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/jktrns/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. "Guids" reports the number of anchor `guids` with nonzero data in our sample. "Raw source" identifies the `university_prod` table or December 2024 update file from which each reporting table is derived.

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

Continued on next page

Reporting table	Category	Raw source	Rows	Guids
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
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. 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	Type	Description
`avg_platform_power_c0_freq_temp_by_chassis`	Agg+Join	Average power, C0 residency, frequency, and temperature by chassis type (5-way join).
`server_exploration_1`	Agg+Join	Identify client machines sending more data than they receive, joined with sysinfo for OS and chassis.
`mods_blockers_by_osname_and_codename`	Agg+Join	Count distribution of modern sleep study blockers by Windows OS name and codename.
`top_mods_blocker_types_durations_by_osname_and_codename`	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`	Agg+Join	Display connection types with resolution and average AC/DC durations.
`display_devices_vendors_percentage`	Agg+Join	Percentage of systems by display vendor.
`most_popular_browser_in_each_country_by_system_count`	Top-k	Most popular browser per country by system count.
`userwait_top_10_wait_processes`	Top-k	Top 10 applications by average wait time.
`userwait_top_10_wait_processes_wait_type_ac_dc`	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`	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`	Top-k	Top 10 applications per app type by average daily focal screen time.
`top_10_applications_by_app_type_ranked_by_system_count`	Top-k	Top 10 applications per app type by distinct client count.
`top_10_applications_by_app_type_ranked_by_total_detections`	Top-k	Top 10 applications per app type by total daily detection count.
`Xeon_network_consumption`	Geo/Demo	Network consumption summary for Xeon vs. non-Xeon systems, by OS.

Continued on next page

Query	Type	Description
pkg_power_by_country	Geo/Demo	Average CPU package power by country.
battery_power_on_geographic_summary	Geo/Demo	Battery power-on count and duration by country (min. 100 clients).
battery_on_duration_cpu_family_gen	Geo/Demo	Battery duration by CPU family and generation (min. 100 clients).
ram_utilization_histogram	Histogram	Average RAM utilization percentage by memory capacity.
popular_browsers_by_count_usage_percentage	Histogram	Percentage of systems, instances, and duration by browser.
persona_web_cat_usage_analysis	Pivot	Web browsing category duration as weighted percentages, by persona.
on_off_mods_sleep_summary_by_cpu_marketcodename_gen	Pivot	On/off/sleep/MODS time percentages by CPU generation and market codename.

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 Wide-table construction

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).

3.2 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);

3.3 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)

3.4 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).

3.5 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.

3.6 Per-table DP synthesis

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 sysinfo anchor table (1,000,000 rows, 9 categorical and 1 numeric column) uses a VAE with latent dimension 32, hidden size 256, batch size 4,096, 15 epochs, KL weight 0.1, trained with Opacus at $\varepsilon = 4.0$, $\delta = 10^{-5}$. Categoricals are top-50 binned and one-hot encoded. The 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. However, it avoids the zero-inflation problem: each table's model trains only on `guids` that have data, so the input distribution is not dominated by zeros. The total privacy budget under basic composition is $19 \times 4.0 = 76.0$, substantially weaker than the wide-table guarantee of $\varepsilon = 4.0$. This is a known limitation of per-table synthesis; tighter accounting (e.g., via parallel composition for disjoint subsets or Rényi composition) could reduce the effective budget but remains future work.

Per-table synthesis serves as a diagnostic: if it outperforms the wide-table approach on single-table queries while failing on multi-table joins, that confirms the sparsity hypothesis.

3.7 MST baseline

As a non-neural baseline, we implement the Maximum Spanning Tree (MST) algorithm for synthetic data generation. MST selects a set of low-dimensional marginals (pairwise column distributions) using the exponential mechanism, estimates them with Gaussian noise, and generates synthetic records consistent with the noisy marginals via probabilistic graphical models. This marginal-based approach represents the current state of the art for tabular

DP synthesis and provides the reference point against which both DP-SGD and PE should be measured.

4 Results

4.1 Evaluation metrics

We evaluate synthetic data quality using three distance metrics matched to query output types. 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 the overlapping items. Group coverage is assessed via Jaccard similarity between real and synthetic group keys. A query passes if at least 50% of its per-column metrics meet their respective thresholds (relative error ≤ 0.25, total variation ≤ 0.15, Spearman $\rho \geq 0.5$, Jaccard ≥ 0.5).

