Differential Privacy: an Extensive Evaluation of Open-Source Tools for eHealth Applications

Master's thesis in Computer Science

Anton Hägermalm
Sanjin Slavnic

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Differential Privacy: an Extensive Evaluation of Open-Source Tools for eHealth Applications

Anton Hägermalm
Sanjin Slavnic

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Differential Privacy: an Extensive Evaluation of Open-Source Tools for eHealth Applications
Anton Hägermalm, Sanjin Slavnic

Supervisor: Shiliang Zhang, Department of Computer Science and Engineering and Elad Michael Schiller, Department of Computer Science and Engineering
Examiner: Magnus Almgren, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Differential Privacy: an Extensive Evaluation of Open-Source Tools for eHealth Applications
Anton Hägermalm, Sanjin Slavnic
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Medical data is fundamentally important for improving and generating new insights in healthcare. However, personal health data is especially sensitive and statistics about it can leak and compromise individual integrity. In order to protect the data of individuals, or smaller groups of individuals, formal techniques can be applied with software to modify the data and decrease the risk of unauthorized disclosure. One such formal method that has been receiving increasing attention in the last couple of years is differential privacy (DP).

DP adds carefully calculated noise to a statistical query, during the training of machine learning models, or during synthetic data generation to guarantee the privacy of the individual participants in the dataset. However, there is a gap between fundamental and applied research on the topic of DP. Particularly, the lack of knowledge exists over what software tools can be leveraged in privacy application development, and to what level of performance developers can expect from potential tools.

In this thesis, we review and evaluate a set of DP tools to gain insights into how they perform in practice. For reasonable comparison between the DP tools' performances, we in the evaluation categorize the tools into three domains, namely statistical queries, machine learning, and synthetic data release. Specifically, for statistical queries we look at *Google DP* and *Smartnoise*, for machine learning *Tensorflow Privacy*, *Diffprivlib* and *Opacus*, and for synthetic data release we look at *Smartnoise* and *Gretel*. These considered tools are open-source real-world deployments that are created in collaboration with and vetted by a community of engineers, scientists and experts.

In our evaluation, we measure how the tools affect data analysis, using the metrics of accuracy and overhead, by comparing results of differential private analysis with analysis without privacy protection. The evaluation employs two datasets in the generating of analysis results: *Parkinson Telemonitoring* and *2018 Massachusetts Health Reform Survey*. For the implementation of the evaluation, we develop a framework where the system overhead and loss of accuracy can be quantified for any tool and any dataset, which can be reused for further tests. The full source code for the framework is openly released as: `https://github.com/anthager/dp-evaluation`. This evaluation allows us to look into how these tools perform on real data and how they compare. With the evaluation results, we provide guidance for how these tools can be applied to ultimately help improve privacy for individuals.

Our work provides abundant testing results on the considered tools under varieties

of settings, allowing the view of how the tools trade off privacy and utility under different conditions. We gain in the evaluation some general observations and trends via comparison on the tool performance, e.g., it shows that conducting statistical queries and ML tasks on DP synthetic data has significantly larger impact on data accuracy than using statistical query and ML tools on non-privacy protected data, where DP is integrated in the tools themselves. The evaluation also reveals that it is not trivial to configure the tools, which is highly dependent on both the data, the tool being used, and the use case. The results from our evaluation can serve as guidelines for the optimally configuration of the tools, e.g., how the value of ϵ can be configured for different data sizes, and what data utility can be expected in those circumstances. We detail a summary of our results in Section 5.1.

Keywords: Differential Privacy, Privacy Tools, Statistical Queries, Machine Learning, Synthetic Data, eHealth

Acknowledgements

We would first and foremost like to thank our supervisor Shiliang Zhang for his lasting support and guidance throughout this project, as well for sharing his expertise in machine learning. Further, we want to thank our co-supervisor Elad Schiller for his support and general assistance throughout the thesis. We would also like to thank Mats Pettersson for his valuable time, and giving us the opportunity to collaborate with AstraZeneca. We want to thank Anders Rydén, Henrik Algestam and Kate Campbell for their time and assistance at AstraZeneca. Lastly, we want to thank our families and friends for their continuous support and encouragements throughout this project.

