An Evaluation of Open-source Tools for the Provision of Differential Privacy

Shiliang Zhang∗ Anton Hagermalm† Sanjin Slavnic †
February 22, 2022

Abstract

The concept of differential privacy has widely penetrated academia and industry, with its formal guarantee on individual privacy that leads to compliances with privacy legislation, e.g., GDPR. However, there is a lack of understanding on tools capable of achieving differential privacy, and it is not clear what to expect from existing differential privacy tools when implementing privacy protection. Such an obstacle limits private applications’ further prosperity. This paper reviews and evaluates the state-of-the-art open-source differential privacy tools of different domains using various estimating categories and privacy settings. Particularly, we look into the performances of three differential privacy tools for machine learning, two for statistical query, and four for synthetic data generation. We test all the tools on both continuous and categorical data and quantify their performance under different privacy budget and data size w.r.t. utility loss and system overhead. The accumulated evaluation results reveal several patterns that users can follow to optimally configure the tools, and provide preliminary guidelines on tool selection under different criteria. Finally, we openly release our evaluation coding repository, a framework that users can reuse to further evaluate the studied tools and beyond. We anticipate this work to provide a comprehensive insight into the performances of the existing dominant privacy tools, and a concrete reference for a potentially large developer community on private applications, thus narrowing the gap between conceptual differential privacy and private functionality development.

Keywords: differential privacy, open-source tools, evaluation

1 Introduction

Differential privacy [1] (DP) has been proposed as a formal privacy guarantee for data. It reinforces a more substantial privacy basis for privacy regulations such as GDPR [2] or HIPAA [3] than existing approaches like anonymization, and is regarded as the most viable privacy protection technique for real-world data analysis [4]. Nevertheless, a gap exists between fundamental and applied research on the topic of DP [5]. The academic literature around DP is less intuitive, with concepts that do not necessarily map well to the vernacular used in industry. Therefore, there lacks a bridge between the academic DP landscape and the practical application of DP in consumer technology. Particularly, there is an absence of knowledge over what software tools can be leveraged in privacy

∗Computer Science and Engineering, Chalmers University of Technology. Email: shiliang@chalmers.se
†Spotify. Email: {anton.hagermalm, sanjinslavnic}@gmail.com
application development with DP principles, and to what level of performance can researchers and developers expect from the existing tools \[6\]. This disadvantage hinders further prosperity of DP in real-world applications and consequently induces practical obstacles to protect individual privacy from a software & application perspective.

Recently, a number of open-source tools were proposed for applying differential privacy (DP) in statistical queries \[7\], machine learning \[8\], and synthetic data generation \[9\]. Those tools connect DP theories with DP service developments by avoiding application development from scratch, facilitating developer communities to achieve expected functionalities under varieties of DP configurations. However, there is no comparative evaluation of these tools that provide generic guideline information, e.g., how they differ and what privacy settings should be used to achieve DP in practice.

To address the concerns as mentioned above, we conduct this work to evaluate open-source tools for differential privacy (DP). We defined the metrics of data utility and system overhead for evaluating different tools, and evaluated three domains of DP tools: statistical query, machine learning, and synthetic data generation, categorized by the difference in DP configurations. Through this evaluation, we aim to provide deeper insight into existing open-source DP tools, specifically their performance in DP functionalities, and find suitable configurations for the tools to provide recommendations for how these tools can be used to their full extent.

1.1 Related work

This section reviews existing works regarding performance evaluation of privacy tools that apply DP. While there have been numerous studies around differential privacy, none, to the best of our knowledge, provide a comparative study between the various open-source tools that can be applied in practice, nor did they offer sufficient insight into how to apply and configure DP tools in privacy-preservation. Therefore, we aim in this review to introduce previous and relevant research, and compare techniques and measures for how to evaluate privacy tools regarding their performance. Since DP’s influence differs due to differences in services, the review is categorized into three functionality domains: statistical queries, machine learning, and synthetic data generation, as described in the following.

1.1.1 Statistical Queries

Statistical queries in this work refer to the analysis of data to extract statistical features, e.g., the operation of \textit{COUNT}, \textit{SUM}, \textit{AVERAGE}, etc. One differentially private (DP) query engine that has been integrated into the industry is Flex \[10\]. The open-source library for Flex is deprecated\(^1\); however, their evaluation remains relevant to provide clues on evaluating privacy tools.

Johnson \textit{et al.} \[10\] evaluated Flex using an SQL-compatible interface, which makes it convenient to put the interface in front of any already deployed SQL-compatible database and, in turn, lowers the bar of adoption. Their evaluation uses a large set of real-world queries run by Uber’s data engineers in production, which gives insights into how Flex would perform in the industry. However, this evaluation merely benchmarks one \(\epsilon\) value (\(\epsilon = 0.1\)) and presents only one single value of the additional overhead of 4.86ms corresponding to 0.03 \% of the average execution time of their non-privacy protected queries. Furthermore, the 4.86ms overhead does not include the pre-collection of frequent join attributes, which has to be updated each time the underlying data is updated.

\(^1\)https://github.com/uber-archive/sql-differential-privacy
Such evaluating procedure induces the risk that the actual overhead might be more significant than presented in practice. Beyond that, there is a lack of illustration of whether overhead changes when settings differ, e.g., in dataset sizes, privacy configurations or queries, etc.

The industry also has observed the DP tools of Smartnoise and Google Differential Privacy integrated into private statistical-query services. Smartnoise results from years of cumulative experience building and deploying privacy tools for research and has recently become a tool in Microsoft’s privacy ecosystem. However, even though the open-source software community provides transparency and demonstrates examples, no comparative research has been done on the tool’s query engine.

Google DP has been evaluated alongside Flex and PinQ\(^2\). Their evaluation performed 1,000,000 runs with various aggregate functions, with a fixed \(\epsilon\) value of 0.1, where a benchmark TPC-H dataset was used. While in the comparison, Flex and PinQ were run only 10,000 times due to performance concerns. This evaluation points out that compared with Google DP, Flex and PinQ can not enforce contribution bounds for databases where one single user can contribute multiple samples, leading to query results that are not differentially private. Furthermore, because Flex or PinQ assumes that the underlying database is associated with at most one record per user, their performance comparison with Google DP that supports the contribution of multiple samples can be problematic.

1.1.2 Machine Learning

Differentially private machine learning (ML) has gained attention at companies like Google, Facebook, and IBM. Investigations have been done on the performances of DP stochastic gradient descent (SGD)\(^1\), which is a dominant algorithm for private training of ML models. Nevertheless, DP-SGD can increase the training time significantly compared to non-private SGD, i.e., it has a significant run-time overhead\(^1\). In a recent study, the Subramani et al. reduced the run-time overhead when executing DP-SGD\(^1\). They implemented the functionality in the open-source library of Tensorflow Privacy by exploiting language primitives\(^1\). Microsoft has shown support in the DP-ML field by implementing Opacus and demonstrating the impact of epsilon and dataset size on DP-ML\(^1\). However, no comparison to other tools was taken into consideration.

Beyond improvements regarding performance, there is also work recently to improve the privacy guarantees of DP-ML\(^1\) and evaluate different means of DP with different privacy budgets, which shows how the trade-off varies between utility and privacy under different settings. Their study focuses on gradient perturbation mechanisms, e.g., DP-SGD, and uses Tensorflow Privacy to evaluate Rényi Differential Privacy among others\(^1\). It demonstrates that the privacy guarantees of DP in practical machine learning implementations may provide unacceptable utility-privacy trade-offs. Their aimed to find epsilon values that balance utility and privacy for different DP approaches rather than for different DP tools. In another study, Tramér et al. provide an approach to improve performances of DP models using primarily Tensorflow and parts of the Opacus library\(^1\). They point out that prior works have underestimated privacy-utility guarantees and demonstrate that solid privacy may come at only a nominal cost in accuracy by tailoring the training to the data. However, this work has not yet been generalized as a tool that can be easily leveraged amongst the developer community.

\(^{2}\)The research project Privacy Integrated Queries (PinQ) is a programming language and execution platform in which all expressible programs satisfy DP\(^7\).
1.1.3 Synthetic Data

Differentially private (DP) synthetic data generation (SDG) is a relatively new field that has evolved in pace with the advancement of DP mechanisms in practical applications. It produces fake data which holds as many features in the real one as possible and enables the substitution of actual data in data analysis, while meeting the requirement of DP that does not leak actual data's information during synthetic data generation. DP SDG’s theoretical concepts have been adopted into practice and improved by new emerging techniques, which has contributed to the enhancement of data utility regarding the privacy-utility trade-off that appears in DP.

Recently, one study by Rosenblatt et al. got incorporated into OpenDP’s Smartnoise [9], where they evaluated five synthetic data generation techniques and showed their performances of data utility. Specifically, they surveyed a histogram-based approach called MWEM and four DP generative-adversarial-networks (GANs) for data synthesis (DPGAN, PATE-GAN, DP-CTGAN and PATE-CTGAN), using evaluating metrics of the distributional similarity and the ML efficiency on the generated data. Their work focuses on the utility of synthetic data on ML tasks and provides instructions on selecting data synthesis approaches in programming.

Though comparative results are provided, Rosenblatt’s study does not evaluate the performance in comparison with other DP tools, nor takes runtime and memory overhead into account, which is critical to show whether a DP service runs effectively. Additionally, while their study focused on evaluating the performance of classification and regression tasks by training ML models on the synthetic data, and evaluating them on the corresponding actual data [9] they do not explicitly explore how statistical query tools perform on synthetic data, which is an essential aspect of data-related services.

Recently, Knoors et al. evaluates and compares different SDG techniques [19], where synthetic data obtained from the different techniques are compared by varying epsilon and dataset size. Knoors’ evaluation performs one classification task, specifically train-synthetic test-real, and benchmarks how well the synthetic ML models can predict unseen test cases by misclassification rate and F-score. Their study brings some insight into the influence of data characteristics on the SDG technique’s computational complexity and reflects on different SDG techniques from a developer’s perspective.