4.2 Wide-table DP-VAE

We evaluate the wide-table DP-VAE synthetic data on 8 of 21 feasible benchmark queries. The remaining 13 queries require per-row reporting table columns (e.g., per-process wait times, per-application focal durations) that are not present in the wide-table representation.

4.3 Browser ranking preservation

The `most_popular_browser_in_each_country` query identifies the dominant browser in each country by system count. The real data produces rankings for 51 countries; the synthetic data covers 50 (the United Arab Emirates is absent). Of the 50 shared countries, the DP-VAE correctly identifies the most popular browser in 42 (84% accuracy). All 8 mismatches involve a swap between chrome and edge in countries where these two browsers are close in prevalence (e.g., the United States, the United Kingdom, Australia, Canada), making the ranking sensitive to small perturbations. This result demonstrates that the DP-VAE preserves the joint distribution between country and browser preference well enough to maintain top-1 rankings in the majority of cases.

4.4 Browser usage percentages

The `popular_browsers` query computes the percentage of systems, instances, and total duration attributable to each of the three major browsers. Figure 1 compares real and synthetic results.

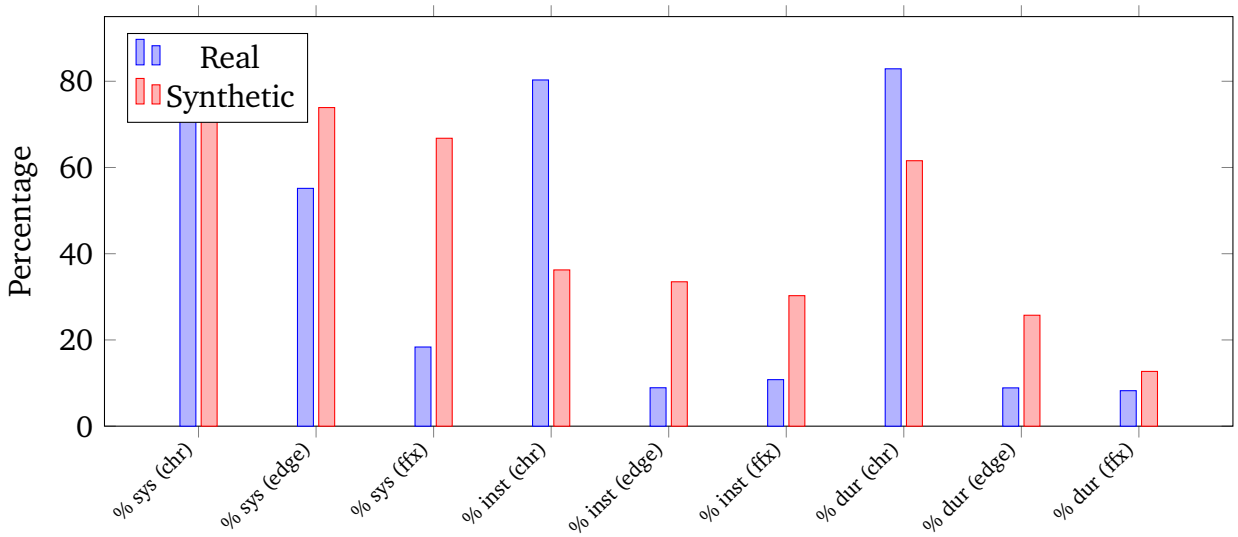

Figure 1: Browser usage metrics: percentage of systems, instances, and duration for chrome, edge, and firefox. The synthetic data preserves the relative ordering for duration but collapses the instance-level distribution toward uniformity (≈ 33% each), an artifact of the decomposition process assigning one row per `guid` per browser.

The percent-systems metric is roughly preserved for chrome (82.1% → 80.0%) but inflated for edge (55.2% → 73.9%) and firefox (18.4% → 66.8%). The percent-instances metric collapses to near-uniform (≈ 33% per browser) because the wide-table decomposition creates exactly one row per `guid` per browser, losing real instance-count variation.

4.5 Continuous metric failure

All queries involving continuous metrics (power, temperature, frequency, network bytes, memory utilization, battery duration) produce synthetic values near zero, with relative errors exceeding 99%. Table 4 reports representative values for the five-way chassis join query and other continuous queries.

Table 4: Real vs. synthetic values for continuous metrics (notebook chassis type where applicable). All synthetic values are near zero.