Anton Hägermalm and Sanjin Slavnic, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background and Motivation

Healthcare data is fundamentally important for generating new insights and improving healthcare, and the usage of big data has become one of the core resources of life-science research [1, 2]. Generally, a data collector, a.k.a a curator, collects data and is responsible for releasing the data for analyses. This procedure introduces distinctive challenges concerning privacy-sensitive information leakage, which emerges during the critical process of knowledge extraction from datasets consisting of individual healthcare records [1, 2].

Moreover, since the introduction of data protection legislation, such as the General Data Protection Regulation (GDPR), privacy protection has become a legal requirement rather than merely an ethical expectation. Specifically, healthcare data is defined in GDPR Article 4 [3] as personal data and must thus be protected.

The undesirable risk of privacy cost has motivated diversities of privacy-preserving techniques to be developed and taken into effect. However, many of the traditional methods employed by curators like *k-anonymity*[1] are inherently flawed [5, 6]. A piece of information that by itself is insignificant can, when combined with other pieces of information, either from the same or another dataset, reveal privacy-sensitive information. Moreover, using the concepts of exchangeability and de Finetti's theorem [7], has demonstrated weaknesses in many data sanitization techniques [8].

These flaws were exposed by Dinur and Nissim [9] who showed that it is impossible to publish arbitrary queries on a private statistical database without revealing any private information. Their study showed as an example that the content in an anonymized database can be fully revealed with only a small number of queries.

Differential privacy (DP) on the other hand, offers an alternate and stronger notion of privacy than anonymization and provides a formal privacy guarantee for data. It reinforces a stronger privacy basis for GDPR than existing approaches like anonymization, which is a common method to avoid privacy legislation violation yet demonstrably vulnerable to disclosure attacks. That is why DP is regarded by some investigations [10] as the most viable privacy protection technique for real-time

[1] K-anonymity is a *"hiding in the crowd"* guarantee, i.e., if an individual is part of a group, then any of the records in this group could merely correspond to a single person. [4]

medical devices data.

Nevertheless, a gap exists between fundamental and applied research on the topic of DP. The academic literature around DP is less intuitive, filled with concepts that do not necessarily map well to the vernacular used in industry. There is a need for a bridge between the academic DP landscape and the practical application of DP in technology. Particularly, knowledge is absent over what software tools can be leveraged in privacy application development with DP principles, and to what level of performance developers can expect from the existing tools.

This disadvantage hinders the further prosperity of DP in real-world applications and consequently induces practical obstacles to the protection of individual privacy from the perspective of software communities.

1.2 Problem Description

Recently, several open-source tools were proposed for applying differential privacy (DP) in statistical queries, machine learning (ML), and synthetic data generation. Those tools bridge DP theories with DP service developments by avoiding application development from scratch, facilitating developer communities with the ability to achieve expected functionalities under a variety of DP configurations.

However, there is no comparative evaluation of these tools that also provide generic guideline information. E.g., how they differ and what privacy settings should be used to achieve DP in practice. We, therefore, perform an extensive evaluation of the most relevant and interesting tools, with an emphasis on the evaluation of the usage of eHealth datasets that consist of individual privacy-sensitive information.

1.3 Purpose

The purpose of this work is to evaluate open-source tools for DP, with an emphasis on eHealth datasets. We define the metrics of data utility and system overhead for evaluation of different tools (Section 4.1) and evaluate three domains of DP tools (Table 1.1) categorized by the difference in DP services. We aim to provide deeper insight into existing open-source DP tools, specifically their performance in DP functionalities, and to find suitable configurations for the tools, to provide recommendations for how these tools can be used to their full extent.

To this end, we implement a framework where we compare the DP tools in a set of real-world scenarios. We want to study the performance that the tools can achieve by running the functionalities that these tools provide. We aim to provide guidelines for what tool and what configuration yields the best trade-offs between data utility and privacy for the given datasets (Table 4.1), and their respective use-case. Moreover, this framework can be leveraged outside of our work to evaluate new tools and

also help other developers pick the best privacy tool for their specific datasets and requirements.

