

# 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 . . . . .	3
1.1	Motivation . . . . .	3
1.2	Prior work . . . . .	3
2	Data and problem statement . . . . .	5
2.1	Intel DCA telemetry data . . . . .	5
2.2	Analytical query workload . . . . .	7
2.3	Query inventory . . . . .	7
2.4	Formal benchmark definition . . . . .	9
2.5	Research questions . . . . .	9
3	Methods . . . . .	11
3.1	Wide-table construction . . . . .	11
3.2	DP-VAE architecture . . . . .	11
3.3	DP-SGD training and privacy accounting . . . . .	12
3.4	Synthetic data generation and decomposition . . . . .	13
3.5	Private Evolution (planned) . . . . .	13
4	Results . . . . .	14
4.1	Browser ranking preservation . . . . .	14
4.2	Browser usage percentages . . . . .	14
4.3	Continuous metric failure . . . . .	15
4.4	Wide-table sparsity . . . . .	16
5	Discussion . . . . .	17
6	Conclusion . . . . .	17
	References . . . . .	19
A	Project proposal . . . . .	20
A.1	Problem statement . . . . .	20
A.2	Data . . . . .	20
A.3	Methods . . . . .	20
A.4	Research questions . . . . .	21
A.5	Expected outputs . . . . .	21
B	Contributions . . . . .	22

# 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  $(\epsilon, \delta)$ -differentially private if for all neighboring databases  $D, D'$  differing by one record and all measurable output sets  $S$ :

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

The parameter  $\epsilon$  controls the privacy-utility tradeoff. Smaller values confer stronger privacy at the cost of noisier outputs. The slack  $\delta$  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  $\ell_2$  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 ( $\epsilon \approx 32$  for CIFAR-10). This motivates the search for methods that achieve comparable fidelity at lower  $\epsilon$ .

[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  $\text{FID} \leq 7.9$  at  $\epsilon = 0.67$  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 $\times$  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
<code>system_sysinfo_unique_normalized</code>	Metadata	Direct copy	1,000,000	1,000,000
<code>system_cpu_metadata</code>	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_centrigrade	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_frngnd_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_frngnd_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.<sup>1</sup>

---

<sup>1</sup>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  $\mathcal{Q} = \{q_1, \dots, q_{21}\}$  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:

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

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

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

where  $\tau_j$  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  $(\epsilon, \delta)$ , 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  $1,000,000 \times 307$  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  $\mu$  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:

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

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  $\varepsilon^*$ ,  $\delta$ , clip norm  $C$ , epochs  $E$ , batch size  $B$

**Output:** Trained parameters  $\theta$ , final  $\varepsilon$

Wrap optimizer with Opacus:  $\sigma \leftarrow \text{make\_private\_with\_epsilon}(\varepsilon^*, \delta, E, B)$ ;

**for**  $epoch = 1, \dots, E$  **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:  $\tilde{g} \leftarrow \frac{1}{B} (\sum_i \bar{g}_i + \mathcal{N}(0, \sigma^2 C^2 I))$ ;

        Update:  $\theta \leftarrow \theta - \eta \tilde{g}$ ;

$\varepsilon \leftarrow \text{PrivacyAccountant.get\_epsilon}(\delta)$ ;

---

### 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  $\sigma$  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	$307 \rightarrow 512 \rightarrow 512 \rightarrow (64\mu, 64 \log \sigma^2)$
Decoder architecture	$64 \rightarrow 512 \rightarrow 512 \rightarrow 9$ 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 $\sigma$	auto-calibrated by Opacus
Target $\varepsilon$	4.0
$\delta$	$10^{-5}$
Final $\varepsilon$	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 (planned)

We plan to 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 iteratively evolves a population of synthetic records through three steps per iteration:

1. *Candidate generation*: prompt a foundation model to produce variations of current synthetic records that conform to the DCA schema.
2. *Private selection*: compute a nearest-neighbor histogram in which each real record votes for its closest synthetic candidate under the workload-aware distance. Add Gaussian noise calibrated to the histogram sensitivity (1, since each `guid` contributes exactly one vote).
3. *Resampling*: resample the synthetic population according to the noisy histogram, retaining candidates that best approximate the real data distribution.