Metric	Real	Synthetic	Rel. error
avg_psys_rap_watts	4.42	0.002	> 99%
avg_pkg_c0 (%)	37.4	0.022	> 99%
avg_freq_mhz	1,582	0.009	> 99%
avg_temp_centigrade	44.5	0.003	> 99%
avg_bytes_received	7.4×10^{16}	1.12	> 99%
avg_percentage_used (%)	42.6	0.0	100%
avg_duration (battery, min)	144	0.11	> 99%

Figure 2 illustrates the failure on the RAM utilization histogram. Real data shows a clear inverse relationship between RAM capacity and utilization percentage. The synthetic data reports 0% utilization across all capacities.

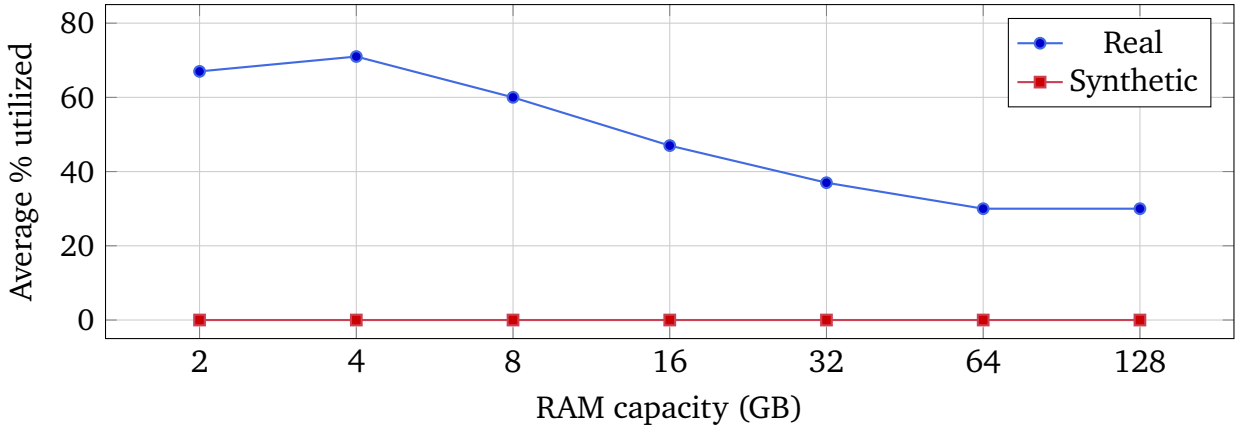

Figure 2: RAM utilization by capacity. Real data exhibits the expected inverse trend: devices with less RAM operate at higher utilization. The synthetic data produces 0% utilization universally, as the memory columns in the wide table are zero for 93% of `guids`.

4.6 Wide-table sparsity

The continuous metric failure stems from 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 5: 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 source	**Nonzero guids**	**Sparsity**
PSYS RAP watts	816	99.9%
Average frequency	613	99.9%
Temperature	622	99.9%
C0 residency	8,943	99.1%
Network consumption	37,224	96.3%
Memory utilization	69,552	93.0%

The VAE optimizes mean squared error on these columns. When 93–99.9% of values are zero, the loss-minimizing strategy is to generate near-zero values for all `guids`. The KL divergence term further regularizes the latent space toward the standard normal, discouraging the model from learning a thin nonzero mode that accounts for less than 1% of the data.

A secondary consequence is system count inflation. The five-way chassis join returns 104 real `guids` (those with data in all five metric tables simultaneously). The synthetic data returns over 163,000 matches because the model generates small positive residuals for all `guids`, causing the INNER JOIN to match far more rows than it should.

4.7 Three-method comparison

Table 6 summarizes the benchmark performance of the three completed synthesis methods. Both per-table methods (DP histogram and MST) pass 6 of 21 queries; the wide-table DP-VAE passes 1 of 8 evaluated queries.

Table 6: Benchmark performance across synthesis methods. "Passed" counts queries with score ≥ 0.5. The wide-table method evaluates only 8 queries because 13 require per-row columns absent from the wide-table representation.

