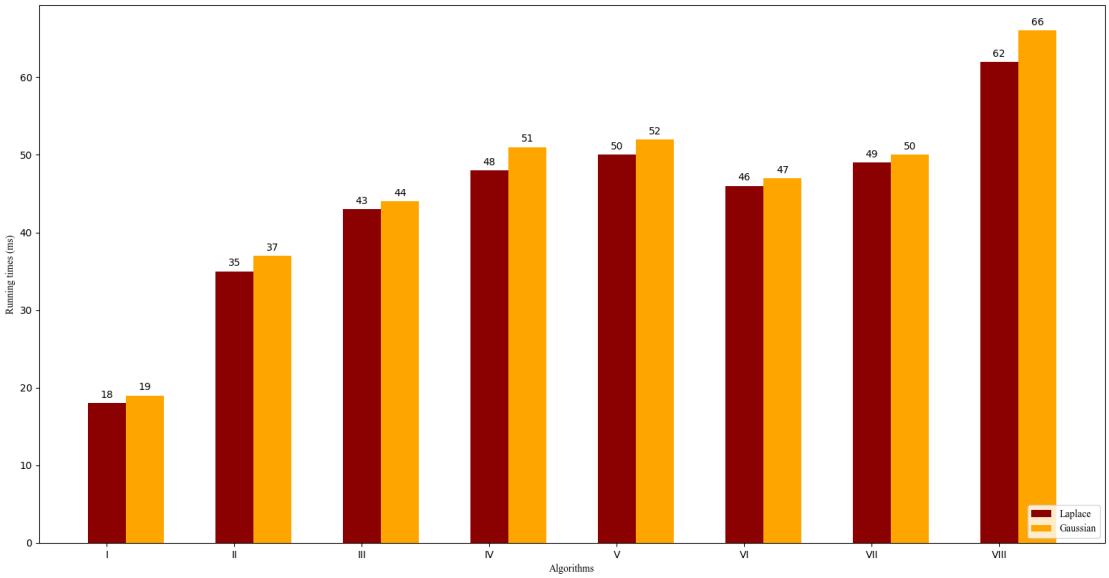

Fig 11. Under different noise addition mechanisms, the running time of 8 algorithms (I: COUNT_VISITS_PER_HOUR, II: COUNT_VISITS_PER_DAY, III: SUM_REVENUE_PER_DAY, IV: SUM_REVENUE_PER_DAY_WITH_PREAGGREGATION, V: CAL_VARIANCE_PER_DAY, VI: CAL_MEAN_PER_DAY, VII: CAL_STANDARD_DEVIATION, VIII: CAL_BOUNDED_QUANTILES)

The running time is the average value obtained by running 100 times. In computational methods, Gaussian noise is indeed more complex than the Laplacian noise mechanism [30], therefore requiring longer computational time. This result is in line with theoretical expectations. As the complexity of the algorithm increases, the running time also gradually increases. For the quantile tree algorithm, we also tested the running time under different tree heights and branch numbers and found that the running time increases with the increase of these indicators. Therefore, we can conclude that under the optimal parameters, the running time of the quantile tree algorithm is directly proportional to the data size, tree height, and number of branches.

5.3 Protection performance

We simulate potential attacks in real situations by querying private data. We conducted 10000 attacks on differential privacy data by randomly simulating algorithms such as Count and Sum. Figure 12 shows the statistical results of the success and failure rates of attacks. The results show that the success rate of the attack fluctuates within the range of $[0.150\%, 1.001\%]$. The arithmetic mean value of attack success rate is 0.518% which means that approximately one out of every two hundred attacks may succeed. Therefore, in terms of protection performance, this application performs better. In terms of file reading and writing, algorithm computing performance, and other aspects, the application has a good balance between availability and protection.

Fig 12. Attack success and failure ratio pie chart
(10000 attacks, the scenario corresponds to Fig 11.)

6. Future Work

When the underlying data is itself discrete (e.g., population counts), adding continuous noise makes the result less interpretable [31]. Discrete Laplace and Gaussian mechanism can be introduced in the future work to solve such specific problems.

At present, the project only provides Java/Kotlin interface, and only provides MongoDB database interface. Future work can provide interfaces for more relational database or NoSQL database to improve availability and scalability.

The current algorithm does not consider thread safety, which means there may be conflicts such as WW (write-write), WR (write-read), and RW (read-write) [32]. In the future, algorithms and application program optimization can be optimized into thread safe applications. Of course, the cost of doing so may be sacrificing a portion of operational efficiency (such as increasing running time, resulting in increased computational costs). Finding a balance between the two is also one of the directions for future work.