1.2 Our contribution

We study a critical aspect of data systems, which is the protection of privacy-sensitive information. Our study reviews, evaluate, and provide preliminary guidelines for the selection of open source tools for differential privacy. The area of privacy protection has a well-recorded history of failed solutions. To the best of our knowledge, differential privacy is the only framework to offer qualitative guarantee to the protection of privacy-sensitive information. The implementation of tools for providing solution that are differential private has its own set of traps and pitfalls since it is a non-trivial effort to assure that all cryptographic and system aspects are well-addressed. A successful approach for meeting this challenge is to focus on open source solutions since they can be scrutinized by a large community of developers. To the best of our knowledge, we are the first to offer a comprehensive study that compares the performance of these tools from the application utilization and system perspectives.

\[^{3}\text{Specifically, train-synthetic test-real (TSTR) that they compare with train-real test-real (TRTR).}\]
Through this work, we develop an evaluating framework to compare privacy-preserving tools to get a nuanced picture of the trade-offs in a data analysis where the tools are used. The framework is implemented on Docker that is compatible with the dominant operating systems of Windows, Linux, and iOS and offers thus the flexibility to programmers for software reuse purposes. The designed framework uses the defined criteria to quantify an analysis’ utility loss and system overhead compared to a non-private benchmark. Using the devised framework, we evaluate the most relevant open-source tools, based on throughout tool review, and compare their performance on differentially private data analysis. Through the evaluation results, we provide insights into how the considered tools can be optimally configured, how their performance varies under different settings, and which tools developers are suggested to use according to their use-case. Finally, we release our work as open-source software enabling reuse of our work in further evaluation on the considered tools and beyond. We anticipate that this work can bring a landscape of the pros and cons of existing open-source privacy tools and an intuitive knowledge to practitioners on how to leverage them in their privacy protection service development, and ultimately narrow the gap between theoretical and applied research on DP.

2 Theoretical background on privacy and differential privacy

Differential privacy (DP) is a mathematical definition of privacy that, by introducing carefully calculated random noise, retains the statistical properties of the dataset while prohibiting that information of individuals contained in the data can leak during analysis. DP guarantees that an individual will be exposed to essentially the same privacy risk whether or not his or her data is included in a DP analysis. I.e., it ensures that personal private information cannot be revealed from the analysis on the database, regardless of the adversary’s compute power or access to any additional information that may exist, or will ever exist.

Formally, a randomised mechanism $M$ is said to give $(\epsilon, \delta)$-DP for all events $S$, $S \subseteq \text{range}(M)$, given two adjacent databases $D$ and $D'$, such that

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

(1)

Where $\text{range}(M)$ denotes the output range with a given input, and $\Pr[\cdot]$ denotes probability distribution. If $\delta = 0$, the randomized mechanism $M$ gives $\epsilon$-DP by its strictest definition. The parameter $\epsilon$ refers to the privacy budget, which controls the level of privacy guarantee achieved by mechanism $M$.

Informally, DP guarantees that the addition or removal of a single entry in a database essentially does not affect the result of any analysis or query and limits the risk of privacy disclosing associated with joining or refraining from joining a database [1].

3 Review on open source tools with differential privacy services

3.1 OpenDP Smartnoise

OpenDP Smartnoise has its roots in Harvard University Privacy Tools Project. This project gained experience in building and deploying PSI, a system developed to share and explore

https://github.com/anthager/dp-evaluation

https://privacytools.seas.harvard.edu/
privacy-sensitive datasets with privacy protections of differential privacy (DP), and ultimately contributed to their efforts towards Smartnoise. OpenDP Smartnoise has also incorporated insights from other DP tools, such as PinQ [11], ekteio [22], PrivateSQL [23], Fuzz [24], and LightDP [25]. While most of the tools like PSI and PinQ are research prototypes, OpenDP Smartnoise is now putting efforts into further developing DP concepts into production-ready tools. With those features, OpenDP Smartnoise has obtained significant popularity amongst the developer community, with more than two hundred stars in its open-source software repository.

![Diagram of high-level system layers](image)

Figure 1: (a): Diagram of high-level system layers, showing the three sub-projects that address individual architectural concerns and communicate via Protobuf messages; (b): Diagram illustrating a Protobuf that encodes an analysis composed of components, which are in turn made up of either operators or statistics.

The system of OpenDP Smartnoise has three layers: analysis construction, validation, and execution. The first layer is run by the **smartnoise-sdk** library, which contains Python language bindings and provides a programming interface for building and releasing analyses. The **smartnoise-sdk** provides compatibility and can be integrated with a range of databases, including PostgreSQL, MySQL, SQL server, and plain CSV files through Python’s Pandas module. The **smartnoise-core** written in Rust is responsible for the validation and execution layer. It consists of the main library **validator-rust** and **runtime-rust** where the analysis is executed, shown in Figure 1a. The **validator-rust** provides utilities for deriving and checking if conditions are sufficient for a DP analysis, e.g., to determine if the system releases DP data, as well as the scaling of noise, property tracking, and accuracy estimates. The **runtime-rust** is an execution engine for DP analyses on an arbitrary dataset. Those three layers communicate via Protocol Buffers (Protobuf) messages that encode an abstraction called an **analysis**, illustrated in figure [1b].

The **analysis** is a description of an arbitrary computation or a computational graph of instances of various **components**. Each component represents an abstract computation, e.g., a **Mean** component for aggregating data. When a component is a mechanism, e.g., the **LaplaceMechanism** component for privatizing data, it consumes a privacy budget. Mechanisms are building blocks used by **statistics** and are not capable of privatizing data on their own if placed in an analysis graph. In addition, each component in the analysis graph is either an **operator**, e.g., transformations, subsets, aggregations and joins, or a statistic, e.g., a **Mean** that may be composed of **Sum**, **Count** and **Divide** components. Furthermore, a statistic is the only component that can privatize data. A high-level diagram in Figure 2 shows how data flows on the basic components.

[https://github.com/opendp/smartnoise-core](https://github.com/opendp/smartnoise-core)
3.2 Google Differential Privacy

Recently, Google released an open-sourced version of the differential privacy (DP) library that empowers some of its core products [26]. Available in Java and Go, this library captures years of Google’s developer experience and offers practitioners and organizations potential benefits from their implementation, with a fairly low entrance level of expertise in DP.

In contrast to OpenDP Smartnoise, Google DP offers a PostgreSQL extension installed within the database engine itself instead of running as a separate server, illustrated in Figure 3. The PostgreSQL layer is a submodule in the C++ core library that implements noise addition primitives and DP aggregation.

Under the structure of Google DP, the `postgres` package implements a custom SQL syntax that requires a new semantics of queries. It also provides a collection of private aggregation operators, referred to as anonymous functions or ANONs, mapping to their corresponding DP mechanism in the core library. *e.g.*, `sum` aggregation is passed using the `ANON_SUM`. All mechanisms accessible in Google DP inherit from the `Algorithm` base class in the core library and are constructed according to the user’s need. Every time we extract a result from a mechanism, we use the privacy budget. The tools for tracking DP budgets are available as part of the Google DP library [7].

An essential property of Google DP is that it does not require that each user is only present in one single row [26]. Google DP uses a technique called *user bounded contribution* that, instead of

---

[7]https://github.com/google/differential-privacy/blob/main/common_docs/Privacy_Loss_Distributions.pdf
assuming that each user only contributes once, bounds how much each user can contribute. This row ownership is propagated through all subqueries for all SQL queries. Consequently, it requires that any join in a subquery also joins the user identifier. In applying this functionality, the Google DP PostgreSQL extension uses ANONs and a query rewriter, which is responsible for performing the anonymization and the validation of the query. In order to use these, the analyst applies the special syntax in the SQL query that triggers the query rewriter and allows the usage of the private operators.

Google DP system design comes with the advantage of saving the intermediate communication bandwidth between the database and the server when running the privacy application. Intuitively, the privatization inside the database itself might offer increased performance. However, letting the database handle DP computations might be undesired in systems with a heavy load on the database. In comparison, Smartnoise seems easier to scale horizontally since its stateless. Besides, GDP’s PostgreSQL extension is incapable of assessing standard SQL queries since uses ANONs to apply DP, and it also depends on PostgreSQL 11 and does not work with any other database engines.

3.3 Opacus

Opacus is Facebook’s library for differentially private (DP) machine learning (ML) services, built on top of PyTorch. It is developed in collaboration with Facebook AI Research, the PyTorch team, and OpenMined\(^8\), an open-source community dedicated to developing privacy techniques for ML and AI. The service from Opacus targets both ML practitioners and professional DP researchers with its general and specific features.

Opacus enables the training of DP ML models by applying DP stochastic gradient descent (DP-SGD). Its main component is the PrivacyEngine, which can be attached to the model training procedures to gain a ML model that satisfies DP. Specifically, during the model training, the gradients generated by the learning engine are clipped by the \(\text{max} \_\text{grad} \_\text{norm}\) parameter. Then the \(\text{noise} \_\text{multiplier}\) parameter adds a calculated amount of noise into the gradient to obtain the trained models that protect the training sample’s information from disclosure.

3.4 Tensorflow Privacy

TensorFlow Privacy (TFP) is a ML framework developed and released by Google, initially inspired by the work of Abadi et al.\(^{[12]}\), who implemented a similar optimizer for TensorFlow and a privacy cost tracker. It emerges and adapts differentially private (DP) mechanisms to TensorFlow to allow users to leverage differential privacy in the training of ML models. Furthermore, TFP is configurable that developers can define their own ML models, with which developers can implement their operators in their applications. With its flexibility and DP services, TFP has become an open DP tool that is leveraged and contributed by a large developer community.

TFP’s service wraps existing Tensorflow optimizers such as SGD and Adam into their DP counterparts. With those components, Gradients generated in TFP’s model training are clipped by the \(l2\_\text{norm} \_\text{clip}\) parameter, and the calculated amount of noise is added by the \(\text{noise} \_\text{multiplier}\) parameter. The optimizer adds Gaussian noise to the gradient at each round of training to achieve a DP ML model.

\(^8\)https://www.openmined.org/
It is worth noting that TFP includes a submodule and high-level framework called Keras, which has access to TFP’s mechanisms in the `tensorflow.keras` API. Keras was originally created and developed by Francois Chollet, and since TensorFlow 2.0, Keras has become the official high-level API of TensorFlow. As a result, it is now a suitable combination for users in a diverse range of industries and provides user-friendliness while accessing all low-level classes of TensorFlow.

### 3.5 Diffprivlib

Developed by the industrial giant IBM, Diffprivlib empowers differential privacy in machine learning tasks, including classification, regression, clustering, dimensionality reduction, and data regularization. It is supposed to be a general-purpose tool for conducting experiments, investigations, and application developments with differential privacy. With its detailed product manual, practitioners of different levels can easily find what they need during their interaction with Diffprivlib. As a result, Diffprivlib’s open repository has gained considerable attention amongst developers.

Diffprivlib applies the Python module Numpy and inherits services from the SciKit Learn library, extending its functionality by adding differentially private mechanisms on top of it. To clarify how Diffprivlib currently incorporate DP during the model training process, we present a high-level diagram in Figure 4. With Diffprivlib, a DP machine learning model can be trained by injecting noise into the cost function coefficients, which is executed by the Functional Mechanism as shown. Generally, the Functional Mechanism in Figure 4 is an extension of the Laplace mechanism that (i) adds noise to the coefficients of the objective function and (ii) derives the model parameter that minimizes the perturbed function.