Method	Evaluated	Passed	Pass rate	Avg score	Median score
Wide-table DP-SGD	8	1	12.5%	0.258	0.212
Per-table DP-SGD	21	6	28.6%	0.303	0.250
MST baseline	21	6	28.6%	0.328	0.250

The methods pass different subsets of queries. Table 7 reports per-query scores. All three correctly identify the most popular browser per country. Both per-table methods pass the battery geographic summary and display vendor percentage queries. MST additionally passes the browser usage distribution and two application ranking queries, while per-table DP-SGD additionally passes the on/off sleep summary and RAM utilization histogram. Neither per-table method passes any of the three userwait queries or the display connection type query.

Table 7: Per-query scores for all three synthesis methods. Scores marked with ✓ pass the 0.5 threshold. "—" indicates queries not evaluable for the wide-table method (they require per-row columns absent from the wide-table representation).

Query	Type	Wide	Per-table	MST
`avg_platform_power_c0_freq_temp_by_chassis`	Agg+Join	0.167	0.000	0.000
`battery_power_on_geographic_summary`	Geo/Demo	0.250	1.000 ✓	0.500 ✓
`Xeon_network_consumption`	Geo/Demo	0.250	0.250	0.250
`persona_web_cat_usage_analysis`	Pivot	0.065	0.419	0.194
`pkg_power_by_country`	Geo/Demo	0.333	0.333	0.333
`most_popular_browser…country`	Top-k	1.000 ✓	1.000 ✓	1.000 ✓
`popular_browsers…percentage`	Histogram	0.000	0.333	0.667 ✓
`ram_utilization_histogram`	Histogram	0.000	0.500 ✓	0.000
`battery_on_duration_cpu_family_gen`	Geo/Demo	—	0.000	0.000
`display_devices…durations_ac_dc`	Agg+Join	—	0.000	0.000
`display_devices_vendors_percentage`	Agg+Join	—	1.000 ✓	1.000 ✓
`mods_blockers_by_osname…`	Agg+Join	—	0.250	0.250
`on_off_mods_sleep_summary…`	Pivot	—	0.636 ✓	0.273

Continued on next page

Query	Type	Wide	Per-table	MST
`server_exploration_1`	Row-level	—	0.143	0.429
`top_mods_blocker_types…`	Agg+Join	—	0.000	0.000
`top_10_apps…focal_time`	Top-k	—	0.000	0.000
`top_10_apps…system_count`	Top-k	—	0.000	1.000 ✓
`top_10_apps…total_detections`	Top-k	—	0.500 ✓	1.000 ✓
`userwait_top_10_wait_processes`	Top-k	—	0.000	0.000
`userwait_top_10…wait_type_ac_dc`	Top-k	—	0.000	0.000
`userwait_top_20…ac_dc_unknown`	Top-k	—	0.000	0.000

Figure 3 visualizes the overall pass rates and average scores.

