Training-based versus training-free differential privacy for data synthesis

Mehak Kapur
mekapur@ucsd.edu

Hana Tjendrawasi
htjendrawasi@ucsd.edu

Jason Tran
jat037@ucsd.edu

Phuc Tran
pct001@ucsd.edu

Yu-Xiang Wang
yuxiangw@ucsd.edu

Abstract

Differentially private synthetic data generation promises to resolve the tension between data utility and individual privacy, enabling the release of datasets that preserve the statistical properties analysts need while bounding what any adversary can learn about a single record. Two paradigms have emerged to fulfill this promise. Training-based methods inject calibrated noise during model optimization, coupling privacy to the learning process itself. Training-free methods instead leverage foundation models through black-box API access, achieving privacy through selection mechanisms that never touch the model's parameters. Both have demonstrated success on image and text benchmarks, yet their behavior on realistic, multi-table relational data remains largely unexplored. We investigate both approaches on Intel's Driver and Client Applications (DCA) telemetry corpus, evaluating against a benchmark of 21 analytical SQL queries representative of production business intelligence workloads.

Code: https://github.com/mekapur/DSC180B-Q2

1 Introduction

1.1 Motivation

Organizations routinely collect detailed telemetry from their products to drive business decisions. Engineers use it to diagnose failures, product teams analyze usage trends, and analysts extract insights that shape product roadmaps. The same granularity that makes telemetry valuable also makes it sensitive. Browsing patterns, work schedules, and device fingerprints can re-identify specific individuals even after conventional anonymization.

Privacy regulations such as GDPR and CCPA, along with institutional policies, increasingly restrict how such data can be stored, shared, and analyzed. The resulting tension between data utility and privacy protection motivates the development of *synthetic data generation*, producing artificial datasets that preserve the statistical properties necessary for analysis while providing formal guarantees that no individual's information can be recovered.

Two paradigms have emerged for generating synthetic data with rigorous privacy guarantees. *Training-based methods* optimize generative models under constraints that bound individual influence, adding calibrated noise during the training process. *Training-free methods* leverage pre-trained foundation models through black-box API access, achieving privacy through carefully designed selection mechanisms rather than private optimization. Both approaches have demonstrated success in isolation on image and text benchmarks, yet no comprehensive comparison exists under controlled experimental conditions with realistic analytical workloads on multi-table relational data.

This project provides that comparison. We implement both paradigms on Intel's Driver and Client Applications (DCA) telemetry corpus and evaluate them against a benchmark of 21 SQL queries representative of production analytics, measuring which approach better preserves query fidelity under equivalent privacy budgets.

1.2 Prior work

Differential privacy, introduced by Dwork and Roth (2014), provides the foundational framework for our privacy guarantees. A randomized mechanism $\mathcal{M}$ is (ε, δ)-differentially private if for all neighboring databases D, D' differing by one record and all measurable output sets S:

$$\Pr[\mathcal{M}(D) \in S] \leq e^{\varepsilon} \Pr[\mathcal{M}(D') \in S] + \delta. \tag{1}$$

The parameter ε controls the privacy-utility tradeoff. Smaller values confer stronger privacy at the cost of noisier outputs. The slack δ permits rare violations and must remain negligible relative to the database size. The work established key mechanisms (Laplace, Gaussian, exponential) for releasing numeric queries, along with composition theorems demonstrating that privacy loss accumulates across multiple analyses.

Abadi et al. (2016) extended differential privacy to deep learning through DP-SGD. Standard gradient descent leaks information through the unbounded influence any single train-

ing example can exert on model parameters. DP-SGD bounds this influence by clipping each per-sample gradient to a fixed ℓ_2 norm and injecting calibrated Gaussian noise to mask individual contributions. The accompanying *moments accountant* yields substantially tighter privacy bounds than naive composition, enabling practical deep learning under modest privacy budgets.

Ghalebikesabi et al. (2023) demonstrated that fine-tuning diffusion models with DP-SGD generates synthetic images of reasonable quality, though their approach requires substantial privacy budgets ($\varepsilon \approx 32$ for CIFAR-10). This motivates the search for methods that achieve comparable fidelity at lower ε.

Lin et al. (2025) introduced Private Evolution (PE), a fundamentally different paradigm that avoids model training entirely. PE operates through black-box API access to pre-trained foundation models, iteratively evolving synthetic samples by computing differentially private nearest-neighbor histograms. Each private data point votes for its nearest synthetic candidate; Gaussian noise is added to the vote histogram, and candidates are resampled according to the noisy distribution. PE achieved FID ≤ 7.9 at $\varepsilon = 0.67$ on CIFAR-10, a substantial improvement over DP-Diffusion in both privacy cost and output quality. The privacy analysis is straightforward. Each iteration releases one Gaussian mechanism, and T iterations compose to a single mechanism with noise multiplier $\sigma/\sqrt{T}$.

Xie et al. (2024) extended PE to text through Aug-PE, introducing fill-in-the-blanks variation, adaptive text lengths, and rank-based selection. Aug-PE with GPT-3.5 outperformed DP-finetuning baselines at the same privacy budget while running 12–66× faster.

Swanberg et al. (2025) adapted PE for tabular data using a workload-aware distance function that measures proximity in terms of query-relevant predicates rather than raw feature similarity. Their central finding is negative. API access to large language models does not yet improve differentially private tabular synthesis beyond established marginal-based baselines such as MST and JAM. This result informs our expectations for applying PE to the DCA telemetry corpus.

2 Data and problem statement

2.1 Intel DCA telemetry data

Our investigation uses the Intel Driver and Client Applications (DCA) telemetry dataset, a large-scale collection of system-level signals from Windows client machines. The full corpus comprises approximately 115 tables in the `university_prod` schema (9.1 TB total), organized around a globally unique identifier (`guid`) assigned to each client system. Raw event tables record per-interval measurements (hourly or daily) across power, thermal, network, application, and browsing domains. Individual tables range from tens of millions to billions of rows.

The 24 benchmark queries do not reference the raw event tables directly. They reference a `reporting` schema of pre-aggregated views that Intel analysts built on top of the raw data. These reporting tables aggregate per-event measurements to per-`guid` summaries (or per-`guid`-per-day, depending on the table). Since the reporting views are not distributed with the raw data, we reconstruct them from the raw tables using DuckDB aggregation scripts that follow Intel's documented ETL logic.

We work with a sample of the full corpus. Each raw table in the repository is split into multiple partition files of varying size, and we download a single partition per table (the first available file, typically between 1 and 5 GiB). Several tables are supplemented or replaced entirely by pre-aggregated files from a December 2024 update that Intel added to the repository. The `system_sysinfo_unique_normalized` anchor table is downloaded in full (1,000,000 `guids`). All event tables are filtered to `guids` present in this anchor, ensuring a coherent sample. Total data on disk is approximately 20.7 GiB.

Table 1 lists the 19 reporting tables we construct, organized by category. The `guid` coverage column reports how many of the 1,000,000 anchor `guids` have at least one nonzero entry in each table. Coverage varies by two orders of magnitude. The web browsing pivot table covers 512,077 `guids` (51.2%), while the PSYS RAP power metric covers only 816 (0.08%). This heterogeneous coverage is a defining characteristic of the dataset and, as we show in Section 4, a primary source of difficulty for synthesis.

Table 1: The 19 reporting tables constructed from raw DCA telemetry data.

Reporting table	Category	Raw source	Rows	Guids
`system_sysinfo_unique_normalized`	Metadata	Direct copy	1,000,000	1,000,000
`system_cpu_metadata`	Metadata	Dec. 2024 update	1,000,000	1,000,000
`system_os_codename_history`	Metadata	Dec. 2024 update	639,223	—
`system_hw_pkg_power`	Power/thermal	`hw_metric_stats`	318,791	∼800
`system_psys_rap_watts`	Power/thermal	`hw_metric_stats`	4,846	816

Continued on next page

Reporting table	Category	Raw source	Rows	Guids
system_pkg_C0	Power/thermal	hw_metric_stats	945,500	8,943
system_pkg_avg_freq_mhz	Power/thermal	hw_metric_stats	12,844	613
system_pkg_temp_centigrade	Power/thermal	hw_metric_stats	13,091	622
system_batt_dc_events	Battery	Dec. 2024 update	~49,000	—
system_on_off_suspend_time_day	Battery	Dec. 2024 update	1,582,017	—
system_frgnd_apps_types	Application	Dec. 2024 update	56,755,998	55,830
system_userwait	Application	userwait_v2	175,223,880	38,142
system_web_cat_pivot_duration	Browsing	web_cat_pivot	512,077	512,077
system_web_cat_usage	Browsing	web_cat_usage_v2	21,354,922	64,276
system_network_consumption	Network	os_network_consumption_v2	121,843,286	37,224
system_memory_utilization	Memory	os_memsam_avail_percent	21,688,089	69,552
system_display_devices	Display	Dec. 2024 update	220,997,262	209,239
system_mods_top_blocker_hist	Sleep study	Dec. 2024 update	92,460,980	—
system_mods_power_consumption	Sleep study	Dec. 2024 update (stub)	10,000	1

The anchor table (system_sysinfo_unique_normalized) provides static client attributes such as chassis type, country, OEM, RAM capacity, CPU family and generation, processor number, operating system, and a derived persona classification. Every benchmark query that segments by demographic or geographic attributes joins against this table.

The five power and thermal tables are all derived from a single raw source (hw_metric_stats) by filtering on the name column (e.g., HW::PACKAGE:IA_POWER:WATTS: for package power, HW::PACKAGE:C0_RESIDENCY:PERCENT: for C0 residency). Each provides per-guid weighted averages and sample counts. Coverage is sparse. The C0 metric has 8,943 guids, while PSYS RAP, frequency, and temperature each cover fewer than 1,000.

The application tables contain per-process and per-application breakdowns. system_userwait records wait events (duration > 1 second) with the offending process name, wait type, and AC/DC power state. system_frgnd_apps_types records foreground application usage by executable name, application type, and daily focal screen time.

The browsing tables take two forms. system_web_cat_pivot_duration is a wide table with one row per guid and 28 browsing-category columns (education, finance, gaming, mail, news, social media, etc.), each recording total duration. system_web_cat_usage provides per-browser statistics (chrome, edge, firefox) with system count, instance count, and duration.