![Figure 4: High level overview of how differential privacy is applied to the cost function](image)

Additionally, Diffprivlib provides `PrivacyLeakWarningCustom` warning to capture possible privacy leaks posed by incorrect parameter settings. I.e., when the user does not specify the bounds or range of data to a model, or if input data to a model falls outside the bounds or range initially specified.

### 3.6 Smartnoise Synthetics

The synthetics module of Smartnois is part of the software OpenDP Smartnoise, with the capacity to release synthetic data as a substitute for the actual data, at the same time satisfying the standards of differential privacy that protect the original data’s information during data generating.

---

9. [https://diffprivlib.readthedocs.io/en/latest/](https://diffprivlib.readthedocs.io/en/latest/)
10. [https://github.com/IBM/differential-privacy-library](https://github.com/IBM/differential-privacy-library)
At a high level, the synthetics module of Smartnoise consists of synthesizers and a sampler. In the data synthesizing, an implementation of the abstract class SDGYMBaseSynthesizer can be applied to create an instance of a synthesizer that trains a synthetic model. New rows are then generated by sampling the synthetic model. The above procedure is visualized in Figure 5.

![Figure 5: Overview of the synthetic data generation module of Smartnoise](image)

Smartnoise comes with three implementations of Synthesizers, (i) MWESynthesizer that is free from any external dependencies in Smartnoise, (ii) DPCTGANSynthesizer, and (iii) PATECTGANSynthesizer. The latter two use PyTorch and the CTGAN module from SDV for the generator and Opacus for the discriminator. The generator and discriminator are primary concepts in generative adversarial networks (GANs) that collaborate to substitute the original data. The visualization of the cooperation between the generator and the discriminator is shown in figure 6.

![Figure 6: Overview of the inner working of the DPCTGAN synthesizer in Smartnoise Synthetics](image)

### 3.7 Gretel Synthetics

Gretel Synthetics from Gretal.ai works as a tool to release synthetic data. Gretel synthetics tries to learn patterns in raw data text by using tokenizers, which encodes the characters in a text with integer-based IDs. Together with configuration for the underlying machine learning engine, these tokenizers will output TokenizerTrainers that trains on the underlying machine learning engine to learn patterns in the dataset and create a model to generate new synthetic dataset rows.

Gretel Synthetics offers robust solutions during data generation. However, since the model learns patterns directly from the raw text, there is a risk that rows contain invalid values or are poorly formatted, thus preventing the data synthesis. To address this problem, Gretel synthetics...
provides a row validator that only allows values in the original dataset, or if this approach is not good enough, the user can provide its own row validator. This flow is visualized in Figure 7.

While the founder of the tool seems less dominant than those mentioned above, like Google or IBM, Gretel Synthetics has shown sound attraction between developers [11], indicating its potential to be adopted in DP service applications.

Figure 7: Overview of the synthetic data generation with Gretel synthetics

3.8 ektelo

ektelo is designed to support interactive queries. It handles tasks including releasing contingency tables, histograms, range queries, answering online analytical processing (OLAP), and conducting private machine learning. ektelo is expected to help programmers implement varieties of differential privacy (DP) algorithms [22]. In particular, ektelo interprets different DP algorithms as combinations of operators that can operate basic functions. This interpretation manages to summarize privatizing procedures into a set of operator classes: transformations, query selection, partition selection, measurement, and inference. Different private programs from use-cases are described as plans over a library of operators, which is expected to be compatible with various differential privacy algorithms. ektelo also provides the implemented plans with proof of privacy, which releases developers from proving the differential private features of their programs. Among others, ektelo employs implementation techniques that allow programmers to scale to large data inputs using this software without incurring unwanted restrictions.

The software architecture contains two objectives: (i) isolating private interactions with raw data using a specified private server and (ii) modularizing private algorithms into operators, which promote code re-use and compactness. The code of private functionalities is run in a private server, which has access to the unaltered private data. The client, who requests the privatized query result from the private server, is assisted by a privacy engineer that mediates the client’s interaction with the private server. The total privacy budget is tracked for each query in the private server, and the client’s requests are satisfied until this budget has been exceeded.

Though ektelo is promising and unique as possessing both statistical query and machine learning services, it has not yet drawn sufficient attention within the open-source community [12], possibly due to its younger age of existence.

[11] https://github.com/gretelai/gretel-synthetics
[12] https://github.com/ektelo/ektelo
3.9 Chorus

Chorus utilizes a cooperative architecture to achieve DP statistical queries. It leverages industrial-grade database management systems (DBMS) for data processing tasks and even queries that need to be modified or, in some cases, entirely rewritten. This architecture has three primary components, namely rewriting, analysis and post-processing. The rewriting component is used to modify queries to perform functions like clipping, the analysis component to analyze queries to determine different properties such as required noise to satisfy differential privacy, and the post-processing component to process the result of the queries. An example from Chorus is the implementation of a summation mechanism with clipping. The rewriting component can modify the original query so that the DBMS executes the clipping and the summation, leaving the rest of the summation mechanism to the analysis and the post-processing component.

What separates Chorus from previous work is that it is DBMS-independent. Unlike an integrated approach, Chorus does not require modifying or affecting the database or changing to a purpose-built database engine. This is why Chorus can leverage DMBS to ensure scalability when working on datasets that hold large amounts of data.

Added safeguards can be necessary when deploying Chorus to minimize the chance of a malicious actor acquiring sensitive data. For example, in the case of Chorus’ deployment on Uber, sensitive data was only available through a centralized query interface, which was protected along with the privacy budget account and the DMBS from tampering.

While the early repository of Chorus archived by Uber is deprecated, a new version emerged that is maintained and active among the open-source community. The new repository of Chorus has not gained much attention as its relatively new existence; however, since Chorus showed strength in its early version, and more concrete results and popularity among developers can be visioned for its current version.

4 Evaluation settings

4.1 Tool selection and data adoption

We evaluate seven differential privacy tools shown in Table 1. Those tools are considered since they have support from comparably larger communities, technology companies, or research institutions and have gained wider acceptance in the open-source community. They also possess theoretical backup and sufficient documentation that is friendly to practitioners. Therefore, they have more potential to be adopted in privacy-preserving application developments and more possibility to influence a broader scope of privacy services.

The considered tools are evaluated within three domains, i.e., statistical query, machine learning, and data synthesis. Since tools of different categories follow different evaluating procedures, we analyze the evaluation results within each single domain. During the evaluation, we adopt diversities of settings to look into how different tools’ performances vary under different conditions. We summarize the evaluation strategy as a whole in Table 2 for simplicity. In this evaluation, we use the open-source data of United States Health Reform Monitoring Survey data for the experiments in statistical queries, and the UCI Parkinson dataset for machine learning experiments.

---

13 https://github.com/uber-archive/sql-differential-privacy
14 https://github.com/uvm-plaid/chorus
15 https://gretel.ai/synthetics
Table 1: Differential privacy tools with publicly available open source repositories

| Name                  | Domain                        | Origin          | Repository                                          |
|-----------------------|-------------------------------|-----------------|-----------------------------------------------------|
| Diffprivlib           | Machine learning              | IBM             | [https://github.com/IBM/differential-privacy-library](https://github.com/IBM/differential-privacy-library) |
| Google Differential Privacy | Statistical query         | Google          | [https://github.com/google/differential-privacy](https://github.com/google/differential-privacy) |
| OpenDP SmartNoise     | Statistical query & Data synthesis | Microsoft & Harvard | [https://github.com/opendp/smartnoise-core](https://github.com/opendp/smartnoise-core) |
| Pytorch Opacus        | Machine learning              | Facebook        | [https://github.com/pytorch/opacus](https://github.com/pytorch/opacus) |
| TensorFlow Privacy    | Machine learning              | Google          | [https://github.com/tensorflow/privacy](https://github.com/tensorflow/privacy) |
| Gretel Synthetics     | Data synthesis                | Gretel Labs     | [https://github.com/gretelai/gretel-synthetics](https://github.com/gretelai/gretel-synthetics) |

Table 2: Evaluation strategies in this work

| Environment | Machine learning domain | Statistical query domain | Data synthesis domain |
|-------------|-------------------------|--------------------------|-----------------------|
| Docker      | Prediction accuracy reduction compared with non-private machine learning. | Decreased query accuracy due to privacy measures. | Both prediction accuracy reduction in machine learning task and accuracy decrease in statistical queries. |
| Task to evaluate | Regression. | Regression; Sum; Count; Average; Histogram. | Regression; Sum; Count; Average; Histogram. |
| Influencing factors | Dataset size; Privacy budget $\epsilon$. | | |
| Evaluation criterion | Utility | Overhead | Database to evaluate on |
| Utility | Extra resource consumption induced by differential privacy including: Execution time; Memory consumption. | U.S. health reform monitoring survey data; UCI Parkinson data set. |
4.2 Evaluation criterion

Information of individual data might be disclosed from data analyses. However, tools that prevent this disclosure by leveraging differential privacy might ultimately cost analysis accuracy and increase the system resources \[^{[9,15]}\]. Therefore, our focus is to study the difference between the differentially private (DP) and non-privacy protected (NP) results that these tools produce and how this difference varies between different tools.

We adopt data utility (also referred to as accuracy) and system overhead as evaluating metrics for the considered tools. Utility refers to whether the data is still useful to conduct a specific functionality after the data is perturbed with differential privacy (DP) measures. I.e., how much an outcome deviates from the actual quantity it attempts to estimate. E.g., what degree of accuracy reduction incurs when querying on the perturbed dataset compared with that on the original dataset. System overhead means the additional time and memory it takes to complete a differentially private (DP) query or train a DP-ML model, versus the non-privacy protected (NP) query or ML model. The overhead is further divided into two metrics of Memory overhead and Run-time overhead, which will be detailed below.

The metrics of utility and overhead illustrate what deviations can be expected from the DP results of the tools, and allow for comparison amongst tools’ performances. The criterion utility is quantified by the deviation of the prediction error over several runs of the same experiment, using the Root Mean Square Percentage Error (RMSPE). This is essentially the percentile difference between the DP and NP results shown in Equation (2).

\[
\text{RMPSE} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left( \frac{NP - DP}{NP} \right)^2} \cdot 100 \%
\]

where \( N \) is the number of experiment runs, \( NP \) is the NP benchmark result, and \( DP \) is the DP result.