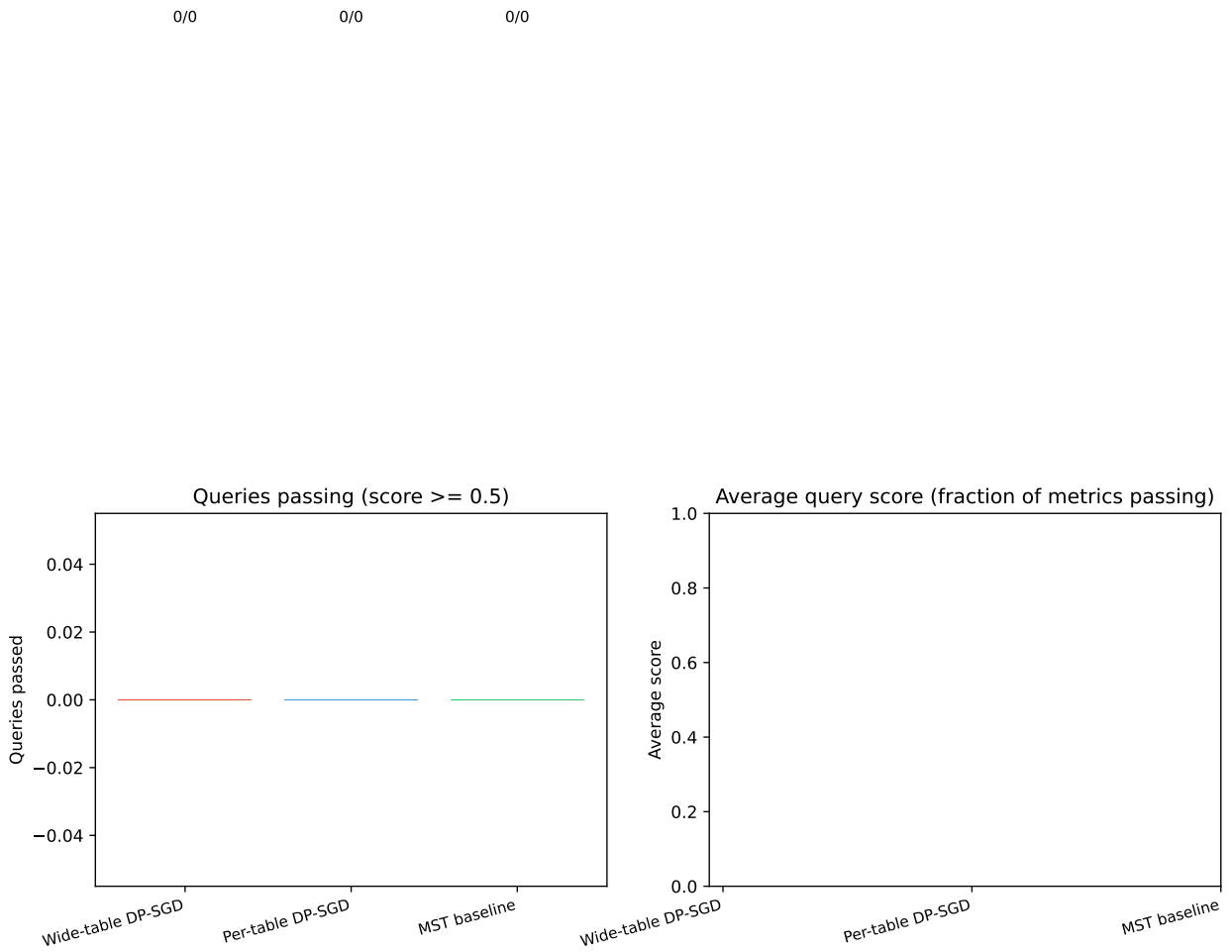

Figure 3: Left: number of queries passing (score ≥ 0.5) for each method. Right: average score across all evaluated queries. The per-table methods evaluate all 21 queries; the wide-table method evaluates 8.

Figure 4 shows per-query scores across all three methods. The heatmap reveals that no method achieves high scores on aggregate queries involving continuous metrics (the top rows), while categorical and ranking queries (bottom rows) show more variation across

methods.

