Compute Engine Testing with Privacy-Compliant Production-Like Synthetic Data

Eric Liu, Jiangnan Cheng, Steve Chuck, Lyublena Antova, Yurgis Baykshtis, Matt David, Ge Gao, Mehrdad Honarkhah, Kuan-Sung Huang, Chen-Kuei Lee, Usman Muhammad, Shihao Peng, Andrii Rosa, Rebecca Schlussel, Michael Shang, Kelvin Silva, Brandon Vo, Zac Wen, Yihao Zhou
Meta Platforms Inc.
USA
{ericyuliu,jncheng}@meta.com

ABSTRACT

Compute engines require thorough and continuous testing where synthetic benchmarks or hand-crafted queries and data sets are not enough, yet accessing customer data and queries either is impossible or brings a big burden of user data protection. At Meta, we build SynQB - a query bank for compute engine testing that leverages privacy-compliant production-like synthetic data with good data quality, generated by a carefully designed differentially-private synthetic data generation (DPSDG) algorithm. This solution solves our common pain points in testing by enabling test coverages for different compute engines, query operators and workloads and facilitating various correctness and performance regression detections. As a result, SynQB greatly improves our compute engine release confidence and enhances the reliability of our data warehouse.

VLDB Workshop Reference Format:
Eric Liu, Jiangnan Cheng, Steve Chuck, Lyublena Antova, Yurgis Baykshtis, Matt David, Ge Gao, Mehrdad Honarkhah, Kuan-Sung Huang, Chen-Kuei Lee, Usman Muhammad, Shihao Peng, Andrii Rosa, Rebecca Schlussel, Michael Shang, Kelvin Silva, Brandon Vo, Zac Wen, Yihao Zhou. Compute Engine Testing with Privacy-Compliant Production-Like Synthetic Data. VLDB 2024 Workshop: International Workshop on Quality in Databases (QDB'24).

1 INTRODUCTION

Compute engines are critical in data-driven organizations, especially when large amounts of data need to be processed to generate insights and support decision-making. As such, they require thorough and continuous testing to ensure correctness, performance and reliability. As in any other domain, simulating realistic workloads during testing helps with the proactive discovery of bugs and inefficiencies before they hit the customer. A big challenge when testing compute engines is that database vendors often do not have access to customer data and queries, or are not allowed to make copies of these outside of the customers environment. To address this challenge, vendors have historically relied on one or both of the following: synthetic benchmarks or hand-crafted queries and data sets to emulate a customer environment. There is a variety of synthetic query and data generators attempting to create a true representation of real customer workloads, including TPC-H and TPC-DS among others. Those however are only a start, as they fail to represent realistic customer workloads. Another approach taken by database vendors is to sample customer queries and workloads and use these for testing. While this provides a much more realistic test bed and removes many of the limitations of testing with synthetic benchmarks, it has limited applicability as it requires customers to approve using their data and queries in a test environment, and puts a big burden on the database vendor to ensure customer data is protected. At Meta, preserving user data privacy is critical and tables are subject to strict restrictions on user access, as well as data retention policies. This presents a challenge to testing with real workloads.

We built SynQB to solve the challenge of testing realistic workloads in a privacy compliant way. SynQB is a query bank framework for compute engine testing that leverages synthetic data generation (SDG) that provides production-like synthetic data for testing. SDG is an emerging technique that generates simulated data with similar statistical properties and characteristics of the real data while protecting the sensitive information of each individual user. It was noted by the National Science and Technology Council a "key technical approach for privacy-preserving data sharing and analytics" [12]. In particular, our designed SDG algorithm is protected by differential privacy (DP) [4, 5], a state-of-the-art privacy standard that guarantees the identification of a specific user is technically impossible. The production-like and privacy-compliant synthetic data are of high data quality, solving our common pain points in testing by enabling test coverage for different compute engines, query operators and workloads and facilitating detection of correctness and performance regressions. As a result, SynQB greatly improves our compute engine release confidence and enhances the reliability of our data warehouse.

In light of prior work, our contributions are as follows. First, we design and implement the system architecture of SynQB, which includes a creation workflow where production queries are rewritten, production-like privacy-compliant synthetic tables are generated, and a usage suite is generated which incorporates various correctness and performance regression detections. Second, we design a generic differentially-private synthetic data generation (DPSDG) algorithm for compute engine testing, which handles user-identifiable information (UII) and user features differently, and can also be parallelized across multiple machines and CPUs. Third,

Proceedings of the VLDB Endowment. ISSN 2150-8097.

through evaluations we validate that SynQB generates high-quality synthetic data that provide good signals to compute engine testing.