The system_mods_power_consumption table is a stub. Intel's December 2024 update contains only 10,000 rows from a single guid. Three benchmark queries reference this table; they execute and produce rankings, but the results reflect one client's power profile rather than population-level statistics. We retain these queries for pipeline completeness but exclude them from quantitative evaluation.

2.2 Analytical query workload

The practical utility of synthetic telemetry data is determined by its ability to support real analytical workloads. We operationalize this through a benchmark suite of 21 SQL queries developed by Intel analysts, spanning five categories:

1. *Aggregate statistics with joins* (6 queries): weighted averages across multiple tables joined on guid, testing whether cross-table correlations survive synthesis.
2. *Ranked top-k* (7 queries): window functions producing ranked lists of applications, processes, or browsers, testing whether relative orderings are preserved.
3. *Geographic and demographic breakdowns* (4 queries): segmentation by country, processor generation, or persona, testing preservation of conditional distributions.
4. *Histograms and distributions* (2 queries): binned aggregations testing whether distributional shapes survive synthesis.
5. *Complex multi-way pivots* (2 queries): high-dimensional joint distributions across browsing categories, devices, or user segments.

2.3 Query inventory

Table 2 lists all 21 feasible queries in the benchmark. Although the original query list contains 24 queries, three additional queries are permanently infeasible because they require the system_mods_power_consumption table, for which no viable data source exists in the DCA corpus available to us.[1]

[1]The three infeasible queries are ranked_process_classifications, top_10_processes_per_user_id_ranked_by_total_power_consumption, and top_20_most_power_consuming_processes_by_avg_power_consumed.

Table 2: Complete inventory of the 21 feasible benchmark queries, grouped by query type. "Agg+Join" denotes aggregate statistics with multi-table joins; "Top-k" denotes ranked lists produced by window functions; "Geo/Demo" denotes geographic or demographic breakdowns; "Histogram" denotes binned distributions; "Pivot" denotes complex multi-way pivots.

Query	Codename	Type	Description
avg_platform_power_c0_freq_temp_by_chassis	ChassisAgg	Agg+Join	Average power, C0 residency, frequency, and temperature by chassis type (5-way join).
server_exploration_1	NetAsymAgg	Agg+Join	Identify client machines sending more data than they receive, joined with sysinfo for OS and chassis.
mods_blockers_by_osname_and_codename	BlockerAgg	Agg+Join	Count distribution of modern sleep study blockers by Windows OS name and codename.
top_mods_blocker_types_durations_by_osname_and_codename	BlockerDurAgg	Agg+Join	Blocker entry counts and average durations by OS name, codename, blocker type, and activity level.
display_devices_connection_type_resolution_durations_ac_dc	DisplayAgg	Agg+Join	Display connection types with resolution and average AC/DC durations.
display_devices_vendors_percentage	VendorAgg	Agg+Join	Percentage of systems by display vendor.
most_popular_browser_in_each_country_by_system_count	BrowserRank	Top-k	Most popular browser per country by system count.
userwait_top_10_wait_processes	WaitRank	Top-k	Top 10 applications by average wait time.
userwait_top_10_wait_processes_wait_type_ac_dc	WaitModeRank	Top-k	Top 10 wait-time applications by wait type (app start vs. in-app) and power state.
userwait_top_20_wait_processes_compare_ac_dc_unknown_durations	WaitStateRank	Top-k	Top 20 wait-time applications with durations pivoted by power state (AC/DC/Unknown).
top_10_applications_by_app_type_ranked_by_focal_time	FocalRank	Top-k	Top 10 applications per app type by average daily focal screen time.

Continued on next page

Query	Codename	Type	Description
top_10_applications_by_app_type_ranked_by_system_count	SystemRank	Top-k	Top 10 applications per app type by distinct client count.
top_10_applications_by_app_type_ranked_by_total_detections	DetectionRank	Top-k	Top 10 applications per app type by total daily detection count.
Xeon_network_consumption	XeonGeo	Geo/Demo	Network consumption summary for Xeon vs. non-Xeon systems, by OS.
pkg_power_by_country	PowerGeo	Geo/Demo	Average CPU package power by country.
battery_power_on_geographic_summary	BatteryGeo	Geo/Demo	Battery power-on count and duration by country (min. 100 clients).
battery_on_duration_cpu_family_gen	BatteryDemo	Geo/Demo	Battery duration by CPU family and generation (min. 100 clients).
ram_utilization_histogram	RamHist	Histogram	Average RAM utilization percentage by memory capacity.
popular_browsers_by_count_usage_percentage	BrowserHist	Histogram	Percentage of systems, instances, and duration by browser.
persona_web_cat_usage_analysis	PersonaPivot	Pivot	Web browsing category duration as weighted percentages, by persona.
on_off_mods_sleep_summary_by_cpu_marketcodename_gen	SleepPivot	Pivot	On/off/sleep/MODS time percentages by CPU generation and market codename.

For the sake of brevity, we will henceforth refer to each query by its codename instead of its full name.

2.4 Formal benchmark definition

Let $\mathcal{Q} = \{q_1, \ldots, q_{21}\}$ denote our SQL query benchmark. Each query q_j maps a database instance to a result set $q_j(D) \in \mathcal{R}_j$, where $\mathcal{R}_j$ may be a scalar, vector, or table. The query discrepancy for synthetic data $\tilde{D}$ is:

$$\Delta_j(D, \tilde{D}) = d_j\big(q_j(D),\ q_j(\tilde{D})\big), \tag{2}$$

where d_j is a distance metric appropriate to the result type (relative error for scalars, Spearman's ρ for rankings, total variation for histograms). The aggregate benchmark score is:

$$\text{Score}(\tilde{D}) = \frac{1}{|\mathcal{Q}|} \sum_{j=1}^{|\mathcal{Q}|} \mathbf{1}\big[\Delta_j(D, \tilde{D}) \leq \tau_j\big], \tag{3}$$

where τ_j is a query-specific tolerance threshold. A synthetic dataset passes the benchmark if it achieves a high score, indicating that analysts could substitute $\tilde{D}$ for D without materially affecting conclusions.

2.5 Research questions

Given the 21-query benchmark and matched privacy budgets, we investigate the following.

1. Under matched (ε, δ), which method achieves higher benchmark scores? Which query types exhibit the largest discrepancy?
2. Does error compound across multi-table joins, or does the synthesis mechanism preserve joint distributions adequately?
3. Which method better preserves minority class frequencies (prevalence $< 5\%$)?
4. Do classifiers trained on synthetic data achieve comparable accuracy to those trained on real data?
5. What are the wall-clock time and resource requirements for each method?

3 Methods

3.1 Training-based synthesis, DP-SGD and VAE

The unit of synthesis is the `guid`. Each `guid` represents one client system, and the privacy guarantee is per-`guid`, meaning neighboring databases differ by adding or removing all rows associated with a single device.

Synthesizing each reporting table independently would destroy cross-table correlations. Most benchmark queries join multiple tables on `guid`; if synthetic tables share no relationship, joins produce either zero matches (mismatched `guids`) or random associations (correct `guids` but uncorrelated attributes). Queries measuring cross-table relationships (e.g., "average network consumption by chassis type") would return meaningless results.

We therefore construct a single wide table with one row per `guid` containing all attributes and pre-aggregated metrics from every reporting table. For each reporting table, we compute `guid`-level aggregations of the columns referenced by benchmark queries. Multi-row-per-`guid` tables (e.g., web browsing by category) are pivoted into separate columns per category. All aggregations are LEFT JOINed onto the sysinfo anchor table (1,000,000 `guids`).

The resulting wide table has 1,000,000 rows and 70 columns, comprising 9 categorical attributes and 59 numeric metrics plus `guid`. We encode categoricals via top-k binning ($k = 50$, with remaining values grouped into "Other") and one-hot encoding. Numeric columns are clipped at the 99.9th percentile, transformed via $\log(1+x)$ to reduce skew, and standardized with zero mean and unit variance. The final feature matrix has 1,000,000 × 307 entries (248 one-hot indicators plus 59 scaled numerics).

3.1.1 DP-VAE architecture

We train a differentially private variational autoencoder (DP-VAE) on the wide table. Let $x \in \mathbb{R}^{307}$ denote a `guid`-level feature vector. The encoder maps x through two hidden layers of 512 units each to produce a 64-dimensional mean μ and log-variance $\log \sigma^2$. We sample $z = \mu + \sigma \odot \epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$.

The decoder uses separate heads for categorical and numeric attributes:

- 9 categorical heads, each producing logits over the corresponding column's vocabulary, trained with cross-entropy loss.
- 1 numeric head mapping z to 59 outputs, trained with mean squared error.

The total loss is:

$$\mathcal{L} = \sum_{c=1}^{9} \mathrm{CE}(x_c, \hat{x}_c) + \mathrm{MSE}(x_{\mathrm{num}}, \hat{x}_{\mathrm{num}}) + \mathrm{KL}\big(q_\phi(z \mid x) \,\|\, \mathcal{N}(0, I)\big). \tag{4}$$

The model has 505,971 trainable parameters.

Algorithm 1: DP-VAE training with DP-SGD

Input : Wide table $X \in \mathbb{R}^{n \times 307}$, target $\varepsilon^\star$, δ, clip norm C, epochs E, batch size B
Output: Trained parameters θ, final ε
Wrap optimizer with Opacus: $\sigma \leftarrow$ `make_private_with_epsilon`$(\varepsilon^\star, \delta, E, B)$;
for *epoch* $= 1, \ldots, E$ **do**
  for *each batch* $\{x_i\}_{i=1}^{B} \subset X$ **do**
    Encode: $(\mu_i, \log \sigma_i^2) \leftarrow \text{Encoder}(x_i)$;
    Sample: $z_i \leftarrow \mu_i + \sigma_i \odot \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, I)$;
    Decode: $\hat{x}_i \leftarrow \text{Decoder}(z_i)$;
    Compute $\mathcal{L}_i = \text{CE}_i + \text{MSE}_i + \text{KL}_i$;
    Clip: $\bar{g}_i \leftarrow \nabla_\theta \mathcal{L}_i \cdot \min(1, C / \|\nabla_\theta \mathcal{L}_i\|_2)$;
    Noise: $\tilde{g} \leftarrow \frac{1}{B}\big(\sum_i \bar{g}_i + \mathcal{N}(0, \sigma^2 C^2 I)\big)$;
    Update: $\theta \leftarrow \theta - \eta \tilde{g}$;
$\varepsilon \leftarrow$ `PrivacyAccountant.get_epsilon`(δ);

3.1.2 DP-SGD training and privacy accounting

We use the Opacus library to apply DP-SGD. Rather than manually selecting a noise multiplier, we use Opacus's `make_private_with_epsilon` to automatically calibrate σ for a target budget of $\varepsilon = 4.0$ at $\delta = 10^{-5}$. Privacy loss is tracked via the Rényi differential privacy (RDP) accountant, which yields tighter bounds than basic composition.