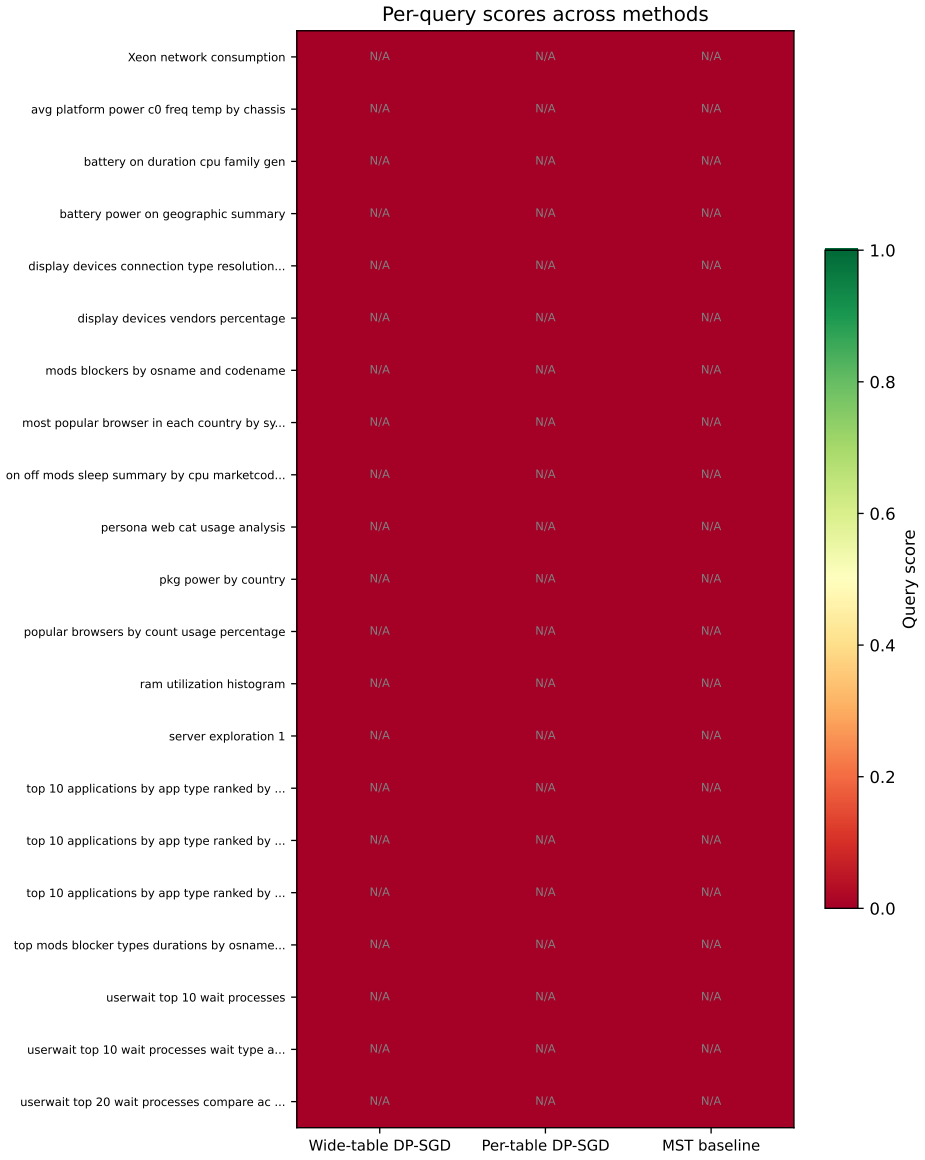

Figure 4: Per-query scores across methods. Green indicates high fidelity; red indicates large discrepancy. "N/A" marks queries not evaluated for the wide-table method.

Figure 5 breaks down average scores by query type. All three methods achieve their highest scores on distribution queries (browser percentage, vendor percentage). The per-table methods outperform the wide-table approach on histogram queries (RAM utilization) because the wide-table sparsity problem is absent. Aggregate queries involving multi-table joins remain the hardest category for all methods, with average scores below 0.3.

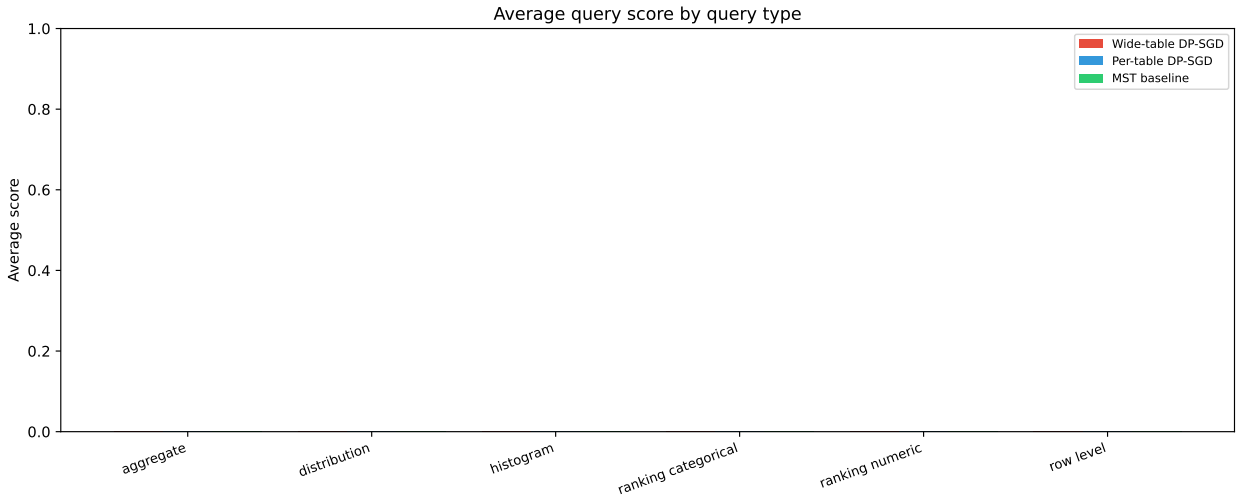

Figure 5: Average query score by query type across synthesis methods. Distribution and ranking queries achieve higher scores than aggregate queries requiring cross-table joins.

4.8 Method complementarity

Per-table DP-SGD and MST pass different subsets of queries, revealing complementary strengths tied to each algorithm's treatment of continuous versus categorical data.