1.4 Research Questions

We notice that it is not intuitive how existing differential privacy (DP) tools impact data analysis in applications, nor do guidelines exist on how to select tools and what can be expected from them during privacy-preserving application development. The situation implies a weakness of the bridge between DP theory and its practical implementation.

This work aims to enhance this bridge by providing an extensive evaluation of industry-relevant open-source DP tools with different criteria and providing a framework that can be reused for evaluations in the future. To measure the performance of these tools, we consider two central factors, data utility and system overhead (defined in Section 4.1), and put forward two primary questions for this evaluation:

- ***How much do DP tools impact data utility and system overhead in DP applications?***
- ***Can DP service practitioners apply these tools, and how can they configure them?***

Our primary motivation for these questions is to find out if the tools provide good enough data utility and system overhead in practice, with a focus on life-science institutions and companies. Furthermore, we aim to find suitable configurations for the tools to provide general guidelines for how they can be used in application development.

1.5 Stakeholder

For our stakeholder *AstraZeneca* (AZ), which is one of the world's largest pharmaceutical companies, privacy is of utmost importance. They handle a wide range of patient data, such as records from clinical studies, medical trials, and data from wearable medical devices. AZ accumulates data on a large scale and investigates if they can improve the privacy in their data pipelines and data publishing.

We believe that differential privacy can be a good fit for this job. AZ desires specifically to consider synthetic data generation in its process of data collection from an electronic medication device.

1.6 Use-Cases

Currently, AZ exports raw usage data from an electronic medication device, which is anonymized and saved to data warehouses. The anonymized data is thereafter

imported to their analysis tool, where they extract statistics through a set of queries. The anonymization part can be regarded as an essential application and is of special interest for AZ. The use-cases for this evaluation are thus based on AZ's use-case, which is to obtain knowledge from the usage of their electronic medication device.

In our work, we evaluate DP tools for statistical queries, machine learning (ML) tasks, and synthetic data generation. We select open datasets in the field of eHealth (Table 4.1) that fit into a similar category as AZ's electronic medication device. We implement applications to ultimately obtain knowledge from the data, by conducting statistical queries and performing ML tasks. While the work is centered around eHealth due to our collaboration with AZ, the insights and results of our work can be generalized to varieties of fields where privacy preservation is essential.

1.7 Privacy Tools

Privacy tool	**Domain**
OpenDP SmartNoise[2]	Statistical Query
Google Differential Privacy[3]	Statistical Query
TensorFlow Privacy[4]	Machine Learning
Pytorch Opacus[5]	Machine Learning
IBM Diffprivlib[6]	Machine Learning
OpenDP SmartNoise[7]	Synthetic Data
Gretel Synthetics[8]	Synthetic Data

Table 1.1: List of differential privacy tools with publicly available open-source repositories.

In this evaluation, we considered tools for *statistical queries*, tools for *machine learning* and tools for *synthetic data generation*. We selected tools following two main criteria that (1) what tools have the most active community of privacy engineers and (2) what tools amongst those have sufficient theoretical backup and documentation. Our criteria disqualified deprecated open-source libraries and academic research tools. Under such criteria, Table 1.1 provides a list of real-world deployments of open-source tools that were selected for this evaluation.

[2] https://github.com/opendifferentialprivacy/smartnoise-sdk
[3] https://github.com/google/differential-privacy
[4] https://github.com/tensorflow/privacy
[5] https://github.com/pytorch/opacus
[6] https://github.com/IBM/differential-privacy-library
[7] https://github.com/opendifferentialprivacy/smartnoise-sdk
[8] https://github.com/gretelai/gretel-synthetics

The tools were considered since they have support from a comparably large community and have gained wide acceptance in the open-source community, as well as possessing theoretical backup and sufficient documentation. They have therefore potential to be adopted in privacy-preserving application developments and a possibility to influence a broader scope of privacy services.

1.8 Related Work

In this section, we review related works regarding the performance evaluation of privacy tools that apply DP. While there have been numerous studies around differential privacy [11, 12, 13, 14, 15, 16, 17], none, to the best of our knowledge, provides a comparative and in-depth, study between the various open-source tools that can be applied in practice.