2 RELATED WORK

2.1 Compute Engines at Meta

Presto [16] is a distributed SQL query engine used for low-latency interactive workloads as well as long-running ETL pipelines involving exabyte-scale data volumes at Meta. It is widely used by teams and systems across the company for diverse analytical use cases such as business intelligence, machine learning, feature engineering, experimentation platforms etc.

In recent years, Meta has successfully implemented several Presto evolutions like Presto-on-Velox and Presto-on-Spark, which can be seen as extensions of the classic Presto engine. These engines integrate Presto with other big data processing technologies: Apache Spark in the case of Presto-on-Spark, and Velox vectorized execution engine [13] for Presto-on-Velox to achieve enhanced query performance, reliability and scalability for specific data analytics workloads. One noteworthy aspect is that these engines all share the same Presto SQL dialect syntax and semantics. The unified language interface, in addition to offering a consistent user experience and reducing user friction, has helped improve the development of SynQB. This is achieved by promoting query bank reusability and seamless cross-engine verification as queries designed for Presto can easily be adapted to run on other engines sharing the same interface.

In addition to Presto, we have also integrated SynQB into other compute engines, such as Spark, by leveraging shared components. This approach allows us to seamlessly upgrade to newer versions of Apache Spark while ensuring that both the production version and the new version produce consistent results without any correctness or performance regressions. We are committed to expanding our coverage and aim to consolidate our testing solutions, with the goal of onboarding SynQB across all compute engines within Meta's data warehouse.

2.2 Compute Engine Testing

Presto Verifier[1] is an open-source testing tool used for executing queries and ensuring their correctness by comparing the results with a control run. Typically, the control side represents the stable production environment while the test side represents the release candidate environment. This process involves checking the consistency of the checksums generated by both query results: any discrepancies with deterministic queries indicate correctness regressions. However, a potential drawback of utilizing Presto Verifier is the production data source may change overtime or even get removed entirely, resulting in unreliable and low-signal test results.

SnowTrail [21] is Snowflake's testing framework that leverages its time travel capability to execute queries at predefined timestamps, thereby ensuring data consistency. However, this framework faces challenges due to constantly evolving schema or data changes that may render original queries ineffective overtime.

SynQB addresses this concern by employing reliable, production-like, and privacy-compliant synthetic data, thereby minimizing data variability and providing a more robust foundation for effective and high signal testing, encompassing both correctness and performance regression detections. This method offers a cleaner solution compared to the previous approaches. Besides, SynQB offers advantages by not negatively affecting the customers' production traffic in two key aspects: 1) By minimizing the necessity for control side runs on the production cluster; 2) By limiting the requirement for both control and test sides to utilize customer production data.

2.3 Synthetic Benchmarks

There is a long history of synthetic benchmarks for database testing with a large focus on performance testing. The Transaction Processing Council (TPC)[2] maintains a number of benchmarks for various domains, including TPC-C, TPC-H, TPC-DS, and the more recent TPCx-AI [3]. In the data analytics space TPC-H and TPC-DS are the main benchmarks. They collectively feature a little over 120 queries and provide a data generator to produce sample data for testing. While these are often a good start, synthetic generators fall short in modeling true customer workloads. For example, neither TPC-H or TPC-DS support complex data types such as arrays or maps, or test for operations on these. In addition, the research community has contributed a number of proposals for synthetic generators, including schema- and query-aware generators [6, 9, 14, 15], but these proposals do not necessarily have privacy in mind.

2.4 DPSDG

There are a wide variety of works designing SDG algorithms with good utility under DP guarantee. Based on the underlying model, they can be split into two main categories: 1) **Marginal-based models** (such as AIM [10] , MWEM-PGM [11] and DP Gaussian Copula Kendall [8]) measure the marginal distributions of the input dataset with DP guarantee, and then generate synthetic data that comply with the marginal distributions; 2) **GAN-based models** (such as DPGAN [20] , DP-CGAN [18] and PATE-GAN [7]) use differentially-private stochastic gradient descent (DPSGD) method [1] to train a generative adversarial net (GAN), which consists of a generator for synthetic data. In most of these works, the utility metrics of synthetic data were defined as marginal distribution, pairwise correlation and the accuracy of machine learning inference of the synthetic data, etc. A comprehensive comparison for these metrics is conducted in [17].

To the best of our knowledge, this paper is the first work to consider the quality of synthetic data in the context of compute engine testing, which is defined as synthetic data's ability to accurately reflect the performance and behavior of compute engines compared to production data. Key aspects include 1) **Query performance**: Testing with synthetic data should replicate the performance metrics of testing with production data, such as query execution time and resource utilization. In particular, the accurate representation of flexible and nested data types, such as maps and structs, is crucial in machine learning and advanced analytics, as modern data systems often use sophisticated data structures to encapsulate rich, multi-dimensional information. 2) **Engine feature testing**: Testing with synthetic data should activate specific features of compute engines and include edge cases, thereby providing comprehensive testing

[1] https://prestodb.io/docs/current/admin/verifier.html

[2] https://www.tpc.org/

coverage. 3) **Reproducibility of results**: Testing with synthetic data should produce consistent results and performance metrics across various environments and configurations.