Table 3 reports the training configuration. The final privacy expenditure is $\varepsilon = 3.996$.

Table 3: DP-VAE training configuration.

Hyperparameter	**Value**
Encoder architecture	$307 \to 512 \to 512 \to (64\mu, 64 \log \sigma^2)$
Decoder architecture	$64 \to 512 \to 512 \to 9$ categorical + 1 numeric head
Total parameters	505,971
Batch size	4,096
Epochs	20
Learning rate	10^{-3} (Adam)
Max gradient norm C	1.0
Noise multiplier σ	auto-calibrated by Opacus
Target ε	4.0
δ	10^{-5}
Final ε	3.996
Training time	359.7 min (CPU)

3.1.3 Synthetic data generation and decomposition

After training, we generate 1,000,000 synthetic `guid` records:

1. Sample $z \sim \mathcal{N}(0, I_{64})$.
2. Decode to produce categorical logits and numeric outputs.
3. For each categorical column, sample from the softmax distribution over the vocabulary.
4. For each numeric column, apply the inverse transformations (de-standardize, apply $\text{expm1}(x) = e^x - 1$, clip to the observed range).

The synthetic wide table is then decomposed back into 12 reporting table schemas by selecting the relevant columns for each table and assigning synthetic `guids`. The original benchmark SQL queries execute unchanged on these synthetic reporting tables.

We evaluate 8 of the 21 feasible queries whose reporting tables are fully reconstructable from the wide-table columns. The remaining 13 queries require per-row columns not present in the wide table (e.g., per-process breakdowns for userwait, per-application data for foreground apps, per-display data for display devices).

3.2 Training-free synthesis, Private Evolution

We implement Private Evolution adapted for tabular data, following the framework of Lin et al. (2025) with the workload-aware distance function proposed by Swanberg et al. (2025). The algorithm operates in a single iteration ($T = 1$, the optimal setting for tabular data per Swanberg et al.'s finding that subsequent iterations provide marginal gains outweighed by composition cost).

1. *Population generation:* prompt a foundation model (OpenAI gpt-5-nano via the Batch API) to produce $N_{\text{synth}} \times L$ synthetic records conforming to the DCA wide-table schema (68 fields: 9 categorical as constrained strings, 59 numeric as floats). Each batch request uses Pydantic Structured Outputs to guarantee schema compliance.
2. *DP nearest-neighbor histogram:* each of the $n = 1{,}000{,}000$ real records votes for its nearest synthetic candidate under the workload-aware distance. Gaussian noise $\mathcal{N}(0, \sigma^2)$ is added to the vote histogram. The sensitivity is 1, since each `guid` contributes exactly one vote.
3. *Rank-based selection:* the top N_{synth} candidates by noisy vote count are selected as the final synthetic population.

The workload-aware distance follows Swanberg et al.'s formulation:

$$d_W(x, c) = \sum_{i \in \text{cat}} w_i \cdot \mathbf{1}[x_i \neq c_i] + \sum_{j \in \text{num}} |x_j - c_j|, \tag{5}$$

where w_i weights categorical features by query frequency (e.g., `chassistype` appears in 6 queries and receives weight 6, `countryname` in 4 queries receives weight 4) and numeric features are min-max normalized before computing L1 distance.

For $T = 1$ iteration, the privacy analysis reduces to a single Gaussian mechanism. We calibrate σ via the analytic Gaussian mechanism of Balle and Wang (2018) to achieve $\varepsilon = 4.0$ at $\delta = 10^{-5}$, yielding $\sigma \approx 1.08$. The total generation budget requires $N_{\text{synth}} \times L = 150{,}000$ API calls at batch size 20 (7,500 batches), submitted through OpenAI's Batch API at 50% reduced cost.

Given Swanberg et al.'s negative result on tabular PE (API access did not improve over marginal-based baselines), we treat this comparison as an empirical test of whether PE's advantages in the image and text domains transfer to heterogeneous, sparse telemetry data.

Algorithm 2: Private Evolution for tabular data ($T = 1$)

Input : Real table $X \in \mathbb{R}^{n \times d}$, target $\varepsilon^\star$, δ, population size N_{synth}, variation factor L
Output: Synthetic dataset S of size N_{synth}
Calibrate σ via analytic Gaussian mechanism (Balle and Wang 2018) for $(\varepsilon^\star, \delta, T{=}1)$;
$S_0 \leftarrow \texttt{RANDOM\_API}(N_{\text{synth}} \times L)$; `// generate candidates via LLM`
for $i = 1, \ldots, n$ **do**
  $j_i^\star \leftarrow \arg\min_j \, d_W(x_i, S_0^{(j)})$; `// nearest neighbor under workload distance`
$h_j \leftarrow |\{i : j_i^\star = j\}|$ for each candidate j; `// vote histogram`
$\tilde{h}_j \leftarrow h_j + \mathcal{N}(0, \sigma^2)$ for each j; `// add calibrated Gaussian noise`
$S \leftarrow$ top N_{synth} candidates ranked by $\tilde{h}_j$; `// rank-based selection`
return S

3.3 Per-table baselines

To isolate the sparsity artifact of the wide-table approach, we implement an independent per-table synthesis baseline. Each of the 19 reporting tables is synthesized separately at $\varepsilon = 4.0$ per table.

Two tables receive dedicated DP-VAE treatment. The `system_sysinfo_unique_normalized` table (1,000,000 rows, 9 categorical and 1 numeric column) uses a VAE with encoder $d_{\text{in}} \to 256 \to 128 \to (32\mu, 32 \log \sigma^2)$, batch size 4,096, 15 epochs, Adam (lr $= 10^{-3}$), KL weight 0.1, trained with Opacus at $\varepsilon = 4.0$, $\delta = 10^{-5}$. Categoricals are top-50 binned and one-hot encoded. The `system_cpu_metadata` table (1,000,000 rows, 5 categorical and 1 numeric column) uses the same architecture.

The remaining 17 tables use DP histogram synthesis. For each table, we compute the join rate (fraction of anchor `guids` with data) and generate proportionally many synthetic rows. Continuous columns are discretized into 20 equal-width bins spanning the 1st to 99th percentile, with Laplace noise (scale $= 1/\varepsilon_c$) added to bin counts and $\varepsilon_c = \varepsilon/k$ split uniformly across k columns. Categorical columns receive analogous noisy histogram sampling.

This approach sacrifices cross-table correlations by design. Synthetic `guids` in one table bear no relationship to those in another. It does avoid the zero-inflation problem because each table-level model trains only on `guids` with observed data, so zeros do not dominate the input distribution. Under basic composition, the total privacy budget is $19 \times 4.0 = 76.0$,

which is substantially weaker than the wide-table guarantee of $\varepsilon = 4.0$. This is a known limitation of per-table synthesis. Tighter accounting may reduce the effective budget and is left for future work.

Per-table synthesis also serves as a diagnostic. If it outperforms the wide-table approach on single-table queries but fails on multi-table joins, that supports the sparsity hypothesis.

3.3.1 MST baseline

As a non-neural baseline, we implement the Maximum Spanning Tree (MST) algorithm (McKenna, Miklau and Sheldon 2021) using the SmartNoise Synth library (OpenDP Contributors 2024). MST is a marginal-based method that operates in three phases:

1. *Marginal selection:* compute all pairwise column marginals and select a spanning tree of high-mutual-information pairs using the exponential mechanism.
2. *Noisy measurement:* estimate the selected marginals with calibrated Gaussian noise, splitting the privacy budget across all selected measurements.
3. *Synthetic generation:* fit a probabilistic graphical model (Private-PGM) to the noisy marginals and sample synthetic records consistent with the estimated distributions.

MST requires all columns to have finite cardinality. We discretize continuous columns into 20 quantile bins (computed on the nonzero values only, with a separate zero bin when present) and cap categorical columns at 50 categories (the same top-k binning used for the DP-VAE). After synthesis, continuous values are reconstructed by sampling uniformly within each bin's range.

Like the per-table DP-SGD approach, MST synthesizes each of the 19 reporting tables independently at $\varepsilon = 4.0$ per table, $\delta = 10^{-5}$, with guid assignment proportional to each table's real join rate against the sysinfo anchor. The total privacy budget under basic composition is $19 \times 4.0 = 76.0$, identical to the per-table DP-SGD baseline. Marginal-based methods represent the current state of the art for tabular DP synthesis (Swanberg et al. 2025) and provide the reference point against which both DP-SGD and PE should be measured.

3.4 Privacy budget summary

Table 4 compares the privacy parameters across all four synthesis methods. The wide-table DP-VAE and PE operate on a single model or mechanism, yielding a total budget equal to the per-unit budget. The per-table methods apply independent synthesis to each of the 19 reporting tables, so their total budget under basic composition is $19 \times \varepsilon = 76.0$ (we use this as a conservative bound, in the sense that it is optimizable in future work).

Table 4: Privacy parameters across synthesis methods.