We aim to introduce previous and relevant research and compare techniques and measures for how to evaluate privacy tools in regards to their performance. Since DP's influence differs due to differences in services, the review in this section is categorized into three functionality domains: statistical queries, machine learning, and synthetic data as described in the following.

1.8.1 Statistical Queries

One differentially private (DP) query engine that has been used in industry is Flex [18]. The open-source library for Flex is deprecated [19] and was not considered in our work. However, their evaluation remains relevant as a comparison for this evaluation.

Johnson et al. [18] 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 significantly. 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 industry. However, this evaluation merely benchmarks one ϵ 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 (NP) queries. 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. Such evaluating procedure induces the risk that the actual overhead might be larger than presented in practice. Beyond that, there is a lack of illustration that whether overhead changes when settings differ in, e.g., dataset sizes, privacy configurations or queries, etc.

The industry also has observed the DP tools of Smartnoise and Google DP integrated in privacy services. Smartnoise is a result of years of cumulative experience in building and deploying privacy tools for research and has recently become a tool

in Microsoft's privacy ecosystem [20]. 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[9] [22]. Their evaluation performed 1,000,000 runs with various aggregate functions, with a fixed ϵ 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 users can have multiple contributions, i.e. the amount of times an individual can be associated in records or queries containing joins. It is an important observation because it can lead to query results that are not DP. Furthermore, because Flex or PinQ make the assumptions that the underlying database is associated with at most one database record, Google DP evaluation of aggregation with joins could not compare these results to the other engines.

Ala Hadi [17] also compares open source tools for DP statistical queries but collects less data and is less in-depth. While he does look at Google DP, he does not analyze Smartnoise and instead analyzes Chorus which is since deprecated, which makes it less relevant.

1.8.2 Machine Learning

Differentially private (DP) machine learning (ML) has gained attention in industry, at companies such as Google, Facebook, and IBM. Interesting studies have been done on the performances of DP stochastic gradient descent (SGD) [23], which is a dominant algorithm for private training of large scale neural networks. Yet DP-SGD can increase the training time significantly compared to non-privacy protected (NP) SGD, i.e., it has a significant run-time overhead [24]. In a recent study, Subramani et al. managed to reduce the run-time overhead which is a common problem when executing DP-SGD [25]. They implemented the functionality in the open-source library of Tensorflow Privacy by exploiting language primitives [24]. Companies such as Microsoft have also shown support in the DP-ML field by implementing Opacus and demonstrating the impact of ϵ and dataset size on DP-ML [4]. However, no comparison to other tools was taken into consideration by previous efforts.

In another study, Tramér et al. managed to provide an approach to improve performances of DP models using primarily Tensorflow and parts of the Opacus library [26]. They point out that prior works have underestimated privacy-utility guarantees and demonstrate that strong privacy may come at only a minor cost of accuracy by tailoring the training to the data. However, this work has not yet been generalized as a tool that can be easily leveraged in the developer community.

[9]The research project Privacy Integrated Queries (PinQ) is a programming language and execution platform in which all expressible programs satisfy DP. [21]

Beyond improvements regarding performance, there is also work recently to improve the privacy guarantees of DP-ML [14], which evaluates different means of DP with different privacy budgets. This work focuses on gradient perturbation mechanisms like DP-SGD and uses Tensorflow Privacy to test Rényi Differential Privacy among other DP approaches [27], and shows how the trade-off varies between utility and privacy under different settings. It demonstrates that the privacy guarantees of DP in practical machine learning implementations may provide unacceptable utility-privacy trade-offs. This work aims to find ϵ values that achieve a balance between utility and privacy for different DP approaches, rather than for different DP tools.

1.8.3 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. Its theoretical concepts have been adopted into practice and improved by new emerging techniques, which in turn 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 [28]. 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-adversary-networks (GANs) for data synthesis (DPGAN, PATE-GAN, DP-CTGAN, and PATE-CTGAN), measuring 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 how to select data synthesis approaches in programming. Their effort has become a sub-module in Smartnoise that we refer to as *Smartnoise Synthetics* in this thesis.