3 SYSTEM ARCHITECTURE

In this chapter we first introduce the creation workflow of SynQB in Sec. 3.1 and then the corresponding usage suite in Sec. 3.2. The advantages of SynQB are highlighted in Sec. 3.3.

3.1 SynQB Creation Workflow

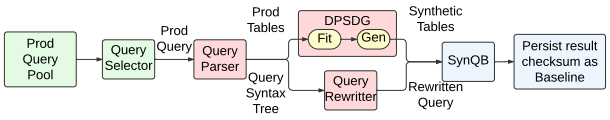


Figure 1: SynQB creation workflow. Queries are selected from production workload and rewritten with generated privacy-compliant and production-like tables.

The SynQB creation workflow, as illustrated in Fig. 1, encompasses the following three steps.

Query selection. The source of SynQB is a production query pool. We adopt a query selector with various selection criteria based on the coverage requirements. There are two types of coverage we consider: 1) **Generic code coverage**, which further encompasses several major dimensions, such as coverage of different Presto operators including TableScan, Join, Aggregation, TableWriter, etc. and coverage of different data structures including map, array, struct, JSON, etc.; 2) **Important workloads coverage**, such as company-wide critical dashboards, ads experiments workloads, etc.

Query rewriting and synthetic table generation. A selected query will then be parsed by a query parser, which extracts the production table names and produces the corresponding query syntax tree. For each production table, we leverage a DPSDG algorithm to generate a production-like and privacy-compliant synthetic table. The algorithm takes two steps: 1) a *fit* step that trains a compressed model M from the input production table D with (ϵ, δ)-DP (defined in Sec. 4.1) guarantee, denoted as

$$M = \texttt{DPSDG-Fit}(D; \epsilon, \delta), \tag{1}$$

and 2) a *generate* step that produces a synthetic table D_{syn} with N_{syn} number of rows according to the model M, denoted as

$$D_{\text{syn}} = \texttt{DPSDG-Gen}(M, N_{\text{syn}}). \tag{2}$$

The algorithm behind these two steps will be discussed in details in Sec. 4.2. Such two-step process enables parallelized generation through vertical scaling and horizontal scaling. After synthetic table generation, the query will also be rewritten with the original query syntax tree and the new synthetic table names.

Query and synthetic table persistence. The rewritten query and the synthetic tables will be added to the SynQB. The query will also be run with a stable version of Presto, and then the checksum of the output will be persisted as the baseline, which can be used in the future test runs for correctness regression detection.

3.2 SynQB Usage Suite

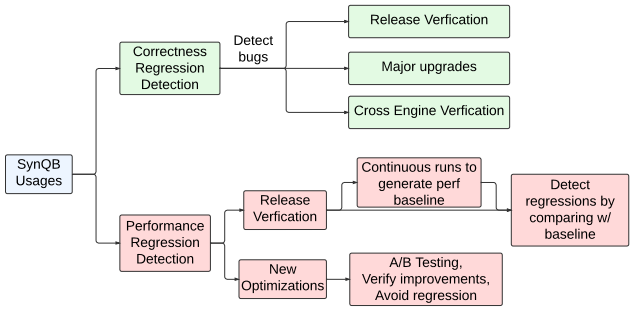


Figure 2: SynQB usage suite. SynQB can be used to detect various kinds of regressions.

As shown in Fig. 2, SynQB is mainly used to detect regressions, including correctness and performance regressions.

Correctness regression detection. For compute engines, the most important requirement is to guarantee that results are correct. Presto, Presto-On-Velox, and Presto-On-Spark all share the same SQL interface, meaning that the same query is expected to work on these engines interchangeably. Therefore, we use the same test query to not only verify continuous releases for each compute engine itself, but also cross check each other. For example, while developing Presto-on-Velox, we compare its query results with baseline result generated by stable Java-based Presto, which has not only exposed correctness issues but also has helped identify missing functionality in the new engine.

Performance regression detection. Queries and tables created for correctness regression detection can also be used for performance regression detection. By using immutable synthetic data and a consistent testing environment, we gather performance results that provide stronger signals than traditional shadow testing solutions and are more representative of our workload than industry benchmarks. The key strategy here is to establish a consistent and trustworthy baseline, which facilitates effective and meaningful regression detection analysis. To accomplish this, we run tests multiple times per day across releases, recording key metrics for each query, such as CPU usage, memory usage, and execution time. Based on this high signal setup, for major upgrades and optimization, we also conduct A/B testing to measure the improvements and avoid accidental regressions. This approach allows us to compare the performance of different versions under controlled conditions, ensuring that enhancements deliver the expected benefits.