Memory overhead is measured by comparing the worst-case memory usage between DP and NP query/ML tasks to guarantee minimum system requirements for the tools. I.e., the measurement shows the percentile difference between the worst-case memory usage of DP vs. NP query/ML results. Specifically, memory usage is recorded when the tools conduct DP and NP query/ML tasks. This criterion of memory overhead is considered in this evaluation since it can be noticed by users who care usability of the privacy tools. E.g., lower memory usage ultimately improves speed due to less paging, fewer cache misses, and faster structure traversals, and it also improves stability by reducing virtual and physical out-of-memory aborts. Moreover, for specific tools that use an external database, including Smartnoise and Google DP, the memory usage of the database container is also recorded. This recording will show how the tools affect the memory usage during load on the database since we are running queries on a high frequency during this process.

Run-time overhead is measured by comparing the time passed before and after entering the critical section, i.e., the part where the tool does a ML task or runs a query. To minimize external impact and obtain reliable and comparable results throughout the experiments, we aim to eliminate operations like initialization or saving results to the highest extent. Similar to the memory usage, we get the difference between the time passed before and after, comparing the DP and NP time usage. However, instead of comparing the maximum run-time, we average the results by applying RMSPE shown in Equation (2).
4.3 Evaluating framework

![Diagram](image)

Figure 8: High-level overview of the experiment flow, showing 1) DP queries and non-privacy protected (NP) queries being conducted on Postgres, 2) queries being conducted on synthetic data, and 3) ML tasks being conducted on synthetic data.

We construct our evaluating framework in pursuit of insights into the impact of DP tools’ privacy measures on accuracy and system resource usage. To this end, we measure the difference between DP and NP results regarding our evaluation metrics: data utility and system overhead. In addition, we vary two parameters that affect these metrics, the privacy budget ($\epsilon$) and dataset size, to further illuminate the privacy-utility trade-off induced by differential privacy measures.

We study the impact of $\epsilon$ and dataset size since they are the parameters that trade-off between privacy and data utility. We illuminate this trade-off to provide developers or DP service practitioners, e.g., healthcare institutions and companies, with practical results to make educated choices when applying these tools. The set of sizes for considered datasets where the evaluation experiments are conducted is listed below.

| Dataset sizes       |
|---------------------|
| Health Survey size  |
| $\in \{1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 9358\}$ |
| Parkinson size      |
| $\in \{1000, 2000, 3000, 4000, 5000, 5499\}$ |

Table 3: List of dataset sizes for each dataset.

Our selection of $\epsilon$ values in the evaluation takes consideration of recommendations both from research works and practical settings in the industry. While theoretical research has evaluated DP algorithms using a privacy budget ranging from 0.01 to 7, industrial practitioners prefer a narrow scale. E.g., Apple Health in iOS 10.2 use $\epsilon = 2$ for gathering what health data types are being edited by users. Microsoft, in collaboration with OpenDP, explains in their product Azure that privacy budgets are typically set between 1 and 3 to limit the risk of re-identification. They

16https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf
17https://docs.microsoft.com/en-us/azure/machine-learning/concept-differential-privacy#differential-privacy-metrics
state that $\epsilon$ values below 1 provide full plausible deniability and that values above 1 come with a higher risk of disclosing the actual data.

In order to cover a wide range of privacy-utility trade-off results, we use the practice of $\epsilon$ as guidelines for selecting a set of $\epsilon$ in our evaluation experiments. The set of $\epsilon$ values considered is listed in Table 4.

| Epsilon (\(\epsilon\)) values | \(\epsilon \in \{0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0\}\) |
|-------------------------------|------------------------------------------------------------------|

Table 4: List of $\epsilon$ values.

Since different DP tools function and conduct computation with different techniques, we group the tools by the service that they offer into three domains, *statistical queries*, *machine learning*, and *data synthesis*. This categorization allows for a reasonable comparison of tool performance within each group of tools.

For the evaluation of statistical query tools, we select a set of queries to conduct on each column in the dataset as listed in Table 5. Note that the HISTOGRAM queries are only conducted on columns composed of categorical values since the statistics of the categorical values can be sorted into buckets. We carry out linear regression tasks within machine learning tools since regression is the only service that all the considered ML tools have in common. Finally, we evaluate data synthesis tools by testing query/ML tasks on their synthetic data, generated based on the real one, and compare how the testing results differ from the query/ML results obtained from the original data.

| Queries | \(qs \in \{\text{SUM}, \text{AVG}, \text{COUNT}, \text{HISTOGRAM}\}\) |
|---------|--------------------------------------------------|

Table 5: List of queries.

To clarify how we conduct our experiments for the different tools, we present a high-level diagram shown in Figure 8 illustrating how data flows through the different tools and generates results.

### 4.4 Experiment implementation

We aim to enable the reuse of our framework to the full extent and allow users to develop and test our code on any environment without installing and handling dependencies locally. Therefore, we build a collective framework for all the tools, focusing on usability and portability.

To make sure that our code works across different environments, we develop the evaluation framework on Docker\(^{18}\) that packages each tool package and its dependencies in a virtual container. Docker also eases the memory usage measurements by exposing a RESTful API running on the host system through a UNIX socket, from which metrics such as memory usage can be fetched.

Each tool evaluated is packaged in its own docker image together with all the packages the tool depends on, along with the framework code. This implementation empowers evaluations using the framework to run on any Linux, Windows, or macOS computer. The complete evaluation, therefore, only requires Docker version 20.10.6 or higher.

\(^{18}\)https://en.wikipedia.org/wiki/Docker_(software)
Figure 9: High-level overview of experiment implementation, showing how data flows in the evaluation framework.

To visualise how the framework is constructed, we present a diagram in Figure 9. It shows the dataflow in the framework as follows: (1) Data and Metadata (with experiment- and hyper parameters) are loaded in Context class, which is (2) initialised in Tester script for respective tool. (3) Tester makes use of various utils from a collective package DPEvaluation. (4) The results are saved after running Tester, whose outcomes are collected by (5) Plot Builder with Aggregator, both of which are parts of the collective utils package.

5 Evaluation results

This section presents the results of our evaluation, where we show to what level vanilla functionalities are affected when differential privacy is integrated under different open-source differential privacy tools. The evaluation is carried out in three tool domains, namely statistical query, machine learning, and data synthesis, which are supposed to cover the need of private service development sufficiently.

We show in the evaluation results that, when running different privacy tools, to what level a particular data-related analysis’ accuracy is reduced, and how much extra execution time and memory usage are induced compared with its non-privacy version. We use diversities of experiment setting to gain a comprehensive insight into how the considered tools trade-off between privacy, utility, and resource usage, and generate plots on the results, provide analyses, and summarise potential guidelines on how to select privacy tools in development practice, as well as suggestions that help to configure privacy tools optimally.

This section might be valuable for readers within developer communities seeking guidelines for selecting and configuring differential privacy tools in their privacy service development. The evaluation results can also provide the academic society with a quantitative assessment of the
effectiveness of dominant privacy tools, which ultimately helps bridge the gap between theoretical and practical differential privacy research. We detail the evaluation results as follows.

5.1 Statistical tools assessment

This section evaluates two tools with statistical query services, namely Google Differential Privacy (Google DP) and OpenDP Smartnoise, using the two considered databases as shown in Table 2. The evaluation varies privacy budget $\epsilon$ and data size during the experiments and looks into how the query results of \texttt{SUM}, \texttt{AVERAGE}, \texttt{COUNT}, and \texttt{HISTOGRAM} change when differential privacy mechanism is integrated. In the evaluation, each query runs 20 times, of which two extreme results are removed, and the remaining 18 are averaged for analysis.

Generally, the results reflect the trend that utility increases given larger $\epsilon$ and data size for both tools, yet no explicit correlation between memory overhead, $\epsilon$, and data size can be observed. There also exits an opposite impact of data size on the run-time overhead for the two tools that, while larger data size brings an increase in run time for Smartnoise, it acts conversely for Google DP, though significant irregularities exist. In comparing the two query tools, we observe that Google DP offers better query accuracy than Smartnoise in all query types except \texttt{HISTOGRAM} and that Google DP induces significantly less run-time than Smartnoise when comparing their DP queries against the benchmark. The overall results indicate an advantage of Google DP on Smartnoise under limited conditions. We detail the quantified evaluation results as follows.

5.1.1 Data utility

Evaluation in this part studies the differential privacy (DP) query tools’ impact on utility by comparing how DP query results differ from non-privacy-protected ones using different settings.

In the evaluation, we anticipate that higher privacy budget $\epsilon$ and larger data size will provide better utility since less noises are injected under such conditions. However, as detailed below, though the experimental results match our anticipation, we notice local irregularities, \textit{e.g.}, in the case of \texttt{HISTOGRAM} queries.

Figure 10 shows contour plots for each tool’s performance regarding different $\epsilon$ and data size. Darker shades of blue in the plots indicate a lower RMSPE, corresponding to higher utility. Conversely, lighter shades indicate a larger RMSPE, corresponding to lower utility. Note that the plots have different RMSPE scales, implying that a shade in one plot (which indicates a RMSPE value) does not necessarily correspond to the same shade in another plot.

The contour plots demonstrate that, for simple queries of \texttt{COUNT}, \texttt{SUM}, and \texttt{AVG}, Google DP bears RMSPE between 0.1% and 20%, indicating 0.1%-20% worse than the benchmark, while Smartnoise performs between 0.2% and 350% over the considered parameter ranges of $\epsilon$ and data size. However, Smartnoise performs better than Google DP on the \texttt{HISTOGRAM} queries with RMSPE between 0.5% and 60%, compared to Google DP’s between 0.2% and 250%. Such results imply Google DP’s advantage over simple query types while converse in the \texttt{HISTOGRAM} query. We also observe that \texttt{HISTOGRAM} queries generally have a more significant impact on data utility than other types of queries for Google DP and Smartnoise. A possible reason might be that \texttt{HISTOGRAM} queries expose more information about the dataset properties; as a result, more noise is injected into the \texttt{HISTOGRAM} results to guarantee the privacy of individuals, which in turn reduces more data utility.

Moreover, the results show that \texttt{HISTOGRAM} queries obtain better results on \textit{Parkinson} than on \textit{Health Survey} for both tools, which is expected since the categorical columns in \textit{Parkinson}
Figure 10: Contour plots for the evaluation of statistical query tools on utility when DP is integrated, for different \( \epsilon \) (Table 4), data size (Table 3) and queries (Table 5). RMSPE is defined in Section 4.2.

have fewer bins than Health Survey data, thus exposing less information about Parkinson data properties. Consequently, less noise is injected into the results on Parkinson data in pursuing individuals' privacy, and higher query accuracy is obtained.