Method	**Per-unit ε**	**δ**	**Mechanism**	**Composition**	**Total ε**
Wide-table DP-VAE	4.0	10^{-5}	DP-SGD (RDP)	single model	3.996
Per-table DP-SGD	4.0	10^{-5}	DP-SGD + Laplace	19× basic	76.0
MST	4.0	10^{-5}	Gaussian + Exp. mech.	19× basic	76.0
Private Evolution	4.0	10^{-5}	Gaussian (analytic)	T=1, single	4.0

4 Results

4.1 Evaluation metrics

We evaluate synthetic data quality by first assigning each query to one of five query categories, then routing it to the corresponding metric and pass rule. The categories are:

- Agg+Join, aggregate statistics computed after joining multiple tables on `guid` (e.g., `ChassisAgg`, `NetAsymAgg`).
- Geo/Demo, grouped summaries by geography or demographics such as country, CPU family, or generation (e.g., `XeonGeo`, `BatteryDemo`).
- Top-k, ranked outputs produced by window functions that return the highest-k items per group (e.g., `BrowserRank`, `WaitRank`).
- Pivot, wide multi-column summaries that represent joint behavior across many dimensions (e.g., `PersonaPivot`, `SleepPivot`).
- Histogram, distributional summaries over categories or bins (e.g., `BrowserHist`, `RamHist`).

For aggregate queries that produce numeric columns grouped by categorical keys, we compute the median relative error across all overlapping groups:

$$\mathrm{RE}(r,s) = |r-s|/|r|. \tag{6}$$

For queries that produce categorical distributions or histograms, we compute total variation distance between normalized real and synthetic distributions:

$$\mathrm{TV}(p,q) = \frac{1}{2}\sum_i |p_i - q_i|. \tag{7}$$

For ranked result sets, we compute Spearman's rank correlation coefficient on overlapping items. Group coverage is assessed via Jaccard similarity between real and synthetic group keys.

We route Agg+Join and Geo/Demo queries to RE, Histogram and Pivot queries to TV, and Top-k queries to Spearman ρ. The default pass thresholds are RE ≤ 0.25, TV ≤ 0.15, and $\rho \geq 0.5$. These are benchmark policy thresholds chosen to enforce moderate fidelity across metric types while remaining realistic under differential privacy noise.

The query score is the fraction of metric columns that pass. A query is marked as passing when its score is at least 0.5, with query-specific overlap thresholds defined in the evaluation metadata. We also test threshold sensitivity in Section 5 to verify that comparative conclusions are not driven by a single cutoff choice.

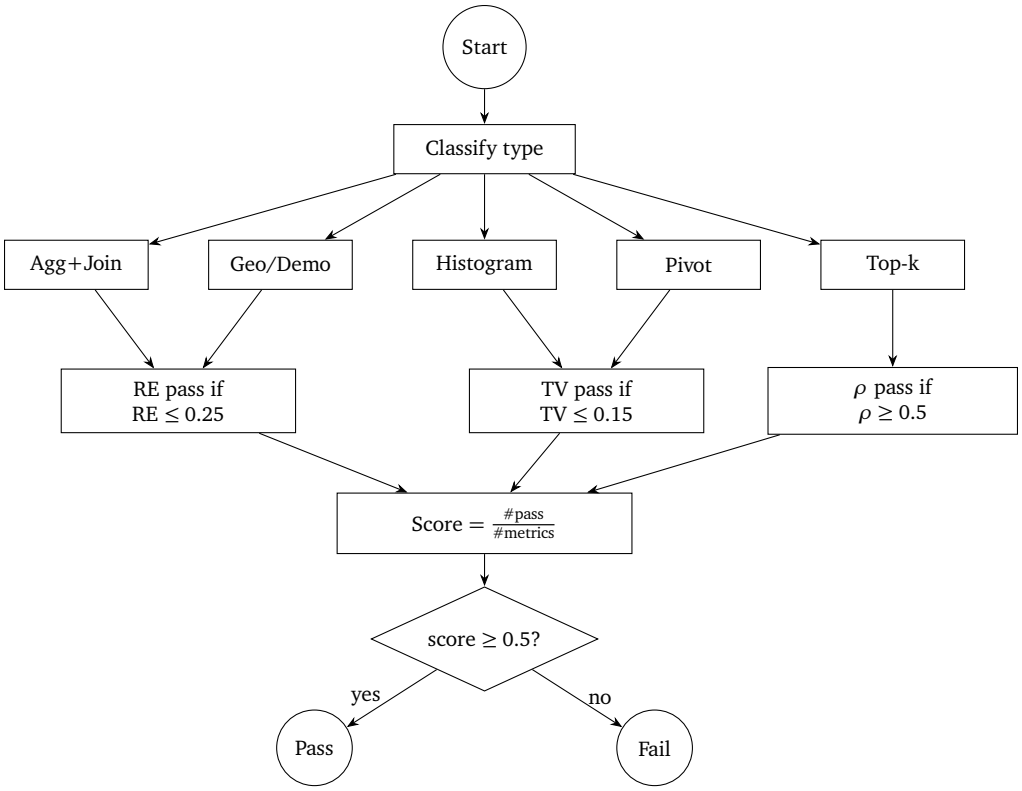

Figure 1: Metric routing and pass/fail logic used for benchmark evaluation.

4.2 DP-SGD and VAE results

We evaluate the wide-table DP-VAE synthetic data on 8 of 21 feasible benchmark queries. The remaining 13 queries require per-row reporting columns that are not represented in the wide table. The model passes 1 of 8 queries with average score 0.258 (Table 12).

To foreground per-query fidelity, Table 5 reports the 0–1 scores for the wide-table DP-VAE subset. Scores are fractions of metric-level passes. A score of 0.5 means that half of the evaluated metrics for that query satisfy threshold.

Table 5: Wide-table DP-VAE query scores on a 0–1 fidelity scale, color-coded as in Table 13: ≥ 0.5, 0.25–0.49, < 0.25.

Query codename	Type	Wide DP-VAE score
`BrowserRank`	Top-k	1.000
`PowerGeo`	Geo/Demo	0.333
`XeonGeo`	Geo/Demo	0.250
`BatteryGeo`	Geo/Demo	0.250
`ChassisAgg`	Agg+Join	0.167
`PersonaPivot`	Pivot	0.065
`BrowserHist`	Histogram	0.000
`RamHist`	Histogram	0.000

The strongest behavior is top-k categorical ranking. On `BrowserRank`, the model reaches score 1.000 and recovers 42 of 50 top-browser assignments on overlapping countries (84% categorical accuracy).

Mediocre behavior appears in several geographic queries with scores in the 0.250–0.333 range (`PowerGeo`, `XeonGeo`, and `BatteryGeo`). These are partial matches where some groups and metrics are correct, but not enough to pass.

The weakest behavior is on continuous-value workloads and join-heavy queries. In `ChassisAgg`, group overlap is moderate (Jaccard 0.571), but numeric accuracy is poor (median relative error 1161.44 for system counts, and relative error near 1.0 for the four hardware metrics). This indicates two failures at once, mismatched groups and incorrect values within overlapping groups.

Across aggregate and distribution workloads that depend on continuous magnitudes (power, temperature, frequency, network bytes, memory utilization, battery duration), relative errors commonly exceed 99%. Table 6 summarizes representative metric columns from this failure mode.

Table 6: Real vs. synthetic values for representative continuous metrics across evaluated workloads. All synthetic values are near zero.

Metric	**Real**	**Synthetic**	**Rel. error**
Platform power (`avg_psys_rap_watts`)	4.2914	0.0020	$> 99\%$
Package C0 (`avg_pkg_c0`, %)	42.6308	0.0222	$> 99\%$
Average frequency (`avg_freq_mhz`)	2,692.871	0.0070	$> 99\%$
Temperature (`avg_temp_centigrade`)	44.7122	0.0029	$> 99\%$
Network bytes received (`avg_bytes_received`)	7.359645×10^{16}	1.1304	$> 99\%$
Memory used (`avg_percentage_used`, %)	42.5682	0.0000	100%
Battery duration (`avg_duration`, min)	143.9710	0.1121	$> 99\%$

This collapse is driven by extreme zero-inflation in the wide table. Most metric columns are nonzero for only a small fraction of the 1,000,000 `guids`, because each event table covers a different subset of devices:

Table 7: Nonzero coverage per metric in the wide table. The PSYS RAP, frequency, and temperature metrics have data for fewer than 0.1% of `guids`.

Metric	**Nonzero guids**	**Sparsity**
Platform power (`avg_psys_rap_watts`)	816	99.9%
Average frequency (`avg_freq_mhz`)	613	99.9%
Temperature (`avg_temp_centigrade`)	622	99.9%
Package C0 (`avg_pkg_c0`)	8,943	99.1%
Network bytes received (`avg_bytes_received`)	37,224	96.3%
Memory used (`avg_percentage_used`)	69,552	93.0%

This failure follows from the combination of sparse data and the VAE objective. The model is trained with mean squared error on continuous columns. When 93–99.9% of values are zero, the easiest way to reduce loss is to predict values close to zero for most rows. The KL regularization term then pushes latent representations toward the prior, which makes it harder to preserve rare nonzero patterns. As a result, columns that depend on uncommon nonzero values are poorly reconstructed.

A secondary effect is inflated counts in join-heavy queries. In `ChassisAgg`, the real data returns 104 `guids` that appear in all five metric tables. The synthetic data returns more than 163,000 matches. This happens because many rows receive small positive values, so INNER JOIN conditions that should filter heavily become much less selective. This is a structural mismatch, and it reduces fidelity for multi-table aggregates.

Overall, the wide-table DP-VAE preserves one useful signal, rank ordering among frequent categorical groups. It does not preserve continuous magnitudes or strict join behavior, because rare nonzero support is not learned well (see the calibration scatter in Figure 4). This motivates the per-table strategy in the next subsection. Per-table synthesis does not preserve cross-table coupling, but it aims to improve continuous distributions.

4.3 Per-table DP-SGD results

As described in Section 3.3, this baseline synthesizes each reporting table independently at the same per-table privacy budget, using DP-VAE for anchor tables and DP histogram synthesis for the remaining tables. The goal is to improve within-table fidelity under sparse supports, at the cost of cross-table consistency.

Per-table DP-SGD passes 6 of 21 queries, with average score 0.303 and median score 0.250. Unlike wide-table DP-VAE and PE, which are evaluated on 8 queries due to wide-table reconstruction limits, per-table DP-SGD is evaluated on the full 21-query benchmark. Table 8 reports all 21 per-query scores.

Table 8: Per-table DP-SGD fidelity on all 21 benchmark queries.