Though comparative results are provided, Rosenblatt's study does not evaluate the performance in comparison with other DP tools, nor does it take into account the run-time and memory overhead, which is critical to show whether a DP service runs effectively. Additionally, 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 real data. [10] However, they do not explicitly explore how statistical query tools perform on synthetic data, which is an important aspect of data-related services.

In a recent master thesis, Knoors et al. evaluate and compare different SDG techniques [15], where they compare the synthetic data obtained from the different techniques by varying ϵ and dataset size. In their evaluation, they perform one classification task, specifically TSTR, and benchmark 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

[10] Specifically, train-synthetic test-real (TSTR) that they compare with train-real test-real (TRTR).

developer's perspective.

Finally, we note that the scope of this report does not consider other approaches for privacy protection, e.g., secret-sharing [29, 30].

1.9 Our Contribution

We study a critical aspect of data systems, which is the protection of privacy-sensitive information. Our study reviews, evaluates, and provides preliminary guidelines for the selection of open-source tools for differential privacy (DP). The area of privacy protection has a well-recorded history of failed solutions. To the best of our knowledge, DP is the only framework to offer qualitative guarantees to the protection of privacy-sensitive information. The implementation of tools for providing solutions that are differentially private has its own set of traps and pitfalls since it is a non-trivial effort to assure that all computer system aspects needed for differential privacy 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. As far as we know, offer the most comprehensive study that compares the performance of these tools from the application utilization and system perspectives.

Our work aims to help system developers strategically choose privacy tools in an informed way. To the best of our knowledge, we are the first to comprehensively compare the most relevant open-source tools to each other, and study their trade-offs within the application's context. Specifically, our study facilitates the work of developers in the area (and not limited to) of life-sciences, when they need to select privacy-preservation tools. We do that via the provision of metrics that we have defined, to concretely see how the studied tools affect data-related analysis methods, such as statistical queries and machine learning tasks.

To this end, we develop an experimental 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 dominant operating systems of Windows, Linux, and macOS, and offers thus the flexibility to programmers for software re-use purposes with our open-source repository. [11] The designed framework uses the defined criteria that allow us to quantify the utility loss and system overhead of analysis, compared to a non-privacy protected benchmark. Using the devised framework, we evaluate the most relevant open-source tools, based on thorough tool review and compare their performance on differentially private data analysis. We provide insights, through the evaluation results, 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. In our experimental evaluation we obtained and shed some light on interesting findings, where further research is welcomed to im-

[11] `https://github.com/anthager/dp-evaluation`

prove upon our results.

Generally, our evaluation shows that conducting statistical queries and ML tasks on DP synthetic data has a significantly larger impact on data utility than using statistical query and ML tools on non-privacy protected data, where DP is integrated into the tools themselves. And when it comes to configuring the tools, our results show that this is not trivial since it is highly dependent on both the data, the tool being used, and the use case. The results from our evaluation can serve as guidelines for how coefficients like the value of ϵ can be configured for different data sizes, and what data utility can be expected in those circumstances.

Specifically, for developers looking at accumulating general statistics about categorical or continuous datasets stored in a Postgres database engine, with a margin of error from about 0.1% to 2% for simple queries (`SUM`, `COUNT`, `AVG`), with $\epsilon \gtrapprox 1.5$, should consider Google DP. For integration with other database engines, given the same set of queries and ϵ values, Smartnoise provides an accuracy of about 0.5% to 5%. Smartnoise also offer better accuracy for `HISTOGRAM` queries, below about 10% compared to Google DP's about 15%, on *Health Survey*.

For developers looking at building DP machine learning models, both Opacus and Tensorflow Privacy (TFP) show promising results, obtaining data utility below around 6% error for $\epsilon \geq 1.0$ on *Parkinson* and *Health Survey*. Nevertheless, TFP manages to obtain around two times better data utility than Opacus, given maximum data size and $\epsilon = 3.0$. Diffprivlib on the other hand outperforms both tools on *Health Survey*, given maximum data size and $\epsilon = 3.0$. It should however be noted that Diffprivlib did not manage to generate any useful results on continuous data, and is also limited to linear regression and logistic regression models, while TFP and Opacus offer building custom neural networks, allowing developers to build complex models for a variety of problems.