Figure 11 shows more details on the results that higher \( \epsilon \) values decrease the RMSPE (see
the definition in Section 4.2, implying higher utility of DP queries is obtained where higher $\epsilon$ values improve accuracy for all queries of \textsc{SUM}, \textsc{AVERAGE}, \textsc{COUNT}, and \textsc{HISTOGRAM} for both of our two considered datasets. This observation is quite explicit, especially for results on \textit{Parkinson}. As anticipated, the relationship that data utility grows with the increase in data size can also be observed. Generally, the results on \textit{Parkinson} are more consistent with our anticipation, while those for \textsc{HISTOGRAM} queries on \textit{Health Survey} data show marginal levels of fluctuation.
5.1.2 Run-time overhead

In this subsection, we illustrate how the run-time differs between conducting differential private (DP) queries and non-private ones by testing the tools of Google Differential Privacy and Smartnoise. Note that we vary the settings of \( \epsilon \) and data size to gain evaluating results under different conditions.

For this evaluation, we anticipate that the run-time overhead might increase when DP is integrated since DP requires additional computations to conduct the query. Intuitively, the run-time for DP queries is also expected to grow with an increase in data size since more information will be processed. The results, as detailed below, demonstrate that Google DP poses less run-time than Smartnoise, while how the two tools are impacted by DP differs. I.e., Smartnoise experiences an increase when data size rises, while Google DP reacts in the opposite. Beyond that, we observe apparent fluctuations in the results and no clear relationship between \( \epsilon \) and run-time.

In Figure 12, we provide contour plots for each tool’s run-time overhead regarding \( \epsilon \) and data size. The plots show that larger data size generally increases run-time for Smartnoise, i.e., between 400\% to 500\% RMSPE for the smallest data sizes and about 450\% to 550\% for the largest. In contrast, Google DP performs does not follow our anticipation with RMPSE between 110\% to 150\% for the smallest data size, and 100\% to 130\% for the largest. Overall, Google DP outperforms Smartnoise, which runs around 400\% to 500\% slower when conducting DP queries, compared to Google DP that runs around 100\% slower. A possible reason is that Google DP performs more efficient DP calculations, using a plugin inside the database compiled to native code (Section 3.2). Therefore, the run-time might improve, since no additional layer operates between the database and the analyst that conducts the queries. In comparison, Smartnoise implements pre-processing of queries before communicating with the database (Section 3.1), which might negatively impact on the run-time.

Figure 13 shows how \( \epsilon \) impacts query accuracy for both Google DP and Smartnoise for both data sets. It reveals that higher \( \epsilon \) values do not necessarily decrease or increase the RMSPE, indicating that \( \epsilon \) values do not generally impact the run-time of DP queries. Though slight local fluctuations exist, this observation is quite explicit and holds for both tools and datasets, especially obvious for Google DP on Health Survey.

5.1.3 Memory overhead

This subsection looks into how differential privacy impact memory usage when running statistical queries on Google DP and Smartnoise. We use various \( \epsilon \) and data sizes to see how the results differ under different settings. To gain a nuanced result, we measure the memory usage in both the container of the Postgres database where noise is added and processed data are stored, and the container that processes the issued private or non-private queries to the database.

In this evaluation, we anticipate the memory overhead for running queries to increase when DP is integrated, and that the memory overhead grows as the data size rises since more data are involved in the calculation procedures. As elaborated below, the results show that DP generally poses an additional memory usage of less than 3\% for the operations in the Postgres database, while the processing of private queries poses 5\%-40\% extra memory consumption. However, the results disclose no clear relationship between \( \epsilon \), data size, and memory overhead, and fluctuations of different levels exist throughout the results. In comparison, there is no apparent advantage of one tool over another regarding memory overhead, yet Google DP slightly outperforms Smartnoise.
Figure 12: Contour plots for the evaluation of statistical query tools on run-time overhead, for different $\epsilon$ (Table 4), data size (Table 3) and queries (Table 5). RMSPE is defined in Section 4.2 on the Health Survey data composed of categorical variables.

Figure 14 shows contour plots for each tool’s performance on memory overhead regarding different $\epsilon$ and data size. While the plots reveal no explicit patterns, the results on HISTOGRAM manifest a slight trend for the Postgres database operation in Smartnoise that memory overhead grows with an increase of data size (Figure 14d, 14h), which holds for both data sets. However, we also observe
Figure 13: The evaluation results of statistical query tools on run-time overhead when DP is integrated, for different \( \epsilon \) (Table 4), data size (Table 3) and queries (Table 5). RMSPE is defined in Section 4.2.

a significant impact of data size on the query processing than that of \( \epsilon \) (Figure 14a, 14c, 14e, 14g), where the value of \( \epsilon \) does not necessarily affect memory overhead, and the data size irregularly influences the memory overhead.

Figure 15 details the impact of DP on the Postgres database, from which we can not summarize any clear correlations between \( \epsilon \), data size, and memory overhead. In general, the induced memory
usage on the operation of the Postgres database is notably low (≤ 3%) for both tools and data sets, which indicates a marginal impact. In comparison, though irregularities exist, Google DP shows lower peaks (less than 2) in the delta axis than Smartnoise (between 2 and 3) in the Health Survey experiments, implying an advantage of Google DP on categorical data set over Smartnoise. In contrast, this advantage is not evident in the Parkinson evaluation.

Figure 16 describes how DP impacts the processing of queries issued to the Postgres database regarding memory overhead. We cannot observe any general relationship between $\epsilon$, data size, and memory overhead through this figure. However, the results show an overhead of 5%-40%, indicating more memory consumption of querying procedures than database operations with less than 3% memory overhead (see Figure 15). The results also suffer from fluctuations, especially for the HISTOGRAM query on Parkinson by Google DP, whereas Smartnoise performs more stably though no better in that case.

5.2 Machine learning tools assessment

We evaluate three machine learning tools with the provision of differential privacy service, i.e., Tensorflow Privacy, Opacus, and Diffprivlib, and investigate how their private models differ from non-private ones learned from two data sets as shown in Table 2. This evaluation considers the regression model for all the tools, which we instantiate as a linear regression model since regression is the only functionality the considered tools hold in common. Furthermore, we vary the privacy budget $\epsilon$ and data size in the evaluation to see how the results differ regarding utility, run-time overhead, and memory overhead defined in Section 4.2. Note that each model training runs ten times, of which two extrema results are removed, and the remaining eight are averaged for analysis.
Figure 15: The evaluation results of statistical query tools on memory overhead when DP is integrated, for different $\epsilon$ (Table 4), data size (Table 3), and queries (Table 5). Specifically, the results on memory overhead in the database (Postgres) that the queries are conducted on. $\delta$ is defined in Section 4.2.

The results generally manifest the trend that the integration of differential privacy in model training induces model accuracy reduction, and this reduction lowers with an increase in either $\epsilon$ or data size, which holds for all tools on both Parkinson and Health Survey data except Diffprivlib on Parkinson data, where no useful result is obtained. We also observe that Tensorflow Privacy poses less memory overhead for the larger data size of the considered data sets, while no clear relationship exists between $\epsilon$, data size, and run-time overhead. In comparison, Opacus induces less model accuracy reduction than Tensorflow Privacy and Diffprivlib, given $\epsilon \leq 0.5$ for both
Figure 16: The evaluation results of statistical query tools on memory overhead when DP is integrated, for different $\epsilon$ (Table 4), data size (Table 3) and queries (Table 5). $\delta$ is defined in Section 4.2.

data sets, while Tensorflow Privacy outperforms Opacus and Diffprivlib on the continuous data set of Parkinson within a wide range of privacy budget ($0.5 \leq \epsilon \leq 3.0$). The results also indicate that Tensorflow Privacy poses less run-time, and Opacus adds less memory usage when differential privacy is integrated in model training. The quantitative evaluating results are detailed in the following.
5.2.1 Data utility

Evaluation in this subsection investigates how a differentially private machine-learning model differs from a benchmark regarding model accuracy to show the trade-off between privacy and utility when a machine learning task is combined with differential privacy (DP) under different experimental settings.

In this evaluation, we expect that higher $\epsilon$ values and larger data size will provide better utility for the considered data sets since such conditions cause less noise added during DP model training. As anticipated, the results show that the trained machine learning model brings better accuracy as $\epsilon$ and data size increase. However, along with this expected trend, there also exists local irregularities, and that Diffprivlib incurs severe accuracy reduction on the Parkinson data making it far from useful in that case. We further detail the evaluation results below.

The contour plots in Figure 17 describe how the learned model’s utility measured by RMSPE varies regarding $\epsilon$ and data size. In general, the plots corroborate that utility grows with larger $\epsilon$ and data size, though there are irregularities in the Parkinson results by Opacus, where the data size 4000 experiences slight increased RMSPE than lower data size. Also, $\epsilon$ does not necessarily affect Opacus’ modeling utility on Parkinson when $\epsilon \geq 1.0$. We also observe that in the Health survey experiments, Opacus exhibits marginal less RMSPE (generally $\leq 6$) than Tensorflow Privacy (generally $\leq 10$), both of which perform much better than Diffprivlib ($\geq 10$ under most of the settings); However, Tensorflow Privacy provides obvious less RMSPE than Opacus and Diffprivlib on Parkinson data, as shown in Figure 17d 17e 17f, indicating Tensorflow Privacy’s advantage on continuous data.

Figure 18 depicts model accuracy reduction under different $\epsilon$, where we observe that higher $\epsilon$
values decrease the RMSPE, implying less accuracy reduction of DP-integrated-model compared with the benchmark. This trend holds for the two considered datasets by Tensorflow Privacy and Opacus, while Diffprivlib provides RMSPE of over $10^8$ on the Parkinson data (Figure 18g, 18h, 18i), making it unacceptable under this scenario. Therefore, we neglect the experimental results of Diffprivlib on Parkinson in the following analyses. Even though, it is worth noting that Diffprivlib generates comparably equal accuracy to Opacus and Tensorflow Privacy when no DP is applied on both datasets. In comparison, Opacus outperforms Tensorflow Privacy and Diffprivlib when $\epsilon \leq 0.5$. However, since RMSPE rises abruptly when $\epsilon$ decreases away from 0.5, the advantage of Opacus here gets insignificant. In contrast, Tensorflow Privacy produces less accuracy reduction for Parkinson data within a wide range of $\epsilon$ (0.5-3.0), implying its better performance on continuous data set than Opacus and Diffprivlib.

5.2.2 Run-time overhead

Evaluation of this part presents how the considered machine learning (ML) tools perform when combined with differential privacy (DP) regarding the run-time overhead induced due to DP. We
vary the experimental settings to see how the results differ under various conditions.

The evaluation is anticipated to observe an increase in DP machine learning’s run-time compared with the benchmark, and we also expect that the run-time overhead grows as the data size rises in the experiments since more data is processed. The results, as detailed below, show that Tensorflow Privacy poses the least run-time increase compared with Opacus and Diffprivlib, with the general RMSPE of $\leq 40$ in run-time for Tensorflow Privacy versus $200 - 230$ for Opacus and $400 - 2000$ for Diffprivlib. Note that we did not display the results of Diffprivlib on Parkinson data since there is no useful result generated, as elaborated in Section 5.2.4. Beyond that, we observe no clear trend between data size, $\epsilon$, and run-time overhead.