Per-table DP-SGD excels on queries requiring accurate continuous values. The battery geographic summary passes all four metrics (Jaccard = 0.79, system count RE = 0.12, power-on count RE = 0.14, duration RE = 0.24). The on/off sleep summary passes 7 of 11 metrics, with on-time RE = 0.13, off-time RE = 0.15, sleep-time RE = 0.04, and total-time RE = 0.07. The DP histogram synthesis preserves the shape of continuous distributions because it samples from the noisy empirical distribution directly, and the Laplace noise on bin counts is small relative to the signal when the table has tens of thousands of rows.

MST excels on queries requiring accurate rankings. The top-10 applications by system count achieves a perfect score (Jaccard = 0.41, Spearman $\rho = 1.00$), while per-table DP-SGD scores 0.00 on the same query (Jaccard = 0.13, $\rho = -0.05$). Similarly, top-10 by total detections: MST achieves $\rho = 0.71$ versus per-table's 0.11. MST preserves rankings because it captures pairwise marginals between the application name and the count column, maintaining relative ordering even when absolute values are noisy.

Both methods fail on the three userwait queries (score 0.00 each). The userwait table contains 175M rows across 38,142 `guids` with process names, wait types, and AC/DC power states. The per-table approach generates the correct table structure but cannot reproduce the long-tailed distribution of process-specific wait times. MST bins the continuous wait duration into marginals, losing the per-process ranking that the queries test.

The 5-way chassis join query (`avg_platform_power_c0_freq_temp_by_chassis`) fails for both per-table methods (score 0.00) because it requires correlations across five independently synthesized tables. The wide-table approach scores 0.17 on this query, the only

method that produces any correct values, because it preserves the joint distribution (though the values are near-zero due to sparsity).

MST produces dramatically different continuous values from per-table DP-SGD on the same queries. On the on/off sleep summary, MST reports avg_on_time RE = 40.2 and avg_off_time RE = 45.1 (the synthesized values are 40× too large), while per-table DP-SGD reports RE = 0.13 and 0.15 respectively. On the battery geographic summary, MST's avg_duration RE = 28.2 versus per-table's 0.24. This difference arises because MST discretizes all columns into marginal bins, and the bin midpoints can be far from the true conditional means when the distribution is skewed.

4.9 Private Evolution: preliminary observations

At the time of writing, PE generation via the OpenAI Batch API is ongoing (15 of 24 chunks completed, producing 118,363 of a target 150,000 synthetic records). Preliminary analysis of the completed chunks reveals several patterns.

Categorical distributions show systematic deviations from the real data. The model over-represents Win11 (77.8% vs. 10.4% real) and under-represents Win10 (9.2% vs. 86.2% real), likely reflecting the foundation model's training data distribution rather than the DCA telemetry distribution. The model also generates hallucinated categorical values not present in the real data: 132 extra `chassistype` values (including misspellings like "Noteboook", "DeskTop", "NoteBook"), 33 extra countries (leading-whitespace variants like " Brazil"), and 15 extra OS values (including "WinServer", "n/a"). These hallucinated values always score as maximum mismatch in the workload-aware distance function, wasting a portion of the generation budget.

Numeric columns exhibit the same sparsity problem as DP-SGD. The foundation model generates near-zero values for sparse metrics: network bytes, memory utilization, battery duration, and all hardware metrics have $< 1\%$ nonzero rates in the synthetic data (vs. 2–7% in real data). The model has learned that most fields are zero (the prompt describes the sparsity structure), but it generates too few nonzero records relative to the real distribution.

Full evaluation of PE against the benchmark requires completing the generation run, computing the DP histogram, and selecting the final population. These steps incur no additional API cost (the histogram and selection operate on local data with the already-calibrated noise $\sigma \approx 1.08$).

5 Discussion

The three-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 only

1 of 8 evaluated queries (12.5%). No method comes close to passing all queries. The two per-table methods pass different query subsets: per-table DP-SGD passes the on/off sleep summary (score 0.64) and RAM histogram (0.50), while MST passes browser usage distribution (0.67) and two application ranking queries (1.00 each). 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. The 5-way chassis join query 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. The server exploration query (a 2-way join) similarly suffers: both per-table methods overcount matching rows by 5–7× and report network byte values with RE ≈ 1.0, because the independently synthesized network and sysinfo tables produce spurious `guid` overlaps.