For developers looking at DP synthetic data generation, as mentioned, our results indicate that, given our data sources, these tools do not provide nearly as good accuracy as the alternatives, which integrate DP into statistical queries and ML tasks themselves. However, the potential advantage of future progress in the field, allowing a non-restricted number of queries on synthetic datasets and providing DP datasets for ML tasks, builds a strong case for continuous research and development in the field. It should also be noted that this is an emerging field, and the tools we used for this evaluation are only months old.

As a summary, we anticipate that this work can bring a landscape of the pros and cons of existing open-source privacy tools, and a piece of intuitive knowledge to practitioners on how to leverage the tools in their privacy-preserving service development, and ultimately narrow the gap between theoretical and applied research on DP.

2
Background

2.1 Definition of Privacy

The perception of privacy originates from Dalenius, 1977 [31, 32], who introduced the notion of semantically secure statistical datasets, which is inspired by the concept of probabilistic encryption. It states that an adversary should not be able to learn anything about the plain text, from the encrypted text and the cipher, that could not be learned without it. Similarly, semantic privacy describes a disclosure in the following way:

> *Given an object O from dataset D, assume a characteristic value C of O. If a statistics release S from a survey makes it possible to determine the value of C, more accurately than it is possible without access to S, a disclosure has taken place.* [31, 32]

Informally, one should not be able to learn anything about an individual, with access to a statistical dataset, that could not be learned without access. Nonetheless, it was later proved by Dwork that data cannot be fully anonymized and remain useful [33, 34].

An alternative definition of privacy was instead proposed by Dwork. Rather than not allowing anything to be learned about an object O, e.g., an individual, from the statistics S, the new definition states that the participation of O in the dataset D should not substantially change the result of statistics S.

The difference between Dalenius' definition and Dwork's is subtle but important. To clarify, let us assume for instance that Bob believes that everyone in Norway has four arms, and it is common knowledge that Alice is from Norway. After a statistical analysis of the database D, it is concluded that everyone in Norway in fact only has two arms. This means that it is now known that Alice also has two arms. The question is whether Alice's privacy is violated, even if Alice might not even be in the analyzed database? According to Dalenius, her privacy has been violated because we have learned something about an individual from the analysis. According to Dwork however, Alice's privacy has not been violated because the outcome of the query is essentially to be the same, whether or not Alice was in the database. This definition by Dwork ultimately became the basis of differential privacy [33].

2.2 E-Health

E-Health is the use of technology and information systems within healthcare. A lot of technology is applied in modern healthcare in order to conduct efficient service, improve costs and utility, and provide valuable insight with the help of medical monitoring devices.

These applications generate a lot of data, and a part of this data is useful for analysis or for building helpful ML models. However, most of the interesting data, from a clinical study, for instance, is privacy sensitive. Using this data in statistical analyses or ML should not risk violating the integrity of the people in the dataset. Privacy-preserving models such as anonymization have been proven to not be fully sufficient [35, 9], and techniques such as k-anonymity can still be privacy vulnerable [36].

An exemplary case is the re-identification of politicians via the browsing history of three million German citizens in 2016 [37], where their medical information and sexual preferences were uncovered from an anonymized dataset. Months before this, the Australian Department of Health released anonymized medical records of 10% of their population, which took only six weeks for researchers to carry out successful re-identify [37].

Considering this, privacy breaches or threats can be divided into three main categories [36].

- *Identity disclosure*: An attacker associates an individual's identity with its records in a published dataset.
- *Attribute disclosure*: An individual's sensitive attribute is disclosed with the help of externally available information.
- *Membership disclosure*: The presence of an individual's record in a published dataset can be guessed with high accuracy.

Consequently, there needs to be a technical solution for protecting an individual's integrity and this is where differential privacy may turn out to be useful, with a rigorous promise for individual integrity.