![Contour plots for the evaluation of DP ML tools on run-time overhead for different $\epsilon$ (Table 4) and data size (Table 3). RMSPE is defined in Section 4.2.](image)

Contour plots in Figure 19 describe each tool’s run-time overhead regarding different $\epsilon$ and data size. Through the plots, we observe that both $\epsilon$ and data size affect the run-time overhead in an irregular manner, where no explicit patterns can be concluded. The results also experience expected fluctuations and irregularities which are hard to explain, e.g., the abrupt increased RMSPE for Tensorflow Privacy on Health survey when $\epsilon = 2$ and data size $= 6 \times 10^3$, and the sudden decreased RMSPE for Opacus on Parkinson when data size $= 3 \times 10^3$. Even though, it is clear that Tensorflow Privacy incurs less RMSPE due to DP in machine learning, compared with Opacus and Diffprivlib on both data sets.

Figure 20 illustrates the induced run-time under different $\epsilon$s on the Health survey and Parkinson data for all the ML tools. The graphs reveal no relationship between $\epsilon$ and run-time during model training of all the considered ML tools, while the results on Health survey demonstrate a stable induced run-time by Tensorflow Privacy and Opacus. However, as shown in Figure 20a, local irregularity exists for Tensorflow Privacy. Overall, Tensorflow Privacy significantly outperforms Opacus and Diffprivlib regarding run-time on both the considered data sets.
5.2.3 Memory overhead

This subsection investigates the additional memory usage posed due to the integration of differential privacy (DP) in machine learning (ML). We conduct experiments using various settings to look into how the considered ML tools perform in DP ML models compared with non-private ones.

For the experiments, we anticipate an increase in memory usage in the training of DP ML models compared with non-private ones since more DP models involve more computation during the model training. As detailed below, the experimental results demonstrate extra memory usage of different levels for all the considered ML tools due to DP. Particularly, Tensorflow Privacy suffers the most memory usage (15%-20% for Health Survey and above 70% for Parkinson) compared with Opacus (below 3.75% for Health Survey and below 4.0% for Parkinson). On the other hand, Diffprivlib generally brings an additional memory usage of 5%-10% for Health Survey, yet it does not provide useful results for Parkinson. Overall, Opacus shows more advantage in memory overhead for DP machine learning.

Figure 21 shows contour plots of each tool’s memory overhead regarding different ϵ and data size. From the plots, we can observe that TFP has a memory overhead of about 90% for the Parkinson dataset for nearly all the data size and ϵ, and about 20% for the Health Survey dataset (Figure 21a-21c), though there is an unexpected decreased memory usage for the highest data size of both Health Survey and Parkinson. Opacus (Figure 21f-21h), on the other hand, has less than 5% memory overhead in both of the datasets, while irregularities exist for some data size and ϵ combinations for Health Survey, and an increased memory usage around the data size of 2000 for Parkinson. Diffprivlib gains memory overhead mainly around 10% and experiences a slightly decreased memory usage under the smallest data size and a significant decrease under the highest data size for Health Survey. We also note that the value of ϵ has no apparent influence on the
results for all the three considered tools.

Figure 21: The results of the impact of ML tools on memory overhead for different $\epsilon$ (Table 4) and data size (Table 3). RMSPE is defined in Section 4.2.

Figure 22 shows representative results on how memory overhead varies over different combinations of data size and $\epsilon$. Beyond the conclusion from Figure 21, we can also observe a higher memory overhead for the Parkinson than that of the Health Survey for Tensorflow Privacy, indicating its better memory usage for categoric data than continuous data. In contrast, Opacus performs stably for both the two data sets with the lowest memory overhead.

5.3 Data synthesis tools assessment

Data synthesis tools present a way of substituting original data to hinder privacy-sensitive information during data analysis. The synthetic data generated from the original dataset aims to retain as many statistical properties as possible, while individual rows in the synthetic data do not belong to any of the rows in the original version. When integrated with differential privacy, such tools can provide formal privacy guarantees that the synthetic data generation (SDG) procedures do not disclose individual privacy [34, 35].

This subsection investigates the impact that SDG tools have on data utility and overhead when integrated with differential privacy (DP). We evaluate two tools for generating DP synthetic data: Gretel Synthetics and Smartnoise Synthetics. Smartnoise Synthetics uses three algorithms, or synthesizers named internally, to generate synthetic data. These are MWEM, PATECTGAN, and CTGAN, each of which will be evaluated individually in this evaluation and compared. Using those data synthesizers, we generate synthetic versions of two datasets: Health Survey and Parkinson (Section 4.1) under different settings, and then measure how much the results of statistical queries and ML tasks on synthetic datasets deviate from those conducted on the original (non-privacy protected (NP)) dataset.
We note that Gretel Synthetics and all the synthesizers in Smartnoise Synthetics provide plenty of hyperparameters for tuning the training process. However, it is not trivial how to optimally configure them. Therefore, our framework allows us to provide different values for each hyperparameter and thus generate a synthetic dataset for each combination of the provided values. After that, we select generated synthetic datasets using the Chi-squared test, which ranks the datasets by comparing the distributions of each column [36]. We do not have to know how to configure the tools using this method. Instead, we provide a list of hyperparameters, let them run, and select the best-performing synthetic dataset according to the Chi-squared test, given the parameter alternatives.

We also note that Gretel Synthetics fails to provide functionality for targeting specific $\epsilon$ values. Even though the library is built around Tensorflow, we could not access the functionality that allows us to compute the noise multiplier value for target $\epsilon$ to flexibly settle the privacy budget, as we do for Tensorflow Privacy. Instead, it runs with a set of hyperparameters and outputs the final $\epsilon$ value at the end of the data generation. Therefore, we do not have results for targeting $\epsilon$ values in experiments on Gretel, as we do for the other tools. Also, note that Gretel Synthetics outputs $\epsilon$ values significantly higher than the $\epsilon$ values that we set for the other tools. To make a reasonable comparison, we generate many synthetic datasets for each data size and with different hyperparameters, select the synthetic dataset with the lowest $\epsilon$ value, and use that for our comparison. Thus, instead of having one synthetic dataset for each combination of $\epsilon$ and data size, we have one synthetic dataset for each data size for Gretel Synthetics, as shown in Table 6.

The following subsections show how different data synthesizers perform for statistical queries and machine learning tasks under different settings regarding utility and overhead when differential privacy measures are integrated.
### Table 6: List of the lowest $\epsilon$ values obtained for each data size of the synthetic versions of the Health Survey dataset, using Gretel Synthetics.

| Dataset size | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 9358 |
|--------------|------|------|------|------|------|------|------|------|------|
| $\epsilon$   | 24.6 | 32.6 | 32.0 | 31.0 | 30.0 | 30.0 | 29.0 | 28.0 | 55.3 |

5.3.1 Statistical query utility

This subsection evaluates the considered data synthesis tools on whether they work efficiently in the context of differential privacy. We perform this evaluation by performing statistic queries on both original data and its synthetic version generated by the synthesis tools. We then analyze the difference between the results and how differential privacy measures impact different tools during synthetic data generation (SDG).

We generate the same number of data records for the SDG tools application as in the original dataset, which implies that there will not be any difference conducting the \texttt{COUNT} query on the synthetic dataset vs. the original and that the RMSPE (Definition 2, Section 4.2) of the \texttt{AVERAGE} and the \texttt{SUM} query will be the same. Consequently, we only include result plots for the \texttt{AVERAGE} and \texttt{HISTOGRAM} query.

Since higher $\epsilon$ values induce less noise added to the data and a larger data set implies that individual data items contribute less to the data analysis [26], we anticipate that synthetic datasets, generated with higher $\epsilon$ values and larger data size, will provide better utility. However, as detailed below, the evaluation results do not show any clear trends that are precisely consistent with our anticipation, and irregularities exist significantly.

The contour plots in Figure 23 show each tool’s performance regarding different $\epsilon$ and data size. The white areas in the plots indicate that the tools failed to generate datasets for the corresponding combination of $\epsilon$ and data size. Note that since Gretel Synthetics does not provide functionality for inputting target $\epsilon$ values, we do not have datasets for desirable $\epsilon$ settings and therefore omit the Gretel Synthetics results from Figure 23. Also, in this evaluation, Gretel and Smartnoise MWEM fail to generate any datasets for the Parkinson dataset, thus the two tools’ evaluation analysis is omitted from the contour plots.

We can observe from the contour plots that, for the evaluation conducted on Health Survey, Smartnoise PATECTGAN and DPCTGAN bring similar utility, yet PATECTGAN manifests a slight trend that utility decrease with the data size, which is opposite to our anticipation. Smartnoise MWEM can not generate valid data for a combination of data size over 5000 and higher $\epsilon$, as shown in Figure 23c and 23l. As for the evaluation on Parkinson data, Smartnoise DPCTGAN shows less competence than PATECTGAN in that DPCTGAN produces no useful synthetic data under many $\epsilon$ and data size combinations. These results indicate that Smartnoise PATECTGAN can provide more stable data synthesis for both categorical and continuous data than DPCTGAN, which performs better on categorical data, and MWEM, which is only available for categorical data.

Figure 24 details how the tools’ data utility compares under different data sizes and $\epsilon$. Note that Gretel Synthetics does not explicitly support setting $\epsilon$ values, and we only show its results for the smallest $\epsilon$ value obtained (see Table 6), which is still relatively high compared to target $\epsilon$ values (see Table 4).

From Figure 24, we can observe that Smartnoise PATECTGAN provides stable utility for \texttt{AVERAGE} queries in both Health Survey and Parkinson data, with RMSPE between 60 and 80
generally. While for \textsc{Histogram} queries, \textsc{PATECTGAN} experiences a much more utility decrease in \textit{Health Survey} than \textit{Parkinson} data, implying \textsc{PATECTGAN}’s better stability in continuous data. In comparison, \textsc{DPCTGAN} brings similar or even better utility than \textsc{PATECTGAN}, \textit{e.g.}, with RMSPE of nearly 20 less than \textsc{PATECTGAN} in Figure 24c for the evaluation on \textit{Health Survey} data. While for \textit{Parkinson} data, \textsc{DPCTGAN} generally bears RMSPE of 20 higher than that of \textsc{PATECTGAN} or even fails to generate data, which indicates \textsc{DPCTGAN}’s better performance in
categorical data than continuous. As for Smartnoise MWEM, it only produces synthetic data for limited settings of $\epsilon$ and data size for the Health Survey data, without significant advantage over DPCTGAN or PATECTGAN. Though Gretel also works on Health Survey data well with relatively low RMSPE, which means high utility, the corresponding $\epsilon$ value is very high (more than 25.0), and the resulting privacy protection can be insignificant.