Query codename	Type	Per-table score
`VendorAgg`	Agg+Join	1.000
`BatteryGeo`	Geo/Demo	1.000
`BrowserRank`	Top-k	1.000
`SleepPivot`	Pivot	0.636
`RamHist`	Histogram	0.500
`DetectionRank`	Top-k	0.500
`PersonaPivot`	Pivot	0.419
`PowerGeo`	Geo/Demo	0.333
`BrowserHist`	Histogram	0.333

Continued on next page

Query codename	Type	Per-table score
BlockerAgg	Agg+Join	0.250
XeonGeo	Geo/Demo	0.250
NetAsymAgg	Agg+Join	0.143
BlockerDurAgg	Agg+Join	0.000
ChassisAgg	Agg+Join	0.000
DisplayAgg	Agg+Join	0.000
BatteryDemo	Geo/Demo	0.000
FocalRank	Top-k	0.000
SystemRank	Top-k	0.000
WaitModeRank	Top-k	0.000
WaitRank	Top-k	0.000
WaitStateRank	Top-k	0.000

This method performs best on several continuous-valued summaries. For example, in SleepPivot, on-time and off-time remain close to real values (RE 0.13 and 0.15). It remains weak on join-heavy queries because independently synthesized tables do not preserve cross-table correlations by guid.

4.4 MST results

As described in Section 3.3.1, MST is a marginal-based baseline that discretizes continuous columns, fits a private maximum-spanning-tree graphical model per table, and then samples synthetic rows from that model. Like per-table DP-SGD, it preserves table-level structure better than cross-table joins.

MST passes 6 of 21 queries, with average score 0.328 and median score 0.250. Like per-table DP-SGD, MST is evaluated on the full 21-query benchmark, while wide-table DP-VAE and PE remain limited to 8 queries. Table 9 reports all 21 per-query scores.

Table 9: MST fidelity on all 21 benchmark queries.

Query codename	Type	MST score
VendorAgg	Agg+Join	1.000
BrowserRank	Top-k	1.000
DetectionRank	Top-k	1.000
SystemRank	Top-k	1.000
BrowserHist	Histogram	0.667
BatteryGeo	Geo/Demo	0.500
NetAsymAgg	Agg+Join	0.429
PowerGeo	Geo/Demo	0.333
SleepPivot	Pivot	0.273

Continued on next page

Query codename	Type	MST score
BlockerAgg	Agg+Join	0.250
XeonGeo	Geo/Demo	0.250
PersonaPivot	Pivot	0.194
BlockerDurAgg	Agg+Join	0.000
ChassisAgg	Agg+Join	0.000
DisplayAgg	Agg+Join	0.000
BatteryDemo	Geo/Demo	0.000
RamHist	Histogram	0.000
FocalRank	Top-k	0.000
WaitModeRank	Top-k	0.000
WaitRank	Top-k	0.000
WaitStateRank	Top-k	0.000

MST is strongest (albeit inconsistently) on ranking-style behavior. It attains perfect ranking agreement on several application ranking queries in the full 21-query workload. It is weaker on some continuous-value aggregates, where discretization can distort magnitudes even when category-level structure is preserved.

4.5 Private Evolution results

As described in Section 3.2, PE is a training-free pipeline. It generates candidate records with a foundation model API, applies a differentially private nearest-neighbor histogram over real records, and keeps the highest-ranked candidates.

PE passes 2 of 8 evaluated queries, with average score 0.150 and median score 0.000. Table 10 reports PE fidelity.

Table 10: Private Evolution fidelity on the shared 8-query subset.

Query	Type	PE score
BrowserHist	Histogram	0.667
RamHist	Histogram	0.500
PersonaPivot	Pivot	0.032
ChassisAgg	Agg+Join	0.000
BatteryGeo	Geo/Demo	0.000
XeonGeo	Geo/Demo	0.000
PowerGeo	Geo/Demo	0.000
BrowserRank	Top-k	0.000

PE completes generation of 150,000 candidates via the OpenAI Batch API (gpt-5-nano), followed by DP nearest-neighbor histogram selection ($\sigma \approx 1.08$) to produce 50,000 records. The final privacy expenditure is $\varepsilon = 4.0$ at $\delta = 10^{-5}$.

PE remains weak on aggregate and geographic queries. In `BrowserRank`, PE generates only 15 of 51 real countries and gets 13 correct among those 15, which leads to poor overall score despite good local matches. `PersonaPivot` also remains low (0.032).

Category support mismatch is a key reason for this behavior. Country support is the largest gap. PE covers 18 of 51 real countries (35.3% coverage), and hallucinates extra categories not present in real telemetry data; this issue is discussed further in Section 5.

Table 11: Category support overlap between real data and PE synthetic data.

Column	Real unique	PE unique	Overlap	Coverage of real	Extra in PE
`countryname_normalized`	51	30	18	35.3%	12
`chassistype`	7	88	7	100.0%	81
`os`	7	19	7	100.0%	12
`persona`	11	60	10	90.9%	50

4.6 Cross-method comparison

Across methods, pass counts are similar for per-table DP-SGD and MST, while wide-table DP-SGD and PE evaluate fewer queries because of schema coverage limits (Table 12).

Table 12: Benchmark performance across synthesis methods. "Passed" counts queries with score ≥ 0.5.

Method	Evaluated	Passed	Pass rate	Avg score	Median score
Wide-table DP-SGD	8	1	12.5%	0.258	0.208
Per-table DP-SGD	21	6	28.6%	0.303	0.250
MST baseline	21	6	28.6%	0.328	0.250
Private Evolution	8	2	25.0%	0.150	0.000

Figure 2 complements this table with pass-count and average-score bars, and Figure 3 breaks average score down by query type.

Table 13: Per-query scores for all four synthesis methods, color-coded by fidelity: ≥ 0.5 (pass), 0.25–0.49, < 0.25, N/A.

Query	Type	Wide	Per-table	MST	PE
`BlockerAgg`	Agg+Join	—	0.250	0.250	—

Continued on next page

Query	Type	Wide	Per-table	MST	PE
BlockerDurAgg	Agg+Join	—	0.000	0.000	—
ChassisAgg	Agg+Join	0.167	0.000	0.000	0.000
DisplayAgg	Agg+Join	—	0.000	0.000	—
NetAsymAgg	Agg+Join	—	0.143	0.429	—
VendorAgg	Agg+Join	—	1.000	1.000	—
BatteryDemo	Geo/Demo	—	0.000	0.000	—
BatteryGeo	Geo/Demo	0.250	1.000	0.500	0.000
PowerGeo	Geo/Demo	0.333	0.333	0.333	0.000
XeonGeo	Geo/Demo	0.250	0.250	0.250	0.000
BrowserHist	Histogram	0.000	0.333	0.667	0.667
RamHist	Histogram	0.000	0.500	0.000	0.500
PersonaPivot	Pivot	0.065	0.419	0.194	0.032
SleepPivot	Pivot	—	0.636	0.273	—
BrowserRank	Top-k	1.000	1.000	1.000	0.000
DetectionRank	Top-k	—	0.500	1.000	—
FocalRank	Top-k	—	0.000	0.000	—
SystemRank	Top-k	—	0.000	1.000	—
WaitModeRank	Top-k	—	0.000	0.000	—
WaitRank	Top-k	—	0.000	0.000	—
WaitStateRank	Top-k	—	0.000	0.000	—

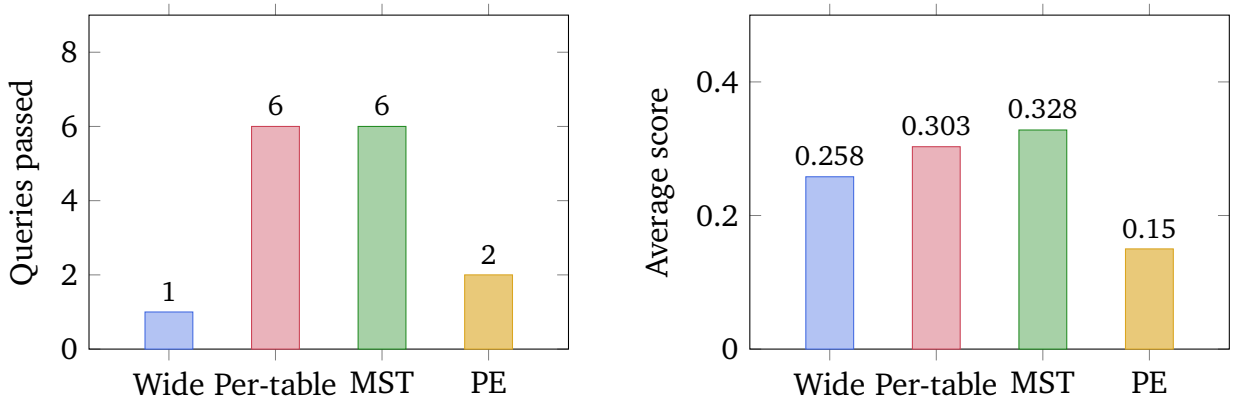

Figure 2: Left: number of queries passing (score ≥ 0.5) for each method. Right: average score across all evaluated queries. The per-table methods evaluate all 21 queries; the wide-table and PE methods evaluate 8.

The per-query scores in Table 13 show that no method achieves high scores on aggregate queries involving continuous metrics, while categorical and ranking queries show more variation across methods.

Figure 3 breaks down average scores by query type. The per-table methods achieve their strongest results on pivot and histogram queries, where the independent per-table approach avoids the zero-inflation that cripples the wide-table method. Aggregate queries involving

multi-table joins remain the hardest category for all methods, with average scores below 0.3.