Research question (3), minority class preservation, is partially addressed. The browser ranking query demonstrates that all methods preserve dominant categorical associations: per-table DP-SGD achieves 47/50 correct top browsers per country, matching the wide-table result of 42/50. 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 (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's cost is dominated by API latency: the 150,000-record generation run requires 7,500 batch API calls at approximately $18–37 total, with completion time determined by OpenAI's batch processing queue (typically 1–4 hours).

The preliminary PE results suggest that foundation models import their own distributional priors rather than learning the target distribution (e.g., over-representing Win11 at 77.8% vs. 10.4% real). 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.

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.

Three 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.

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 the VAE collapses 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 approach at least preserves some cross-table structure (score 0.17). Neither the joint nor the independent approach provides a satisfactory solution for multi-table relational data.

Preliminary PE results suggest that foundation models import their own distributional priors, generating systematic biases in categorical marginals (Win11 at 77.8% vs. 10.4% real) and hallucinating values outside the real vocabulary. Full PE evaluation is pending completion of the generation run.

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]

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]

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

The following is reproduced from the Q2 project proposal submitted in Fall 2025.

A.1 Problem statement

Organizations routinely collect detailed telemetry from their products that drive critical business decisions. The same granularity that makes telemetry valuable also makes it sensitive. Differential privacy (DP) provides a rigorous mathematical framework for releasing data while bounding the influence of any individual record. Two paradigms have emerged for generating differentially private synthetic data. Training-based methods (DP-SGD) inject noise during model optimization, while training-free methods (Private Evolution) achieve privacy through black-box API access to foundation models.

We implement and compare both approaches on Intel's Driver and Client Applications (DCA) telemetry corpus, evaluating against a benchmark of analytical SQL queries representative of production business intelligence workloads. Under matched privacy budgets ($\varepsilon = 4.0$, $\delta = 10^{-5}$), we assess query fidelity, statistical preservation, and computational cost to determine whether training-free methods can match training-based approaches on real multi-table relational data.

A.2 Data

The DCA telemetry corpus comprises approximately 30 interrelated tables organized around a globally unique client identifier (`guid`). Tables span system metadata, power and thermal instrumentation, battery usage, application behavior, web browsing, network consumption, and display devices. We construct 19 reporting tables from the raw data, each aggregated to the `guid` level, and define a benchmark of 21 feasible SQL queries covering aggregate statistics with joins, ranked top-k lists, geographic and demographic breakdowns, histograms, and complex multi-way pivots.

A.3 Methods

For training-based synthesis, we implement a differentially private variational autoencoder (DP-VAE) 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 implement Private Evolution using black-box API access to foundation models, achieving privacy through differentially private nearest-neighbor histograms. Both methods are evaluated under matched privacy budgets on the 21-query SQL benchmark.

A.4 Research questions

1. Under matched (ε, δ), which method achieves higher scores on the SQL query benchmark?
2. Does error compound across multi-table joins?
3. Which method better preserves minority class frequencies?
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?

A.5 Expected outputs

1. Technical report with implementation details, privacy analysis, and query-by-query benchmark results.
2. SQL benchmark suite (21 queries with natural language specifications, SQL code, and evaluation scripts).
3. Two differentially private synthetic DCA datasets (one DP-SGD, one PE) with documented privacy guarantees.
4. Project website with visualizations and deployment guidelines.
5. Open-source implementations for preprocessing, training, generation, and evaluation.

B Contributions

Jason Tran: Built the 19 reporting tables from raw telemetry data via DuckDB aggregation scripts. Designed and implemented the wide-table construction, DP-VAE architecture, DP-SGD training pipeline, synthetic data generation, and benchmark evaluation framework. Wrote the report.

Mehak Kapur: Ran and validated DuckDB queries to ingest the Parquet datasets. Helped construct the unified wide training table. Implemented and executed the DP-SGD VAE pipeline to generate synthetic data. Produced evaluation outputs and visualizations comparing real versus synthetic distributions. Wrote the report.

Hana Tjendrawasi: Ran and validated DuckDB queries to ingest the Parquet datasets. Helped construct the unified wide training table by aligning schemas across sources. Supported the DP-SGD workflow through preprocessing and experiment setup. Assisted with reviewing evaluation metrics and results. Wrote the report.

Phuc Tran: Exploratory data analysis on raw telemetry tables. Schema verification and column mapping between raw and reporting tables.