Figure 24: The evaluation of synthetic data tools on data utility between DP and NP statistical query results for different $\epsilon$ (Table 4) and data size (Table 3), measured in RMSPE (Defined in Section 4.2).
5.3.2 Machine learning utility

This subsection evaluates the considered data synthesizers on machine learning (ML) task by comparing the ML models trained on synthetic data with those on the original version. This evaluation aims to look into how differential privacy measures impact the data synthesis tools regarding whether their generated data still hold utility in machine learning tasks. To this end, for each data synthesizer, we train a linear regression model using 80% of their synthetic data and then test the trained model using the corresponding remaining 20% of the original data. Finally, we analyze different synthesizers’ performances in retaining utility in machine learning tasks based on the test results.

In this evaluation, we anticipate that synthetic data sets generated with higher $\epsilon$ values and larger data size will provide better utility since higher $\epsilon$ values indicate adding less noise to the data, and larger data size implies that individual data items contribute less to the model training [26]. As we detail below, the evaluation results show that higher data size tends to provide better utility on Parkinson data, while no similar trend on Health Survey. We also see no trend regarding $\epsilon$ values on data sets. Generally, Smartnoise PATECTGAN synthesizer performs the best for both datasets. All synthesizers from both Smartnoise and Gretal perform significantly worse on Parkinson than on Health Survey, which might imply that both Smartnoise and Gretal perform better with categorical data for ML tasks.

Figure 25 shows contour plots for each tool’s performance regarding different $\epsilon$ and data size. Note that Gretel Synthetics could not generate data for targeting $\epsilon$; thus, this contour plot does not include results on Gretal. Also, Gretel and Smartnoise MWEM fail to generate any dataset for the Parkinson data. From the contour plots, we can observe a trend for Parkinson where larger data size provides better results, which is more evident for Smartnoise PATECTGAN in Figure 25c. A similar trend can also be seen for Parkinson, where Smartnoise DPCTGAN can generate better utility for larger data sizes (Figure 25c). However, no trends are noticeable for any considered synthesizers on Health Survey. Beyond that, Smartnoise MWEM generates no available data for the Health Survey data under the combination of higher $\epsilon$ and higher data size, as shown in Figure 25c, while DPCTGAN does not work for a wide range of settings, as shown in Figure 25e.

Figure 26 shows the line plots on the considered synthesizers’ performances under different settings. The plots demonstrate that those synthesizers perform significantly better on the categorical data of Health Survey than on the continuous data of Parkinson from the RMSPE values, and that the Smartnoise PATECTGAN outperforms the others on both Health Survey and Parkinson data. In comparison, Smartnoise MWEM merely performs well on Health Survey data with RMSPE generally less than 80 for small $\epsilon$ as depicted in Figure 26a and 26b, and Smartnoise DPCTGAN only shows similar performance with PATECTGAN for low $\epsilon$ for both the two data set as shown in Figure 26a and 26c. Note that Gretel Synhtetics also perform well on Health Survey; However, as previously mentioned, Gretel Synhtetics uses a significantly larger privacy budget $\epsilon$ than the other synthesizers.

5.3.3 Run-time overhead

The process of synthetic data generation (SDG) takes additional time to integrate differential privacy measures and produce new data set. In this evaluation, we study how much time the data generation process takes under different settings and how it compares between the considered data synthesis tools to assess whether the considered tools can perform the data synthesis efficiently
Figure 25: Contour plots for the evaluation of synthetic data tools on machine learning utility for different $\epsilon$ (Table 4) and data size (Table 3). RMSPE is defined in Section 4.2.

Figure 26: The evaluation results of synthetic data tools on machine learning utility for the synthetic data tools and data size (Table 3) for a four given privacy budgets. RMSPE is defined in Section 4.2 from a perspective of running time consumption.

For this evaluation, we anticipate that generating data sets with larger size takes additional time, since more data has to be processed intuitively. As detailed below, the experiment results
show that Smartnoise PATECTGAN takes less running time than Smartnoise DPCTGAN and MWEM, and that there exists the trend that larger datasets take longer time to generate than smaller ones, which was inline with our anticipations.

Figure 27: Contour plots for the evaluation results of synthetic data tools on run-time overhead under different $\epsilon$ (Table 4) and data size (Table 3).

The contour plots in Figure 27 show each tool’s run-time regarding different $\epsilon$ and data size. Note that Gretel Synthetics does not provide functionality for setting the $\epsilon$ in data generation, and thus its result is omitted in the contour plots. Since Gretel Synthetics and Smartnoise MWEM fails to generate any datasets for Parkinson, these cases are also omitted from Figure 27.

From the contour plots, we can observe a trend, though not decisive, that larger privacy budgets ($\epsilon$ values) and larger data size increase the time it takes to generate the datasets, as shown in Figure 27a, 27b, and 27d. It also shows that Smartnoise PATECTGAN performs well stably on both the Health Survey and Parkinson data, with similar RMSPE of around 0-30, while DPCTGAN experiences malfunction under some combination of data size and $\epsilon$ on the Parkinson data, indicating DPCTGAN’s less efficiency on continuous data regarding execution speed. In comparison, Smartnoise only works on the categorical data of Health Survey data under limit settings of data size and $\epsilon$.

Figure 28 shows further details on the evaluation, where we can observe that Smartnoise MWEM and Gretel perform significantly worse than Smartnoise PATECTGAN and DPCTGAN, with additional 100-700 seconds needed than the latter two tools in data generation. For Smartnoise PATECTGAN and DPCTGAN, the former one demonstrates more stable performance on both categorical and continuous data and less running time consumed for higher data size on the Health Survey data. Overall, Smartnoise PATECTGAN manifests its advantage over the other synthesizers in this evaluation.
5.3.4 Memory overhead

This subsection evaluates the memory consumption of the data synthetic tools when running the differential private data generation under different settings, and we also study how the different tools compare regarding memory consumption. In this evaluation, we anticipate that data synthesis for larger data will increase memory consumption since more memory is needed to store the generated data. The evaluation results demonstrate that, as detailed below, Smartnoise DPCTGAN and Smartnoise PATECTGAN behave as we anticipated on Parkinson data, so does Smartnoise DPCTGAN on Health Survey, while not for Smartnoise MWEM and PATECTGAN on Health Survey. Generally, Smartnoise PATECTGAN performs best on both data sets, with lower memory consumption than other synthesizers.

Figure 29 shows contour plots for each tool’s memory consumption regarding different $\epsilon$ and data sizes. Note that Gretel Synthetics does not provide functionality for settings $\epsilon$; therefore, it is omitted from the contour plots. Besides, since Gretel Synthetics and Smartnoise MWEM fail to generate any datasets for Parkinson, no results are included for this case in Figure 29.

We can observe from the contour plots that all synthesizers tend to consume less memory for lower $\epsilon$, except for Smartnoise MWEM on Health Survey. An even more apparent trend is that memory consumption increases with data size, as shown in Figure 29b, 29c, which is in line with our anticipation. However, irregularities exist diverging from this trend, and some settings stop the synthesizers from generating data, shown as the white areas in Figure 29c for Smartnoise MWEM and Figure 29e for Smartnoise DPCTGAN. When summarizing the performances of all tools, it shows that Smartnoise PATECTGAN provides the best results with the lowest memory consumption generally lower than 230 MB for Health Survey and lower than 220 MB for Parkinson data.
Figure 29: Contour plots for the evaluation results of synthetic data tools on memory consumption under different $\epsilon$ (Table 4) and data size (Table 3). Memory is measured in mega bytes (MB).

Figure 30: The evaluation of synthetic data tools on memory consumption under different $\epsilon$ (Table 4) and data size (Table 3). Memory is measured in mega bytes (MB).

Figure 30 provides more details on the evaluation results, where we can observe that less memory usage is incurred for Smartnoise MWEM and Smartnoise PATECTGAN for Health survey on Parkinson compared with Smartnoise DPCTGAN. Though Gretel uses much larger $\epsilon$ when gener-
ating data for *Health Survey*, it still performs worse than the Smartnoise synthesizers and consumes more memory. We can also observe that while MWEM and PATECTGAN perform more steadily under different data sizes and $\epsilon$, the memory cost by Smartnoise DPCTGAN tends to increase as data size grows for both of the two datasets.

### 5.4 Discussions on the results

The evaluation demonstrates that the studied differential privacy tools perform differently for different tasks and under different configurations. While the results reveal some patterns that we can leverage to configure some tools optimally, irregularities exist, making the configuring complex and inducing difficulties in comparing and selecting tools accordingly.

| tool name                      | patterns revealed                                                                 |
|--------------------------------|-----------------------------------------------------------------------------------|
| **machine learning**           |                                                                                  |
| Diffprivlib                    | · higher utility and more run-time overhead for higher $\epsilon$ and larger data size on categorical data |
| Pytorch Opacus                 | · higher utility for larger $\epsilon$ on categorical and continuous data;       |
|                                | · higher utility for larger data size on categorical data                          |
| TensorFlow Privacy             | · higher utility for larger $\epsilon$ and data size on categorical and continuous data; |
|                                | · higher memory overhead for larger data size on categorical and continuous data     |
| Smartnoise PATECTGAN           | · higher utility for larger data size on continuous data                            |
| Smartnoise DPCTGAN             | · higher utility for larger data size on continuous data                            |
| Smartnoise MWEM                | none                                                                              |
| Gretel                         | none                                                                              |
| **statistical query**          |                                                                                  |
| Google Differential Privacy    | · higher utility for larger $\epsilon$ and larger data size on categorical and continuous data; |
|                                | · higher run-time overhead for lower data size on categorical and continuous data   |
| OpenDP SmartNoise              | · higher run-time overhead for higher data size on categorical and continuous data  |
| Smartnoise PATECTGAN           | · higher utility for smaller data size on categorical data                          |
| Smartnoise DPCTGAN             | none                                                                              |
| Smartnoise MWEM                | none                                                                              |
| Gretel                         | none                                                                              |
| **data synthesis**             |                                                                                  |
| Smartnoise PATECTGAN           | · higher run-time overhead for larger $\epsilon$ on categorical data;            |
|                                | · higher run-time overhead for larger data size on continuous data;               |
|                                | · higher memory for larger data size on continuous data                            |
| Smartnoise DPCTGAN             | · higher run-time overhead for higher $\epsilon$ and larger data size on categorical data |
| Smartnoise MWEM                | none                                                                              |
| Gretel                         | none                                                                              |

This subsection summarizes available patterns of the tools under given tasks, which offers the basis for optimally configuring the tools, as shown in Table 7. With the patterns at hand, tool users can reasonably estimate where the best trade-off between a good performance for a task (like to train a model) and a sound privacy protection appears within the configuration range. Note that not all tools have explicit performance patterns under specific settings, where further analysis is needed to associate user’s consideration and choice w.r.t. tool performance. Therefore, we provide a comprehensive comparison between all the tools to guide tool-selecting depending on different criteria. Table 8 details how different tools compare with each other under four criteria, the definitions of which are shown below. Note that the different levels of gray/blue in the table cells indicate the performance ranking of tools under specific criterion, where darker ones mean worse performance and lighter ones mean the opposite. The gray box means the relative performance (*e.g.*, privacy-preserving analysis vs. non-privacy one), and the blue box means absolute performance (*e.g.*, memory usage for data synthesis). Pure black represents failure under the considered criteria.
. **Potential**: the performance of a tool on a data set with the largest data size and highest $\epsilon$ shown in Table 4 and 3. This criterion shows how the tool performs w.r.t. utility/run-time/memory given the most generous conditions of the data and privacy measures.