Privacy follows from composition of  $T$  Gaussian mechanisms, equivalent to a single mechanism with noise multiplier  $\sigma/\sqrt{T}$ . We will match the DP-SGD budget ( $\epsilon = 4.0$ ,  $\delta = 10^{-5}$ ) for a controlled comparison.

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.

## 4 Results

We evaluate the 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.

### 4.1 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.

### 4.2 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.

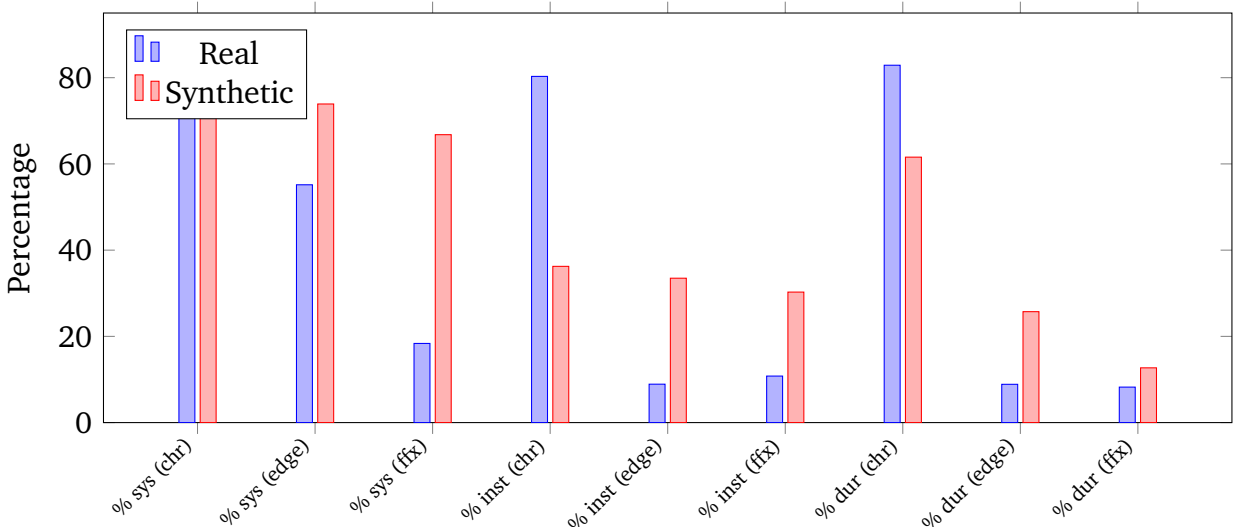


Figure 1: Browser usage metrics: percentage of systems, instances, and duration for chrome, edge, and firefox. The synthetic data preserves the relative ordering for duration but collapses the instance-level distribution toward uniformity ( $\approx 33\%$  each), an artifact of the decomposition process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

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

### 4.3 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_centrigrade	44.5	0.003	> 99%
avg_bytes_received	$7.4 \times 10^{16}$	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.