3.3 Highlights

We highlight some advantages of our SynQB as follows.

Sustainable testing with synthetic data. To ensure production-like quality and immutability for sustainable testing purposes, we generate privacy-compliant synthetic data. SynQB transforms production queries to use these immutable test datasets as inputs. This approach addresses the issue of unreliable data sources from the ground up, enabling us to develop a sustainable and high-signal test framework.

Early signals. From daily testing results, we have proactively prevented correctness regressions from slipping into production, effectively averting potential data corruptions. Furthermore, SynQB offers expedited feedback compared to conventional production shadow testing solutions. This efficiency is achieved through the manipulation of synthetic test data sizes. We use smaller data sets for correctness testing to decreases the runtime of end-to-end testing. Additionally, the short execution time enables the integration of SynQB into the pull-request landing process. This enhancement can accelerate the testing cycle and facilitate the early detection and resolution of bugs. Developers can receive immediate feedback on their changes, promoting a more efficient and iterative development process.

Reliable performance regression A/B testing framework. The use of synthetic data fundamentally reduces variability at the data layer, thereby providing a more robust foundation for reliable testing. Our performance regression detection framework is particularly relevant for projects that involve significant changes or upgrades, such as the introduction of new file formats in data warehouses, SQL query optimizations, and major library upgrades. By using this approach, we ensure that each enhancement not only meets the intended functionality improvements but also maintains or enhances overall system performance.

Historical failed queries. We systematically collect queries that had production regressions and incorporate them into a historical-failed queries test suite. This suite is then used for verifying future releases, ensuring that previously encountered issues are adequately addressed and do not recur.

Flexible and pluggable testing framework. SynQB is highly extensible, facilitating integration with each compute engine's test runner. Presently, it supports a range of engines including Presto, Presto-on-Velox, and Spark. Beyond compute engines, SynQB's synthetic data can also be directly utilized for flexible integration testing and library testing across different platforms.

Minimized Regional Capacity Limitations. In Meta's data warehouse, queries are run on servers geographically near to where the data is stored to ensure low latency and network IO. As such, shadow testing solutions that run on real customer data require test environments in all regions where customer data may be located, which is not always feasible. In contrast, SynQB employs synthetic data that is not bound by regional constraints since the data can be automatically replicated to any region. This flexibility makes it possible to leverage any available test cluster for testing any workload.

4 DPSDG DEEP DIVE

In this chapter, we first briefly introduce some basic definitions and theorems of DP in Sec. 4.1, and then give an in-depth explanation of our DPSDG algorithm for SynQB in Sec. 4.2.

4.1 DP Basics

Consider a dataset D with N records (i.e., rows for a tabular dataset). We use $\mathcal{D}$ to denote the domain of D.

Next we formally define DP, which protects user-level privacy based on the notation of neighboring datasets, i.e., a pair of datasets which differ in just one record.

Definition 4.1 (Differential Privacy (DP) [5]). A randomized algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ is said to satisfy (ϵ, δ)-DP if $\forall D, D' \in \mathcal{D}$ s.t. they are neighboring datasets (denoted by $D \sim D'$), and any possible outcome $\mathcal{S} \subseteq \mathcal{R}$, we always have

$$\Pr(\mathcal{M}(D) \in \mathcal{S}) \leq e^{\epsilon} \Pr(\mathcal{M}(D') \in \mathcal{S}) + \delta. \tag{3}$$

Essentially, with DP guarantee, one cannot readily differentiate whether the input of algorithm $\mathcal{M}$ is D or D', due to the randomness of the algorithm. The combination (ϵ, δ) is referred to as privacy budget, which measures the privacy level of the algorithm $\mathcal{M}$.

Next, we introduce subsample procedure for dataset D and what it implies under the context of DP.

Definition 4.2 (Subsample Procedure [19]). For dataset D with N records, the `subsample` procedure selects a random dataset $\tilde{D}$ from the uniform distribution over all subsets of D of size $\tilde{N}$. The ratio $\gamma := \tilde{N}/N$ is defined as the sampling parameter of the `subsample` procedure.

Theorem 4.3 (Differental Privacy with subsampling[19]). *If $\mathcal{M}$ is (ϵ, δ)-DP, then $\mathcal{M} \circ$ `subsample` is $(log(1+\gamma(e^{\epsilon}-1)), \gamma\delta)$-DP.*

According to the above theorem, if we target at (ϵ, δ)-DP, then we can apply an algorithm $\mathcal{M}$ with only $(\log(1+\frac{1}{\gamma}(e^{\epsilon}-1)), \delta/\gamma)$-DP to the subsampled dataset $\tilde{D}$ as the input.