. **Resilience**: the performance of a tool on a data set with the smallest data size and lowest $\epsilon$. It shows how the tool works w.r.t. utility/run-time/memory-usage given the most restricting conditions of the data and privacy measures.

. **Configuration flexibility**: the level to which a tool can be flexibly configured according to Table 4 and 3. This criterion manifests whether a tool can be configured as the user’s need.

. **Accessibility**: the level to which the utility/run-time/memory-usage of a privacy-preserving task (e.g., to train a differentially private model) can be held by the tool compared with the non-privacy-protection task, with in the range of settings shown in Table 4 and 3.

With the defined criteria and the performance measurement under each criterion, we enable the users to select differential privacy tools smartly according to their own need and priority, e.g., the highest utility, the lightest computing resource requirement, the least running time. Generally, from the performance comparison in Table 8 we recommend TensorFlow Privacy and Pytorch Opacus equivalently for machine learning tasks, and Google Differential privacy for statistical queries. For data synthesis, our advisory choice is Smartnoise PATECTGAN for both machine learning and statistical query.

Table 8: Tool performance comparison on tasks of machine learning and statistical query under different criterion.

|                   | categorical data | continuous data |
|-------------------|------------------|-----------------|
|                   | potential in utility | potential in run-time | potential in memory | resilience in utility | resilience in run-time | resilience in memory | accessibility in utility | accessibility in run-time | accessibility in memory | configuration flexibility |
| Machine learning  |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Diffprivlib       |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Pytorch Opacus    |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| TensorFlow Privacy|                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Smartnoise PATECTGAN |              |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Smartnoise DPCTGAN|                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Smartnoise MWEM   |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Gretel            | NA               | NA               | NA               | NA                   | NA                   | NA                   | NA                   | NA                   | NA                   | NA                     |
| Statistical query |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Google Differential Privacy |          |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| OpenDP SmartNoise |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Smartnoise PATECTGAN |              |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Smartnoise DPCTGAN|                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Smartnoise MWEM   |                  |                  |                  |                      |                      |                      |                      |                      |                      |                       |
| Gretel            | NA               | NA               | NA               | NA                   | NA                   | NA                   | NA                   | NA                   | NA                   | NA                     |

6 Conclusions

This work builds an evaluation framework and evaluates the state-of-the-art open source differential privacy (DP) tools. For the evaluation, we define criteria to quantify how different DP tools perform and how they can be optimally configured. Specifically, we evaluate and measure the impact of DP
on different functionalities that the studied tools provide, i.e., statistical queries, machine learning, and synthetic data generation, and how the tools compare to each other. We use two data sources of different types to obtain a nuanced picture of how well the considered tools perform when DP is applied. The evaluation results demonstrate (i) how much DP tools impact data utility and system overhead in DP applications and (ii) if DP service practitioners can apply these tools with a range of privacy settings and which tools are preferable.

With the insights provided by the accumulated results, we offer intuitions of what level of privacy and utility can be anticipated in developer use-case, and guidelines in selecting DP tools according to user’s need. Thus, we provide privacy-service practitioners with easy access to the selection, usage, and configuration of differential privacy tools and make the conceptional differential privacy more visible to its application side, contributing to protecting the privacy of individuals densely connected by data and information.

Acknowledgement

The work of Shiliang Zhang was supported by the project ‘Privacy-Protected Machine Learning for Transport Systems’ of Area of Advance Transport and Chalmers AI Research Centre (CHAIR). The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at C3SE partially funded by the Swedish Research Council through grant agreement no. 2018-05973. We thank Elad Schiller and Magnus Almgren for important discussions, suggestions, and helping to improve the presentation.

References

[1] C. Dwork, “Differential privacy: A survey of results,” in Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi’an, China, April 25-29, 2008. Proceedings, ser. Lecture Notes in Computer Science, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds., vol. 4978. Springer, 2008, pp. 1–19.

[2] J. Scherer and G. Kiparski, “Buchbesprechungen. feiler, lukas / forgó, nikolaus / weigl, michaela: The eu general data protection regulation (gdpr): A commentary,” Comput. und Recht, vol. 34, no. 6, pp. 69–70, 2018.

[3] S. Mbonihankuye, A. Nkunzimana, and A. Ndagijimana, “Healthcare data security technology: HIPAA compliance,” Wirel. Commun. Mob. Comput., vol. 2019, pp. 1 927 495:1–1 927 495:7, 2019.

[4] M. U. Hassan, M. H. Rehmani, and J. Chen, “Differential privacy techniques for cyber physical systems: A survey,” IEEE Commun. Surv. Tutorials, vol. 22, no. 1, pp. 746–789, 2020.

[5] K. L. van der Veen, R. Seggers, P. Bloem, and G. Patrini, “Three tools for practical differential privacy,” CoRR, vol. abs/1812.02890, 2018.

[6] W. Zhang, “Privacy-preserving statistical tools: Differential privacy and beyond,” Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, 2021.
[7] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson, “Differentially private SQL with bounded user contribution,” CoRR, vol. abs/1909.01917, 2019.

[8] N. Holohan, S. Braghin, P. M. Aonghusa, and K. Levacher, “Diffprivlib: The IBM differential privacy library,” CoRR, vol. abs/1907.02444, 2019.

[9] L. Rosenblatt, X. Liu, S. Pouyanfar, E. de Leon, A. Desai, and J. Allen, “Differentially private synthetic data: Applied evaluations and enhancements,” CoRR, vol. abs/2011.05537, 2020.

[10] N. Johnson, J. P. Near, and D. Song, “Towards practical differential privacy for sql queries,” Proc. VLDB Endow., vol. 11, no. 5, p. 526–539, Jan. 2018.

[11] F. McSherry, “Privacy integrated queries: an extensible platform for privacy-preserving data analysis,” Commun. ACM, vol. 53, no. 9, pp. 89–97, 2010.

[12] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. ACM, 2016, pp. 308–318.

[13] Z. Bu, S. Gopi, J. Kulkarni, Y. T. Lee, J. H. Shen, and U. Tantipongpipat, “Fast and memory efficient differentially private-sgd via JL projections,” CoRR, vol. abs/2102.03013, 2021.

[14] P. Subramani, N. Vadivelu, and G. Kamath, “Enabling fast differentially private SGD via just-in-time compilation and vectorization,” CoRR, vol. abs/2010.09063, 2020.

[15] A. Kopp, “Microsoft smartnoise differential privacy machine learning case studies,” Microsoft Azure white papers, 2021.

[16] B. Jayaraman and D. Evans, “Evaluating differentially private machine learning in practice,” in 28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 1895–1912.

[17] I. Mironov, “Rényi differential privacy,” in 30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017. IEEE Computer Society, 2017, pp. 263–275.

[18] F. Tramèr and D. Boneh, “Differentially private learning needs better features (or much more data),” CoRR, vol. abs/2011.11660, 2020.

[19] D. Knroors, “Utility of differentially private synthetic data generation for high-dimensional databases,” 2018.

[20] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, “Calibrating noise to sensitivity in private data analysis,” J. Priv. Confidentiality, vol. 7, no. 3, pp. 17–51, 2016.

[21] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth, “Differential privacy: An economic method for choosing epsilon,” in IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014. IEEE Computer Society, 2014, pp. 398–410.
[22] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau, “EKE-TELO: A framework for defining differentially-private computations,” in Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018. ACM, 2018, pp. 115–130.

[23] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay, “Architecting a differentially private SQL engine,” in 9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings. www.cidrdb.org, 2019.

[24] A. Haeberlen, B. C. Pierce, and A. Narayan, “Differential privacy under fire,” in 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings. USENIX Association, 2011.

[25] D. Zhang and D. Kifer, “Lightdp: towards automating differential privacy proofs,” in Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. ACM, 2017, pp. 888–901.

[26] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson, “Differentially private SQL with bounded user contribution,” Proc. Priv. Enhancing Technol., vol. 2020, no. 2, pp. 230–250, 2020.

[27] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, “Generative adversarial networks,” Commun. ACM, vol. 63, no. 11, pp. 139–144, 2020.

[28] N. M. Johnson, J. P. Near, J. M. Hellerstein, and D. Song, “Chorus: Differential privacy via query rewriting,” CoRR, vol. abs/1809.07750, 2018.

[29] ——, “Chorus: a programming framework for building scalable differential privacy mechanisms,” in IEEE European Symposium on Security and Privacy, EuroS&F 2020, Genoa, Italy, September 7-11, 2020. IEEE, 2020, pp. 535–551.

[30] S. Rezaei, Z. Shafiq, and X. Liu, “Accuracy-privacy trade-off in deep ensemble,” CoRR, vol. abs/2105.05381, 2021.

[31] M. Platzer and T. Reutterer, “Holdout-based fidelity and privacy assessment of mixed-type synthetic data,” CoRR, vol. abs/2104.00635, 2021.

[32] S. K. Long, “Massachusetts health reform survey, 2018,” Inter-university Consortium for Political and Social Research, 2019.

[33] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig, “Accurate telemonitoring of parkinson’s disease progression by noninvasive speech tests,” IEEE Trans. Biomed. Eng., vol. 57, no. 4, pp. 884–893, 2010.

[34] J. Hayes, L. Melis, G. Danezis, and E. D. Cristofaro, “LOGAN: membership inference attacks against generative models,” Proc. Priv. Enhancing Technol., vol. 2019, no. 1, pp. 133–152, 2019.
[35] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, “Deep models under the GAN: information leakage from collaborative deep learning,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. ACM, 2017, pp. 603–618.

[36] M. Gaboardi, H. Lim, R. M. Rogers, and S. P. Vadhan, “Differentially private chi-squared hypothesis testing: Goodness of fit and independence testing,” in Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 2111–2120.