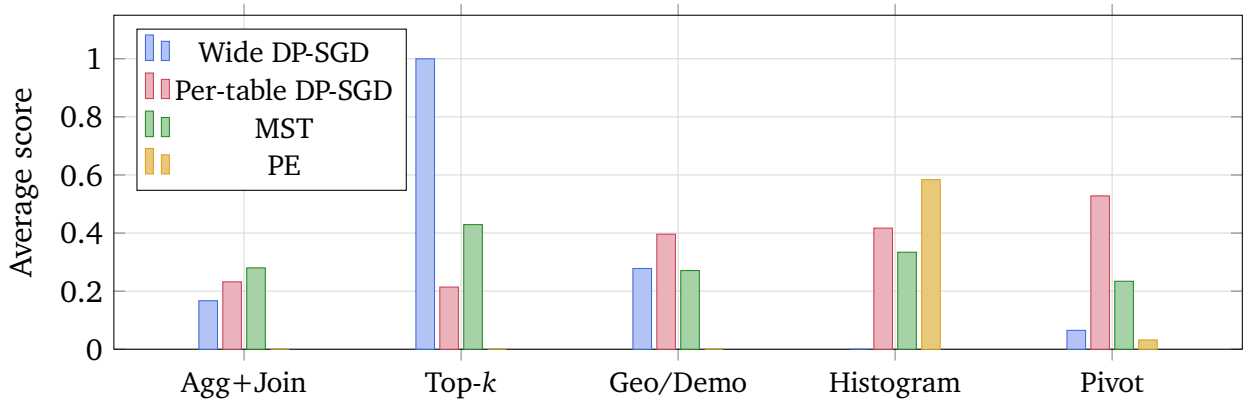

Figure 3: Average query score by query type across synthesis methods.

Figure 4 provides a cross-method calibration view. Each point is a matched group-level value from query outputs, transformed as $\log_{10}(1+x)$. Points near the diagonal indicate better calibration.

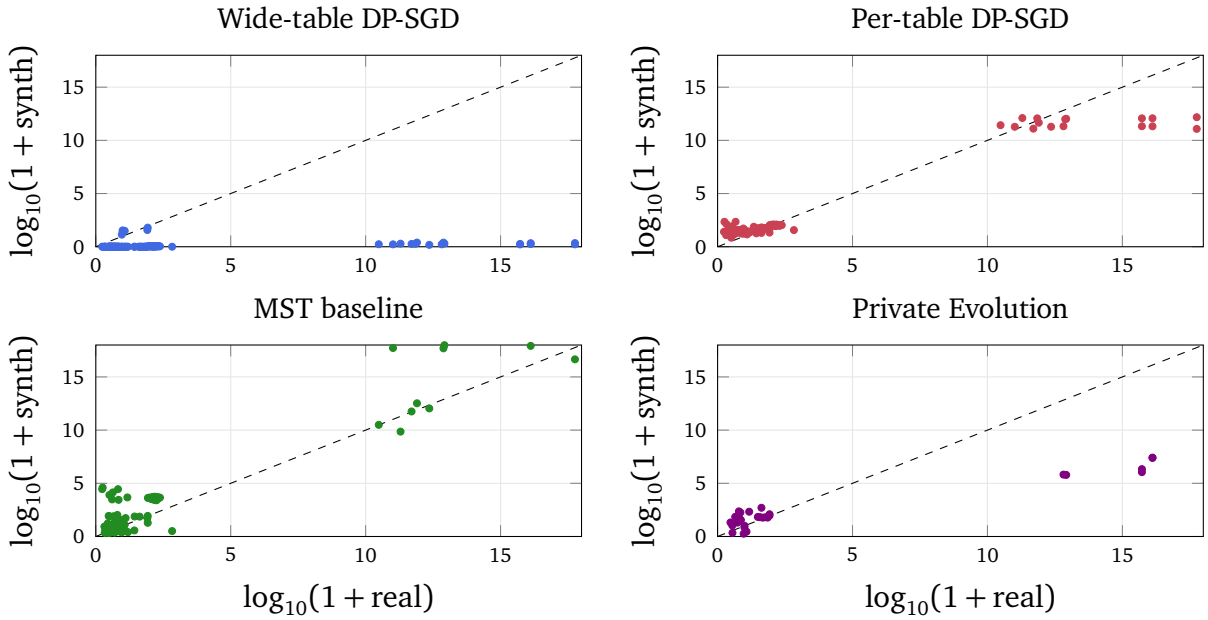

Figure 4: Calibration scatter by method using matched group-level query outputs. The dashed line is perfect calibration.

The calibration patterns differ by method. Wide-table DP-SGD shows strong collapse toward near-zero synthetic outputs, with many points concentrated at low $\log_{10}(1+\text{synth})$ even when real values are large. Per-table DP-SGD and PE both show underestimation with horizontal banding, where outputs plateau at a few repeated synthetic levels rather than

tracking continuous increases in the real values. MST has wider spread and several outliers, including high-value deviations, but its points follow the diagonal more closely across a broader range. This is consistent with MST being noisier pointwise while remaining relatively calibrated overall.

4.7 Additive experiments

We ran a second round of additive experiments to test whether lightweight interventions can improve utility without changing any of the original methods.

The first intervention was post-processing on top of existing synthetic reporting tables. We enforced numeric constraints and canonicalized selected categorical values to real-data vocabularies. This intervention produced no measurable gain. Per-table DP-SGD, MST, and PE all had unchanged pass counts and unchanged average scores.

The second intervention was workload routing. This intervention does not generate new synthetic data. It selects query outputs from the method that performs better for that query type. We routed aggregate and histogram queries to per-table DP-SGD, then routed distribution and ranking queries to MST. This lifted the benchmark from 6 of 21 passed queries to 8 of 21 with an average score of 0.404.

Table 14: Additive follow-up experiment results. The routing policy combines existing query outputs by query type and improves overall benchmark utility.

Run	**Evaluated**	**Passed**	**Pass rate**	**Avg score**
Per-table DP-SGD baseline	21	6	28.6%	0.303
MST baseline	21	6	28.6%	0.328
Postprocessed per-table DP-SGD	21	6	28.6%	0.303
Postprocessed MST	21	6	28.6%	0.328
Hybrid routed outputs	21	8	38.1%	0.404

5 Discussion

The four-method comparison provides answers to several of our research questions.

Research question (1), which method achieves higher benchmark scores, has a nuanced answer. Per-table DP-SGD and MST both pass 6 of 21 queries (28.6%), with MST achieving a slightly higher average score (0.328 vs. 0.303). The wide-table DP-VAE passes 1 of 8 evaluated queries (12.5%). PE passes 2 of 8 (25.0%), outperforming the wide-table DP-VAE on the same query set but with a lower average score (0.150 vs. 0.258). No method comes close to passing all queries. The two per-table methods pass different query subsets: per-table DP-SGD passes `SleepPivot` (score 0.64) and `RamHist` (0.50), while MST passes `BrowserHist` (0.67) and two application ranking queries (1.00 each). PE's two passing queries (`BrowserHist` at 0.667 and `RamHist` at 0.500) overlap with the per-table methods, indicating that the DP histogram selection can correct distributional biases in the LLM output for histogram-type queries. This complementarity reflects fundamental algorithmic differences. DP histogram synthesis preserves continuous distributions by sampling from noisy bin counts, yielding low relative error on time-based metrics (on-time RE = 0.13, sleep-time RE = 0.04). MST preserves pairwise marginals between categorical keys and count columns, maintaining rankings (Spearman $\rho = 1.00$ on application system counts) even when absolute values are distorted. Neither algorithm achieves both properties simultaneously.

Research question (2), whether error compounds across multi-table joins, receives a clear affirmative. `ChassisAgg` requires data from five independently synthesized tables. Both per-table methods score 0.00 because the synthetic tables share no correlated `guids`; the inner join produces empty results (per-table DP-SGD) or random associations (MST). The wide-table DP-VAE scores 0.17 on this query, the only method that preserves any cross-table structure, but the values are near-zero due to sparsity. `NetAsymAgg` (a 2-way join) similarly suffers: both per-table methods overcount matching rows by 5–7× and report network byte values with RE $\approx$ 1.0, because the independently synthesized network and sysinfo tables produce spurious `guid` overlaps.

Research question (3), minority class preservation, is partially addressed. `BrowserRank` demonstrates that all methods preserve dominant categorical associations: per-table DP-SGD achieves 47 of 50 correct top browsers per country, and the wide-table DP-VAE achieves 42 of 50. For rare categories, per-table DP-SGD inflates minority chassis types ("Intel NUC/STK" at 8.4% synthetic vs. 2.0% real) due to the top-50 binning and uniform noise in DP-SGD, while MST underrepresents them (generating too few entries for rare combinations). Both methods underperform on queries involving rare process names or application-specific rankings.

Research question (4), classifier comparability, remains unaddressed. The benchmark queries are analytical (aggregate statistics, rankings, distributions), not predictive, so training downstream classifiers on the synthetic data was not a natural fit for this evaluation framework. A direct comparison would require defining a classification task on the DCA data (e.g., predicting chassis type from usage patterns), training models on real versus synthetic data, and comparing test accuracy. We defer this to future work.

Research question (5), wall-clock time, shows clear differences. The wide-table DP-VAE trains in 360 minutes on CPU (20 epochs, 1M rows, 307 features). Per-table synthesis completes in approximately 90 minutes total (two VAEs at ∼30 minutes each for sysinfo and cpu_metadata, plus histogram synthesis for 17 smaller tables). MST runs in under 10 minutes per table using the `mst` library. PE completes in 59 minutes total: the OpenAI Batch API generates 150,000 candidate records (7,500 batch calls at approximately $20 USD), and the DP nearest-neighbor histogram over 1,000,000 real × 150,000 synthetic records accounts for the remaining computation.

The PE results confirm that foundation models import their own distributional priors rather than learning the target distribution (Win11 at 77.8% vs. 10.4% real). The DP nearest-neighbor histogram partially corrects these biases for histogram-type queries (PE matches MST on browser usage at 0.667 and outperforms the wide-table DP-VAE on RAM utilization at 0.500 vs. 0.000), but cannot compensate for the near-total absence of nonzero continuous metrics in the LLM output. The hallucinated categorical values further reduce effective sample size by generating records that cannot match any real data point in the nearest-neighbor histogram. This aligns with Swanberg et al.'s finding that LLMs capture 1-way marginals reasonably but are inaccurate on k-way marginals.

Several mitigation strategies warrant investigation:

- A two-stage model in which a Bernoulli model predicts zero vs. nonzero per column, followed by a conditional model for nonzero values only.
- Relational DP synthesis methods that handle multi-table structure directly, eliminating the zero-inflation problem while preserving cross-table correlations.
- Method ensembling: using per-table DP-SGD for queries requiring continuous accuracy and MST for queries requiring ranking preservation, selecting per query type.
- Post-processing the synthetic data with mode patching and constraint enforcement, as proposed by recent work on model-agnostic DP post-processing.