Below we introduce two widely-used properties of DP.

Theorem 4.4 (Sequential Composition[5]). *If randomized algorithm $\mathcal{M}_i : \mathcal{D} \to \mathcal{R}_i$ satisfies (ϵ_i, δ_i)-DP, $\forall i \in \{1, 2, \cdots, k\}$, then randomized algorithm $\mathcal{M}(D) \triangleq (\mathcal{M}_1(D), \mathcal{M}_2(D), \cdots, \mathcal{M}_k(D))$ satisfies $(\sum_i \epsilon_i, \sum_i \delta_i)$-DP.*

Theorem 4.5 (Post Processing[5]). *For randomized algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ and algorithm $g : \mathcal{R} \to \mathcal{G}$ (either randomized or deterministic), if $\mathcal{M}$ is (ϵ, δ)-DP, then $g \circ \mathcal{M}$ is also (ϵ, δ)-DP.*

Lastly, we introduce a variant of Gaussian mechanism which protects a deterministic function with DP by adding Gaussian noises.

Theorem 4.6 (Analytic Gaussian Mechanism[2]). *Analytic Gaussian mechanism is a (ϵ, δ)-DP mechanism that adds Gaussian noise to the result of a sensitive deterministic function $g : \mathcal{D} \to \mathbb{R}^p$:*

$$\mathcal{M}(D) = g(D) + \sigma\mathcal{N}(0, \mathbb{I}_p) \tag{4}$$

where σ is the scale of the Gaussian noise that satisfies

$$\Phi(\frac{\Delta_g}{2\sigma} - \frac{\epsilon\sigma}{\Delta_g}) - e^{\epsilon}\Phi(-\frac{\Delta_g}{2\sigma} - \frac{\epsilon\sigma}{\Delta_g}) \leq \delta \tag{5}$$

where Φ is the CDF of the standard univariate Gaussian distribution and $\Delta_g \triangleq \sup_{D \sim D'} \|g(D) - g(D')\|$ is the L_2-sensitivity of g.

4.2 DPSDG Algorithm

We notice that a user dataset generally contain two categories of information: 1) **User features**, i.e., the demographics of an individual, such as country, gender, age, etc; and 2) **UII**, i.e., the information can be used to directly identify a specific user, such as user ID, phone number, name, etc.

An implicit assumption of most prior works is that releasing the *values* of the columns (e.g., users are from these countries: USA and Canada) is not risky, but the *distributions* of the columns (e.g., 70% and 30% of the users are from USA and Canada, respectively) is

Table 1: Split of Dataset D

Feature Columns ($D_{\text{ft.}}$)			UII Columns (D_{uii})		
Country	Gender	$\cdots$	User ID	Name	$\cdots$
USA	Male		1001	John	
Canada	Female		1002	Alice	

Algorithm 1: Differentially-Private Synthetic Data Generation for feature columns `DPGCKendall`

Input: $D_{\text{ft.}}$, ϵ, δ
Output: $D_{\text{ft., syn}}$
1: **procedure** `DPGCKendall-Fit`($D_{\text{ft.}}$; ϵ, δ)
2: Flatten the structural columns and then discretize the continuous columns
3: Split the privacy budget into ϵ_1, δ_1 and ϵ_2, δ_2, s.t. $\epsilon_1 + \epsilon_2 = \epsilon$, and $\delta_1 + \delta_2 = \delta$
4: Compute the empirical 1-dimensional marginal distribution $F_{\text{ft.}}$ and Kendall's τ correlation coefficient matrix $\Sigma^{\tau}_{\text{ft.}}$ of $D_{\text{ft.}}$
5: Apply analytic Gaussian mechanism to $F_{\text{ft.}}$ and $\Sigma^{\tau}_{\text{ft.}}$, making them (ϵ_1, δ_1)-DP and (ϵ_2, δ_2)-DP, respectively
6: Estimate the Pearson's correlation coefficient matrix $\Sigma_{\text{ft.}}$
7: **return** $M_{\text{ft.}} = (F_{\text{ft.}}, \Sigma_{\text{ft.}})$
8: **procedure** `DPGCKendall-Gen`($M_{\text{ft.}}$, N_{syn})
9: Sample $D_{\text{ft., syn}}$ with N_{syn} samples that comply with $F_{\text{ft.}}$,$\Sigma_{\text{ft.}}$
10: Inverse transform the discretized continuous columns and then flattened structural columns of $D_{\text{ft., syn}}$
11: **return** $D_{\text{ft., syn}}$

sensitive and hence should be protected by DP. For feature columns, such assumption normally makes sense, because it is quite common that a non-trivial number of people have the same demographics. However, these prior works didn't consider UII, whose *values* (e.g., there exists a user with ID 1001) are inherently very sensitive, and thus should be handled differently.