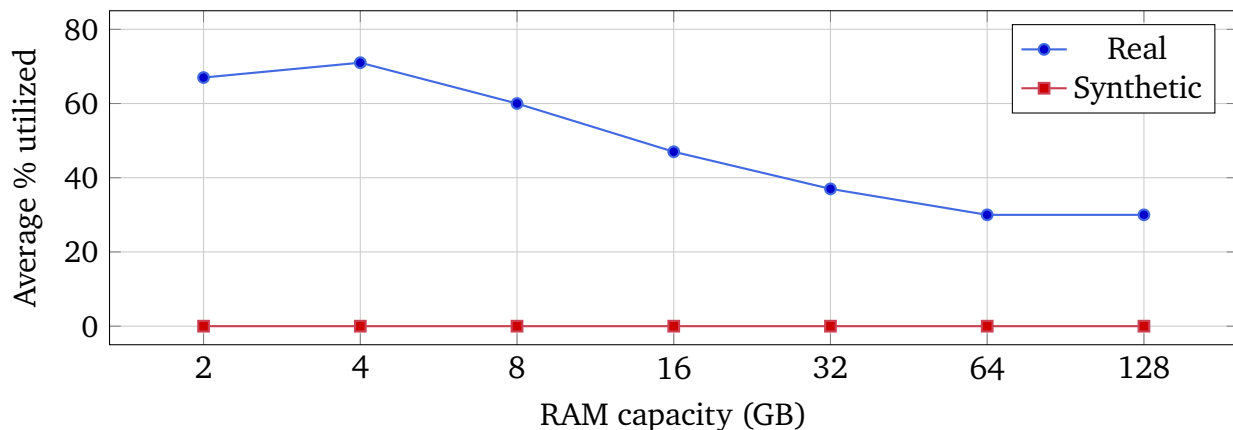


Figure 2: RAM utilization by capacity. Real data exhibits the expected inverse trend: devices with less RAM operate at higher utilization. The synthetic data produces 0% utilization universally, as the memory columns in the wide table are zero for 93% of guids.

## 4.4 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 <code>guids</code>	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.



## 5 Discussion

The DP-SGD results provide a direct answer to research question (2). Error compounds dramatically across multi-table joins when the synthesis approach introduces structural sparsity artifacts. The wide-table design is correct in principle, as it preserves the joint distribution across tables by construction. In practice, it creates a feature matrix where most columns are nearly degenerate. The VAE collapses these sparse columns to their mode (zero), destroying any signal in the nonzero tail.

The browser ranking result demonstrates that the approach succeeds for well-populated categorical attributes. The joint distribution between country and browser preference is dense (every guid has a country; most have web browsing data), so the model learns a meaningful conditional distribution that preserves top-1 rankings in 84% of countries.

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.
- Per-table independent synthesis (sacrificing cross-table correlations) as a baseline to isolate the sparsity effect.
- A zero-inflated loss function replacing plain MSE for numeric columns.
- Restricting the wide table to guids with sufficient coverage (e.g., present in  $\geq 3$  metric tables).

Private Evolution may be better suited to this setting. Rather than learning a generative model over the full 307-dimensional feature space, PE operates in an embedding space and uses a foundation model as a proposal mechanism. If the foundation model can generate plausible telemetry records with realistic sparsity patterns (i.e., most records have data in only a few metric tables), PE could preserve the heterogeneous coverage structure that the VAE collapses.

## 6 Conclusion

We presented a controlled evaluation of differentially private synthetic data generation on Intel’s DCA telemetry corpus. The DP-VAE trained with DP-SGD ( $\epsilon = 3.996$ ,  $\delta = 10^{-5}$ ) preserves categorical joint distributions (42/50 countries receive the correct top browser) but fails on all continuous metrics due to extreme zero-inflation in the wide-table representation.

This result identifies the sparsity structure of multi-table joins as a critical factor in the success or failure of DP synthetic data generation. When coverage varies across tables (as it does in any real multi-table dataset), a unified representation concentrates nearly all probability mass at zero for sparse columns, and standard generative models learn to reproduce this mode.

Our evaluation so far covers only the DP-SGD paradigm. The comparison with Private Evo-

lution, using matched privacy budgets and the same 21-query benchmark, will determine whether training-free methods handle the heterogeneous structure of relational telemetry data more effectively.

## 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 ( $\epsilon = 4.0$ ,  $\delta = 10^{-5}$ ), 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  $(\epsilon, \delta)$ , 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.