5.1 Sensitivity to evaluation thresholds

Figure 5 reports how pass rates change as the relative error, total variation, and Spearman ρ thresholds vary. The analysis confirms that our findings are not artifacts of a particular threshold choice.

The relative error threshold has the largest effect. Per-table DP-SGD's pass rate increases from 14.3% at RE ≤ 0.05 to 57.1% at RE ≤ 1.0, indicating many queries produce synthetic values in the right order of magnitude but outside the strict 25% tolerance. MST is more robust to strict thresholds: it starts at 23.8% at RE ≤ 0.05 (compared to 14.3% for per-table), reflecting its strength on categorical distributions where exact values are less important than distributional shape. The wide-table method is flat at 12.5% until RE ≤ 1.0, at which point it jumps to 75%, confirming that the continuous metric failure is total (relative errors exceed 99%) rather than marginal.

Total variation and Spearman ρ thresholds have minimal effect on pass rates across all methods. TV sensitivity is limited because the distribution queries either match closely (TV

< 0.10) or fail badly (TV > 0.5), with few queries in the boundary region. Rank correlation sensitivity is limited because MST achieves perfect $\rho = 1.0$ on its passing ranking queries while both methods score near zero on failing ones, leaving no queries sensitive to the threshold.

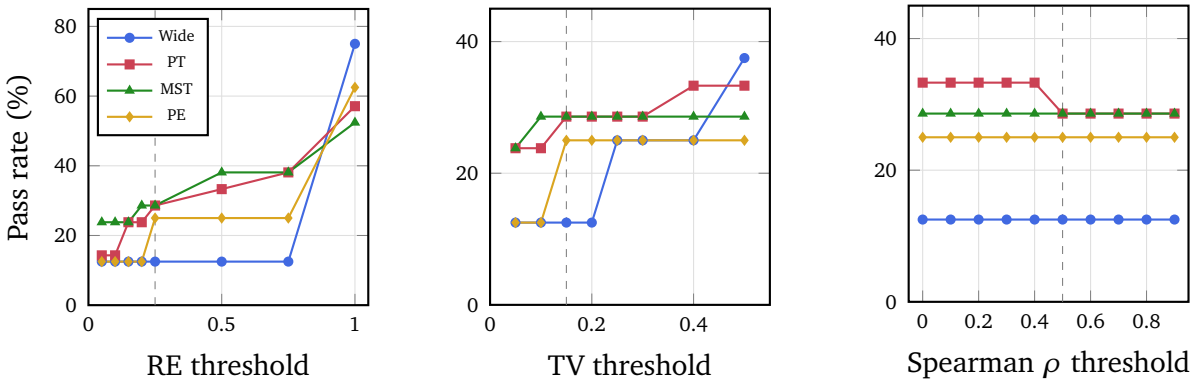

Figure 5: Pass rate sensitivity to evaluation thresholds. Left: relative error threshold (TV = 0.15, $\rho = 0.5$ fixed). Center: total variation threshold (RE = 0.25, $\rho = 0.5$ fixed). Right: Spearman ρ threshold (RE = 0.25, TV = 0.15 fixed). Dashed gray lines indicate the default thresholds.

6 Conclusion

We presented an evaluation framework for differentially private synthetic data generation on Intel's DCA telemetry corpus, implementing four synthesis approaches: a wide-table DP-VAE trained with DP-SGD, per-table DP histogram synthesis, MST marginal-based synthesis, and Private Evolution via foundation model APIs.

All four methods have completed evaluation against the 21-query benchmark. Per-table DP-SGD and MST each pass 6 of 21 queries (28.6% pass rate), passing different subsets that reflect their algorithmic strengths: DP histogram synthesis preserves continuous distributions (on-time RE = 0.13) while MST preserves rankings (Spearman $\rho = 1.00$ on application counts). The wide-table DP-VAE passes 1 of 8 evaluated queries (12.5%), with all continuous metrics near zero due to extreme sparsity in the wide-table representation. PE passes 2 of 8 (25.0%), outperforming the wide-table DP-VAE in pass count but with a lower average score (0.150 vs. 0.258). PE's two passing queries (`BrowserHist` and `RamHist`) are histogram-type queries where the DP nearest-neighbor selection can correct the foundation model's distributional biases.

Two structural findings emerge. First, wide-table sparsity, not the choice of synthesis algorithm, is the dominant failure mode. The wide table has 93–99.9% zero values in metric columns, and both the VAE and the foundation model collapse these to their mode. Second, per-table independence destroys cross-table correlations: the 5-way chassis join produces empty or random results under both per-table methods, while the wide-table approaches (DP-VAE and PE) at least preserve some cross-table structure.

PE confirms the negative result of Swanberg et al. (2025): API access to foundation models does not improve DP tabular synthesis beyond established baselines on the DCA corpus. The foundation model imports its own distributional priors (Win11 at 77.8% vs. 10.4% real), hallucinates categorical values outside the real vocabulary, and generates near-zero continuous metrics. The DP histogram partially corrects distributional queries but cannot compensate for the absence of signal in aggregate and geographic queries.

These results point toward relational DP synthesis as the most promising direction: methods that handle multi-table structure directly could eliminate zero-inflation while preserving cross-table correlations. A secondary direction is method ensembling, selecting per-table DP-SGD for continuous-valued queries and MST for ranking queries, capitalizing on their complementary strengths.

References

Abadi, Martin, 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*. [Link]

Balle, Borja, and Yu-Xiang Wang. 2018. "Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising." In *Proceedings of the 35th International Conference on Machine Learning*. PMLR. [Link]

Dwork, Cynthia, and Aaron Roth. 2014. "The Algorithmic Foundations of Differential Privacy." *Foundations and Trends® in Theoretical Computer Science* 9 (3–4): 211–407. [Link]

Ghalebikesabi, Sahra, Leonard Berrada, Sven Gowal, Ira Ktena, Robert Stanforth, Jamie Hayes, Soham De, Samuel L Smith, Olivia Wiles, and Borja Balle. 2023. "Differentially private fusion models generate useful synthetic images." *arXiv preprint arXiv:2302.13861*

Lin, Zinan, Sivakanth Gopi, Janardhan Kulkarni, Harsha Nori, and Sergey Yekhanin. 2025. "Differentially Private Synthetic Data via Foundation Model APIs 1: Images." [Link]

McKenna, Ryan, Gerome Miklau, and Daniel Sheldon. 2021. "Winning the NIST Contest: A Scalable and General Approach to Differentially Private Synthetic Data." *Journal of Privacy and Confidentiality* 11 (3). [Link]

OpenDP Contributors. 2024. "SmartNoise SDK." [Link]

Swanberg, Marika, Ryan McKenna, Edo Roth, Albert Cheu, and Peter Kairouz. 2025. "Is API Access to LLMs Useful for Generating Private Synthetic Tabular Data?." [Link]

Xie, Chulin, Zinan Lin, Arturs Backurs, Sivakanth Gopi, Da Yu, Huseyin A Inan, Harsha Nori, Haotian Jiang, Huishuai Zhang, Yin Tat Lee, Bo Li, and Sergey Yekhanin. 2024. "Differentially Private Synthetic Data via Foundation Model APIs 2: Text." [Link]

Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2023. *Dive into Deep Learning*. Cambridge University Press. [Link]

A Project proposal

A.1 Abstract

Differentially private synthetic data generation enables the release of realistic datasets without exposing individual records. Two paradigms dominate current research: *training-based* methods such as differentially private stochastic gradient descent (DP-SGD), which inject noise during model optimization, and *training-free* methods such as Private Evolution (PE), which achieve privacy guarantees through inference-only access to pre-trained foundation models.

We propose to implement and rigorously compare both approaches on Intel's Driver and Client Applications (DCA) telemetry corpus, a multi-table dataset comprising system metadata, power consumption, application usage, and browsing behavior across thousands of Windows clients. Our evaluation centers on a benchmark of 22 analytical SQL queries representative of real business intelligence workloads, measuring whether synthetic data can substitute for real data in production analytics pipelines. Under matched privacy budgets, we assess query fidelity, statistical preservation, downstream utility, and computational cost to determine whether training-free methods can match training-based approaches while offering reduced implementation complexity.

A.2 Motivation

Organizations routinely collect detailed telemetry from their products that drive critical business decisions. Engineers use it to diagnose failures, product teams analyze usage trends, and analysts extract insights that shape product roadmaps. However, the same granularity that makes telemetry valuable also makes it sensitive. Browsing patterns, work schedules, and device fingerprints can re-identify specific individuals even after conventional anonymization.

Privacy regulations such as GDPR and CCPA, along with institutional policies, increasingly restrict how such data can be stored, shared, and analyzed. The resulting tension between data utility and privacy protection motivates the development of *synthetic data generation*, which produces artificial datasets that preserve the statistical properties necessary for analysis while providing formal guarantees that no individual's information can be recovered.

Two paradigms have emerged for generating synthetic data with rigorous privacy guarantees. *Training-based methods* optimize generative models under constraints that guarantee privacy (by adding noise during the training process to effectively "anonymize"). *Training-free methods* leverage pre-trained foundation models (e.g., ChatGPT, Claude, Gemini), achieving privacy through carefully designed queries rather than private optimization. Both approaches have demonstrated success in isolation, yet no comprehensive comparison

exists under controlled experimental conditions with realistic analytical workloads.

This project provides that comparison. We implement both paradigms and evaluate them on a real telemetry corpus using a benchmark of 22 SQL queries representative of production analytics, measuring which approach better preserves query fidelity under equivalent privacy budgets.

A.2.1 Technical context

Differential privacy (DP) provides the mathematical foundation for our privacy guarantees. Informally, a mechanism is differentially private if its output looks nearly the same whether or not any single individual's data is included. Formally, a mechanism M satisfies (ε, δ)-DP if for neighboring datasets $D \sim D'$ differing by one record:

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \Pr[M(D') \in S] + \delta. \tag{8}$$

The parameter ε controls the privacy-utility tradeoff, as smaller values mean stronger privacy but noisier outputs.