Therefore, in this paper, as illustrated in Table 1, we split D vertically: $D = (D_{\text{ft.}}, D_{\text{uii}})$, where $D_{\text{ft.}}$ and D_{uii} are datasets with feature columns and UII columns, respectively. In the following sections, we will apply different DPSDG algorithms to $D_{\text{ft.}}$ and D_{uii}. In particular, we carefully designed a DPSDG algorithm for UII such that we can preserve the distribution of UIIs under DP, while the corresponding values will not be leaked.

DPSDG Algorithm for feature columns. Our DPSDG algorithm for feature columns is presented in Algorithm 1, which is largely based on the DP Gaussian Copula Kendall[3] from [8], with some modifications. For the fit step, in line 2, we first flatten the structural columns and discretize the continuous columns, which is because DP Gaussian Copula Kendall, as a marginal-based mechanism, requires the input to be unnested and discretized in order to compute the discrete marginal distribution; Next, in line 3-6, we split the privacy budget and then compute the noisy empirical 1-dimensional marginal distribution $F_{\text{ft.}}$ and Pearson's correlation coefficient matrix $\Sigma_{\text{ft.}}$ (estimated from Kendall's τ correlation coefficient matrix $\Sigma^{\tau}_{\text{ft.}}$, see details in [8]); and in line 7, the trained model includes these noisy marginals and will be returned. For the generate step, we sample N_{syn} samples that comply with these noisy marginals in line 9, and then inverse transform the discretized continuous columns and then flattened structural columns in line 10, which produces the synthetic feature dataset $D_{\text{ft., syn}}$. This algorithm can be proved to be (ϵ, δ)-DP based on Theorem 4.4 and Theorem 4.5.

[3]DP Gaussian Copula Kendall largely preserves the 1- and 2-dimensional distributions of the production dataset, which we find is generally good enough for compute engine testing.

Algorithm 2: Differentially-Private Synthetic Data Generation for UII columns `DPUII`

Input: D_{uii}, ϵ, δ
Output: $D_{\text{uii, syn}}$
1: **procedure** `DPUII-Fit`(D_{uii}; ϵ, δ)
2: Compute the second-order histogram H_{uii} of D_{uii}
3: Apply analytic Gaussian mechanism (Theorem 4.6) to H_{uii}, making it (ϵ, δ)-DP
4: **return** $M_{\text{uii}} = H_{\text{uii}}$
5: **procedure** `DPUII-Gen`(M_{uii}, N_{syn})
6: Sample $D_{\text{uii, syn}}$ from a test domain $\mathcal{D}^{\text{test}}_{\text{uii}}$ with N_{syn} samples that comply with H_{uii}
7: **return** $D_{\text{uii, syn}}$

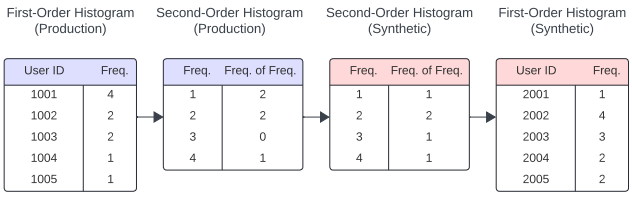


Figure 3: Illustration of second-order histogram computation. The left two tables are the first-order and second-order histograms for production table, and the right two tables are second-order and first-order histograms for synthetic table.

DPSDG Algorithm for UII columns. Our DPSDG algorithm for UII columns is shown in Algorithm 2. The key of the algorithm is to compute the second-order histogram of D_{uii}, denoted by H_{uii}, as in line 2. For example, consider a UII dataset D_{uii} with just one column "User ID". As illustrated in Fig. 3, we first compute the first-order histogram of D_{uii}, which indicates the frequency (or occurrence) of each user ID. Due to the sensitivity of the values of user IDs themselves, we should not directly preserve the first-order histogram. Instead, we further compute the second-order histogram (i.e., the histogram of the first-order histogram) of D_{uii}, which contains only the values of the frequencies as well as the frequencies of these frequencies. This way, we not only drop those sensitive ID values, but also compress the key information since the cardinality of user IDs is in general much larger than their frequencies. Once we have H_{uii}, the remaining part of the algorithm (line 3-7, i.e., noise perturbation and sampling) is quite similar to Algorithm 1, except that we need to sample $D_{\text{uii, syn}}$ from a test UII dataset domain $\mathcal{D}^{\text{test}}_{\text{uii}}$, i.e., synthetic UIIs are essentially pseudonyms