REFERENCE

[1] https://github.com/DGHSRailgun/DifferentialPrivacy-Java

[2] Dwork, C. (2006). Differential Privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds) Automata, Languages and Programming. ICALP 2006. Lecture Notes in Computer Science, vol 4052. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11787006_1

[3] Dwork, C., & Roth, A. (2014). The Algorithmic Foundations of Differential Privacy, Foundations and Trends® in Theoretical Computer Science: Vol. 9: No. 3–4, pp 211-407.

http://dx.doi.org/10.1561/0400000042
[4] https://github.com/google/differential-privacy
[5] F. Liu, "Generalized Gaussian Mechanism for Differential Privacy," in IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 4, pp. 747-756, 1 April 2019, doi: 10.1109/TKDE.2018.2845388.
[6] Q. Geng, P. Kairouz, S. Oh and P. Viswanath, "The Staircase Mechanism in Differential Privacy," in IEEE Journal of Selected Topics in Signal Processing, vol. 9, no. 7, pp. 1176-1184, Oct. 2015, doi: 10.1109/JSTSP.2015.2425831.
[7] Chaudhuri, K., & Monteleoni, C. (2011). Differential privacy: An economics approach. In Proceedings of the 3rd Conference on Innovations in Theoretical Computer Science (pp. 1-10). ACM.
[8] Dong J, Roth A, Su W J. Gaussian differential privacy[J]. Journal of the Royal Statistical Society Series B: Statistical Methodology, 2022, 84(1): 3-37.
[9] Phan N H, Wu X, Hu H, et al. Adaptive laplace mechanism: Differential privacy preservation in deep learning[C]//2017 IEEE international conference on data mining (ICDM). IEEE, 2017: 385-394.
[10] Fernandes N, McIver A, Morgan C. The Laplace Mechanism has optimal utility for differential privacy over continuous queries[C]//2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). IEEE, 2021: 1-12.
[11] Covington C, He X, Honaker J, et al. Unbiased statistical estimation and valid confidence intervals under differential privacy[J]. arXiv preprint arXiv:2110.14465, 2021.
[12] Hosmer D W, Lemeshow S. Confidence interval estimation of interaction[J]. Epidemiology, 1992: 452-456.
[13] https://github.com/google/differential-privacy/tree/main/java/main/com/google/privacy/differentialprivacy/testing
[14] Mironov I. On significance of the least significant bits for differential privacy[C]//Proceedings of the 2012 ACM conference on Computer and communications security. 2012: 650-661.
[15] Goldberg D. What every computer scientist should know about floating-point arithmetic[J]. ACM computing surveys (CSUR), 1991, 23(1): 5-48.
[16] https://en.wikipedia.org/wiki/Double-precision_floating-point_format
[17] https://en.wikipedia.org/wiki/MongoDB
[18] https://www.mongodb.com/docs/drivers/java-drivers/
[19] https://opencsv.sourceforge.net/
[20] Muschko B. Gradle in action[M]. Simon and Schuster, 2014.
[21] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-oriented programming[C]//ECOOP'97—Object-Oriented Programming: 11th European Conference Jyväskylä, Finland, June 9–13, 1997 Proceedings 11. Springer Berlin Heidelberg, 1997: 220-242.
[22] Balle B, Wang Y X. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising[C]//International Conference on Machine Learning. PMLR, 2018: 394-403.
[23] Bringmann K, Kuhn F, Panagiotou K, et al. Internal DLA: Efficient simulation of a physical growth model[C]//Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41. Springer Berlin Heidelberg, 2014: 247-258.

[24]https://github.com/google/differential-privacy/blob/main/common_docs/Secure_Noise_Generation.pdf
[25] Markowitz H. Mean–variance approximations to expected utility[J]. European Journal of Operational Research, 2014, 234(2): 346-355.
[26] https://github.com/google/auto/blob/main/value/userguide/index.md
[27] Ji Z, Lipton Z C, Elkan C. Differential privacy and machine learning: a survey and review[J]. arXiv preprint arXiv:1412.7584, 2014.
[28] Desfontaines D, Voss J, Gipson B, et al. Differentially private partition selection[J]. arXiv preprint arXiv:2006.03684, 2020.
[29] Gillenwater J, Joseph M, Kulesza A. Differentially private quantiles[C]//International Conference on Machine Learning. PMLR, 2021: 3713-3722.
[30] Dadi M I, Marks R J. Detector relative efficiencies in the presence of Laplace noise[J]. IEEE transactions on aerospace and electronic systems, 1987 (4): 568-582.
[31] Canonne C L, Kamath G, Steinke T. The discrete gaussian for differential privacy[J]. Advances in Neural Information Processing Systems, 2020, 33: 15676-15688.
[32] Stearns R E, Rosenkrantz D J. Distributed database concurrency controls using before-values[C]//Proceedings of the 1981 ACM SIGMOD international conference on Management of data. 1981: 74-83.