Two methodologies dominate differentially private synthetic data generation:

1. *Training-based (DP-SGD):* Train a generative model while adding noise to gradients at each step (Abadi et al. 2016). This is effective, but is computationally expensive and requires careful hyperparameter tuning.
2. *Training-free (Private Evolution):* Usw API access to pre-trained foundation models and treat them as black boxes, iteratively selecting and mutating candidates via differentially private histograms (Lin et al. 2025). This is much easier to deploy, but is theoretically more abstract.

Our central research question: *can training-free methods match the statistical fidelity of training-based approaches on real analytical workloads?*

A.3 Data and problem statement

A.3.1 Intel DCA telemetry data

Our investigation involves the Intel Driver and Client Applications (DCA) telemetry dataset, a large-scale collection of system-level signals from Windows client machines. Telemetry data is invaluable for product monitoring, performance optimization, and failure detection, yet contains sensitive behavioral patterns that could re-identify individual users. We describe the dataset structure and the analytical workload that synthetic data must preserve.

Schema overview The DCA telemetry corpus comprises of ≈ 30 interrelated tables, organized around a globally unique identifier (`guid`) assigned to each client system. Tables are indexed by `guid` and, where applicable, a date column (`dt`), The schema spans several categories:

- *Client metadata:* The `system_sysinfo_unique_normalized` table provides static attributes, like chassis type (notebook, desktop, 2-in-1, tablet), country, OEM, RAM capacity, processor family/generation, graphics card, screen size, and a derived *persona* classification (Gamer, Office/Productivity, Content Creator, etc.).
- *Power and thermal:* Multiple tables capture physical instrumentation:
  - `system_hw_pkg_power`: Processor power draw (mean, max) in Watts
  - `system_psys_rap_watts`: Total system power across AC/DC states
  - `system_pkg_temp_centigrade`: Processor temp. distributions by power state
  - `system_pkg_c0`: C0 state residency (percentage of time fully active)
  - `system_pkg_avg_freq_mhz`: Clock frequency statistics
- *Battery and mobility:* The `system_batt_dc_events` table summarizes battery utilization (duration on DC, battery percentage, power cycle frequency). Daily durations for the on/off/sleep states are also recorded.
- *Application behavior:* Foreground usage in `system_frgnd_apps_types` and a daily summary (process names, categories, focal time). The `system_userwait` table measures wait times for application starts.
- *Web browsing:* The `system_web_cat_pivot_*` family decomposes browsing across 28 categories (social, productivity, entertainment, gaming, etc.) by duration, page loads, and domain counts.
- *Network and memory:* The `system_network_consumption` table summarizes bytes sent/received; `system_memory_utilization` provides RAM usage statistics.
- *Display:* The `system_display_devices` table records connected displays (connection type, resolution, refresh rate, duration by AC/DC state).

Analytical query workload The practical utility of synthetic telemetry data is determined by its ability to support *real analytical workloads*. We operationalize this through a benchmark suite of 22 SQL queries representative of business intelligence tasks performed on the DCA corpus. These queries, developed by Intel analysts, span several classes, where the following are examples of such:

1. *Aggregate statistics with joins* involve weighted statistics across multiple tables joined on `guid`:
```
-- Average platform power, C0 residency, frequency, and
 ↪  temperature by chassis type
SELECT chassistype, COUNT(DISTINCT guid) AS n_systems,
       SUM(nrs * avg_psys_rap_watts) / SUM(nrs) AS avg_power,
       SUM(nrs * avg_pkg_c0) / SUM(nrs) AS avg_c0, ...
FROM system_sysinfo_unique_normalized a
INNER JOIN system_psys_rap_watts b ON a.guid = b.guid
INNER JOIN system_pkg_c0 c ON a.guid = c.guid ...
GROUP BY chassistype;
```
2. *Ranked top-k queries* involve window functions for ranked lists:
```
-- Top 10 applications per category by focal screen time
SELECT app_type, exe_name, avg_focal_sec_per_day,
       RANK() OVER (PARTITION BY app_type
                    ORDER BY avg_focal_sec_per_day DESC) AS rank
FROM ( ... ) WHERE rank <= 10;
```
3. *Geographic/demographic breakdowns* involve segmentation by country, processor generation, OEM, or persona:
```
-- Battery usage by country
SELECT countryname_normalized, COUNT(DISTINCT guid),
       AVG(num_power_ons), AVG(duration_mins)
FROM system_batt_dc_events a
INNER JOIN system_sysinfo_unique_normalized b ON a.guid = b.guid
GROUP BY countryname_normalized HAVING COUNT(DISTINCT guid) > 100;
```
4. *Histograms and distributions* involve binned aggregations testing distributional shape preservation:
```
-- RAM utilization by memory capacity
SELECT sysinfo_ram / 1024 AS ram_gb, COUNT(DISTINCT guid),
       ROUND(SUM(nrs * avg_percentage_used) / SUM(nrs)) AS avg_pct
FROM system_memory_utilization
WHERE avg_percentage_used > 0
GROUP BY sysinfo_ram ORDER BY sysinfo_ram;
```
5. *Complex multi-way pivots* involve high-dimensional joint distributions, e.g., web category usage percentages by persona across 28 browsing categories.

Formal benchmark definition Let $\mathcal{Q} = \{q_1, \ldots, q_{22}\}$ denote our SQL query benchmark. Each query q_j maps a database instance to a result set $q_j(D) \in \mathcal{R}_j$, where $\mathcal{R}_j$ may be a scalar, vector, or table. The *query discrepancy* for synthetic data $\tilde{D}$ is:

$$\Delta_j(D, \tilde{D}) = d_j(q_j(D), q_j(\tilde{D})), \tag{9}$$

where d_j is a distance metric appropriate to the result type (relative error for scalars, Spearman's ρ for rankings, total variation for histograms). The aggregate benchmark score is:

$$\text{Score}(\tilde{D}) = \frac{1}{22} \sum_{j=1}^{22} \mathbf{1}[\Delta_j(D, \tilde{D}) \leq \tau_j], \tag{10}$$

where τ_j is a query-specific tolerance. A synthetic dataset "passes" the benchmark if it achieves high scores, indicating analysts could substitute $\tilde{D}$ for D without materially affecting conclusions.

A.3.2 Problem statement

Formal setup Let $D = \{x_1, \ldots, x_n\}$ be a private dataset. Two datasets D, D' are *neighbors* if they differ by one record. A mechanism M satisfies (ε, δ)-differential privacy if for all neighbors $D \sim D'$ and all outputs S:

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \Pr[M(D') \in S] + \delta. \tag{11}$$

Our objective is to construct a synthetic data mechanism M producing $\tilde{D} = M(D)$ that satisfies (ε, δ)-DP while preserving utility (i.e., the synthetic distribution should approximate the real distribution sufficiently for downstream analytics).

Limitations of existing approaches Both paradigms exhibit known limitations:

- *DP-SGD limitations:*
  - Gradient clipping bias disproportionately affects rare classes
  - Noise accumulation degrades convergence over many iterations
  - Complex hyperparameter tuning (batch size, learning rate, clip norm, noise multiplier)
  - High computational cost (see Ghalebikesabi et al. (2023))
- *Private Evolution limitations:*
  - Utility depends on foundation model quality, and somes may underperform on specialized tabular data
  - Nearest-neighbor histograms assume embedding similarity correlates with distributional similarity
  - Limited theoretical understanding compared to optimization-based methods

Research questions Given these limitations and our 22-query SQL benchmark, we investigate:

1. Under matched (ε, δ), which method achieves higher scores on the SQL query benchmark? Which query types exhibit the largest discrepancy?
2. Does error compound across multi-table joins, or do synthetic data preserve joint distributions adequately?
3. Which method better preserves minority class frequencies ($< 5\%$ prevalence)?
4. Do classifiers trained on synthetic data achieve comparable accuracy to those trained on real data?
5. What are the wall-clock time and resource requirements for each method?

These questions remain open because prior work evaluates methods in isolation, on different datasets, under incomparable conditions. Our contribution is a controlled comparison on a concrete analytical workload.

A.3.3 Approach

For training-based synthesis, we will implement a differentially private generative model using PyTorch and the Opacus library, with privacy guarantees via DP-SGD (per-sample gradient clipping and calibrated noise injection). For training-free synthesis, we will implement Private Evolution using black-box API access to foundation models, achieving privacy through differentially private nearest-neighbor histograms. Both methods will be evaluated under matched privacy budgets on our 22-query SQL benchmark, comparing query fidelity, statistical preservation, downstream utility, and computational cost.

A.3.4 Expected outputs

1. *Technical report:* Implementation details, privacy analysis, query-by-query benchmark results, statistical fidelity analysis, and practical recommendations.
2. *SQL benchmark suite:* The 22-query benchmark with natural language specifications, SQL code, tolerance thresholds, evaluation scripts, and baseline results.
3. *Differentially private synthetic datasets:* Two synthetic versions of the DCA corpus under matched (ε, δ) (one from DP-SGD, one from PE) with documented privacy guarantees.
4. *Project website:* Public-facing summary with visualizations, query-level comparisons, and deployment guidelines.
5. *Open-source implementations:* Reproducible code for preprocessing, training, generation, and evaluation.

B Contributions

Mehak Kapur ran and validated DuckDB queries to ingest the Parquet datasets. They helped construct the unified wide training table. They implemented and executed the DP-SGD VAE pipeline to generate synthetic data. They produced evaluation outputs and visualizations comparing real versus synthetic distributions. They contributed to report writing.

Hana Tjendrawasi ran and validated DuckDB queries to ingest the Parquet datasets. They helped construct the unified wide training table by aligning schemas across sources. They led project logistics. They supported the DP-SGD workflow through preprocessing and experiment setup. They assisted with reviewing evaluation metrics and results. They contributed to report writing.

Jason Tran built the 19 reporting tables from raw telemetry data via DuckDB aggregation scripts and parsed the 24 queries. They designed and implemented the wide-table vs. per-table synthesis strategy, the PE pipeline, and the MST baseline workflow for tabular comparison. They supported the DP-SGD workflow through implementation. They implemented benchmark execution, decomposition, and evaluation across real and synthetic outputs, integrated methods and findings into the report. They contributed to report writing.

Phuc Tran performed exploratory data analysis on raw telemetry tables. They conducted schema verification and column mapping between raw and reporting tables.