Algorithm 3: Differentially-Private Synthetic Data Generation DPSDG

Input: D, ϵ, δ
Output: D_{syn}

procedure DPSDG-Fit($D; \epsilon, \delta$)
  Split the privacy budget into $\epsilon_{\text{ft.}}, \delta_{\text{ft.}}$ and $\epsilon_{\text{uii}}, \delta_{\text{uii}}$, s.t. $\epsilon_{\text{ft.}} + \epsilon_{\text{uii}} = \epsilon$, and $\delta_{\text{ft.}} + \delta_{\text{uii}} = \delta$
  Subsample $D_{\texttt{ft.}}$ and $D_{\texttt{uii}}$ with rate $\gamma_{ft.}$ and γ_{uii} respectively. Get subsampled dataset $\tilde{D}_{\texttt{ft.}}$ and $\tilde{D}_{\texttt{uii}}$
  Adjust $\epsilon_{\text{ft.}}, \delta_{\text{ft.}}$ and $\epsilon_{\text{uii}}, \delta_{\text{uii}}$ according to Theorem 4.3. Get $\tilde{\epsilon}_{\text{ft.}}, \tilde{\delta}_{\text{ft.}}$ and $\tilde{\epsilon}_{\text{uii}}, \tilde{\delta}_{\text{uii}}$
  $M_{\text{ft.}}$ = DPGCKendall-Fit($\tilde{D}_{\texttt{ft.}}; \tilde{\epsilon}_{ft.}, \tilde{\delta}_{ft.}$)
  M_{uii} = DPUII-Fit($\tilde{D}_{\texttt{uii}}; \tilde{\epsilon}_{uii}, \tilde{\delta}_{uii}$)
  return $M = (M_{\text{ft.}}, M_{\text{uii}})$
procedure DPSDG-Gen(M, N_{syn})
  $D_{\text{ft., syn}}$ = DPGCKendall-Gen($M_{\text{ft.}}, N_{\text{syn}}$)
  $D_{\text{uii, syn}}$ = DPUII-Gen($M_{\text{uii}}, N_{\text{syn}}$)
  return $D_{\text{syn}} = (D_{\text{ft., syn}}, D_{\text{uii, syn}})$

unlinkable to the original UIIs. In the example of Fig. 3, the synthetic user IDs start from 2001, and cannot be linked back to the real user IDs in the input dataset.

Complete DPSDG Algorithm. The complete DPSDG algorithm to synthesize a dataset D is given in Algorithm 3, which incorporates privacy budget splitting, subsampling and utilizing Algorithm 1 and 2. According to Theorem 4.3, 4.4 and 4.5, the algorithm can be proved to be (ϵ, δ)-DP.

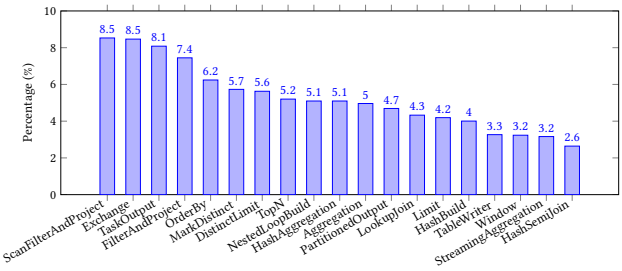
5 EVALUATION

This chapter presents our experience with utilizing privacy-compliant, production-like synthetic data generated by SynQB for compute engine testing. The key results are summarized as follows.

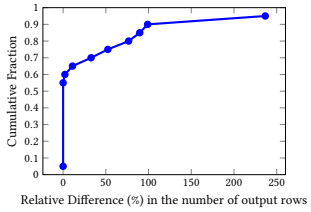
SynQB provides a comprehensive coverage of diverse query types. Fig. 4(a) illustrates the extensive range of operator types covered by SynQB through the distribution. This highlights the significant advantage of SynQB, which is its high degree of adaptability and control over operator and feature coverage. This flexibility allows for customizing coverage to suit various requirements or objectives for continuous verification efforts.

The similarity between synthetic data and production data is high for compute engine testing. Utilizing SynQB's extensive collection of queries for Presto testing, we executed the modified queries using the corresponding synthetic data sets and compared the resulting output row counts with the production runs. As illustrated in Fig. 4(b), the cumulative distribution function (CDF) of the relative differences in output rows indicates that approximately 70% of the queries exhibit <32.8% variation, with 55% of those yielding identical output row counts as the production runs. Although the remaining 30% display a greater relative difference, which may not be suitable for performance assessments but can still serve the purpose of correctness testing.

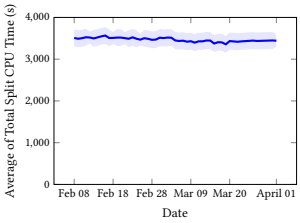
The performance test results over time serve as an indicator of potential regressions. We selected a set of queries and



(a) Operator Distribution.



(b) CDF of relative difference in the number of output rows.



(c) Average of total split CPU time from February to April.

Figure 4: SynQB Evaluation.

monitored their performance over time. Fig. 4(c) depicts the fluctuation in the average total split CPU time between February and April. Notably, the average CPU time remained remarkably stable, implying that the Presto engine did not experience any significant regressions during the specified period.

6 CONCLUSIONS

We built SynQB - a query bank that leverages a carefully designed DPSDG algorithm to generate privacy-compliant production-like synthetic data. It enables test coverage for different compute engines, query operators and workloads and facilitates detection of correctness and performance regressions, and therefore greatly improves our confidence in new compute engine releases and enhances the reliability of our data warehouse.

ACKNOWLEDGMENTS

We thank our colleagues Pedro Eugenio Rocha Pedreira, Ananth Raghunathan and Haozhi Xiong for providing feedback on the manuscript; our legal counsels Sarani Millican, Alastair Agcaoili and Santosh Putchala for facilitating the privacy review; and our managers Vivek Gaur, Naveen Cherukuri, Raghu Raman, Kenneth

Ho, and Payman Mohassel, as well as our directors JR Tipton and Paolo Raden, for supporting the execution of this project.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[2] Borja Balle and Yu-Xiang Wang. 2018. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*. PMLR, 394–403.

[3] Christoph Brücke, Philipp Härtling, Rodrigo D Escobar Palacios, Hamesh Patel, and Tilmann Rabl. 2023. TPCx-AI-an industry standard benchmark for artificial intelligence and machine learning systems. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3649–3661.

[4] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 265–284.

[5] 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.

[6] Amir Gilad, Shweta Patwa, and Ashwin Machanavajjhala. 2021. Synthesizing linked data under cardinality and integrity constraints. In *Proceedings of the 2021 International Conference on Management of Data*. 619–631.

[7] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. 2018. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*.

[8] Haoran Li, Li Xiong, and Xiaoqian Jiang. 2014. Differentially private synthesization of multi-dimensional data using copula functions. In *Advances in database technology: proceedings. International conference on extending database technology*, Vol. 2014. NIH Public Access, 475.

[9] Miro Mannino and Azza Abouzied. 2019. Is this real? Generating synthetic data that looks real. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 549–561.

[10] Ryan McKenna, Brett Mullins, Daniel Sheldon, and Gerome Miklau. 2022. Aim: An adaptive and iterative mechanism for differentially private synthetic data. *arXiv preprint arXiv:2201.12677* (2022).

[11] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. 2019. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning*. PMLR, 4435–4444.

[12] NATIONAL SCIENCE AND TECHNOLOGY COUNCIL. 2023. NATIONAL STRATEGY TO ADVANCE PRIVACY-PRESERVING DATA SHARING AND ANALYTICS. https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Strategy-to-Advance-Privacy-Preserving-Data-Sharing-and-Analytics.pdf.

[13] Pedro Pedreira, Orri Erling, Masha Basmanova, Kevin Wilfong, Laith Sakka, Krishna Pai, Wei He, and Biswapesh Chattopadhyay. 2022. Velox: meta's unified execution engine. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3372–3384.

[14] Tilmann Rabl, Manuel Danisch, Michael Frank, Sebastian Schindler, and Hans-Arno Jacobsen. 2015. Just can't get enough: Synthesizing Big Data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1457–1462.

[15] Entong Shen and Lyublena Antova. 2013. Reversing statistics for scalable test databases generation. In *Proceedings of the Sixth International Workshop on Testing Database Systems*. 1–6.

[16] Yutian Sun, Tim Meehan, Rebecca Schlussel, Wenlei Xie, Masha Basmanova, Orri Erling, Andrii Rosa, Shixuan Fan, Rongrong Zhong, Arun Thirupathi, et al. 2023. Presto: A Decade of SQL Analytics at Meta. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[17] Yuchao Tao, Ryan McKenna, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. 2021. Benchmarking differentially private synthetic data generation algorithms. *arXiv preprint arXiv:2112.09238* (2021).

[18] Reihaneh Torkzadehmahani, Peter Kairouz, and Benedict Paten. 2019. Dp-cgan: Differentially private synthetic data and label generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 0–0.

[19] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. 2019. Subsampled rényi differential privacy and analytical moments accountant. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 1226–1235.

[20] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. 2018. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739* (2018).

[21] Jiaqi Yan, Qiuye Jin, Shrainik Jain, Stratis D Viglas, and Allison Lee. 2018. Snowtrail: Testing with production queries on a cloud database. In *Proceedings of the Workshop on Testing Database Systems*. 1–6.