2.3 GDPR

In addition to the ethical issues of techniques of storage, processing, and sharing sensitive data, privacy protection has become a requirement from the legislation. In 2018 the EU introduced the General Data Protection Regulation (GDPR) [3] to set ground rules for how personal data should be handled and regulate the right to data protection. Fundamentally, it is a system with a set of principles that are actualized by laws. The core principles are fairness and transparency, which state that it must be clear and open to how personal data is processed and stored. Moreover, storing

and processing of personal data requires explicit approval, or under conditions complying with the law. With legislation constraints, there is a need for technologies such as machine learning to consider whether privacy violation is incurred in their usage of individual healthcare data.

Medical research relies on the processing of certain health data, which is defined by GDPR and generalized as personal data. Article 4 of GDPR states: [38]

> *"'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person can be identified, directly or indirectly, ..."*

One seeming method to avoid violation against privacy legislation is anonymization [39]. Anonymized data is not regarded as personal data, and is thus outside the scope of GDPR. Recital 26 states: [40]

> *"The principles of data protection should therefore not apply to anonymous information, namely information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable. This regulation does not, therefore, concern the processing of such anonymous information, including for statistical or research purposes."*

This notion of anonymization implies the prevention of identifiability, which in technical terms means the capacity of protection against identity disclosure. However, GDPR might not provide a complete solution for the issue of data anonymization [39], as anonymization fails to preserve privacy in some cases. The technology of pseudonymization implies that the processing of personal data should prevent identity disclosure without the use of additional information, provided that such additional information is kept separately. However, such a definition does not indicate a guarantee of complete anonymity.

Furthermore, GDPR considers a dataset anonymous when re-identification is only possible with high effort or unlikely means [39]. Nonetheless, various anonymization methods have been demonstrated to produce data that is not legally anonymous, given access to alternative data sources [36]. Therefore, in practice, more concrete measures should be taken on medical data life-cycle, rather than merely anonymization.

2.4 Differential Privacy

Differential privacy (DP) manages privacy in a way that differs from traditional anonymization techniques. It is a mathematical definition of privacy that, by introducing carefully calculated random noise, retains the statistical properties of the dataset while prohibiting that personal private information can leak during analysis. DP guarantees that an individual will be exposed to essentially the same privacy

risk whether or not her data are included in a DP analysis. I.e., it ensures that individuals' 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 randomized mechanism M is said to give (ϵ, δ)-DP for all events S, $S \subseteq \text{range}(M)$, given two adjacent databases D and D', such that

$$\Pr[M(D) \in S] \leq e^{\epsilon} \cdot \Pr[M(D') \in S] + \delta \tag{2.1}$$

Where range(M) denotes the range of output with given input, and Pr[ ] denotes probability distribution [34]. If $\delta = 0$, the randomized mechanism M (see Section 2.4.4) gives ϵ-DP by its strictest definition. The parameter ϵ refers to the privacy budget (see Section 2.4.1), 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 therefore there is no risk of privacy disclosing associated with joining or refraining from joining a database [33, 41].

In practice, DP can be applied during three different stages of the data analysis, as shown in Figure 2.1:

1. *During data collection*, where data subjects introduce noise before sending data to the curator, who aggregates to get statistical info [42, 4]. This does not require the subject to trust the curator. This method is also called the *local model* and will not be covered in this thesis.
2. *Before data release*, by the data curator to the dataset itself, i.e., before any analysis task is defined. This means that the subjects send their raw data to the curator, who has to be trusted by the subjects. The curator then adds noise to the data in the dataset or trains an ML model on the data that can generate new rows of the data but maintain the statistical properties of the original dataset. By applying this method the curator can, by the composition properties of differential privacy (DP), use non-privacy-protected tools for analysis with the privacy guarantees of DP. Furthermore, the perturbed datasets can be released to anyone for analysis without the risk of violating the privacy of any of the subjects. This method is applied by the synthetic data tools in our evaluation [4].
3. *During data analysis* process where noise is added directly to the answer of statistical queries or during training of ML models. This is the method used by both the statistical query and ML tools in our evaluation [4].

For clarity on how DP is applied during different stages of data analysis, we show an illustration in Figure 2.1. The first pane shows an overview of the process when DP is applied *During data collection*, the second shows *Before data release* and the third pane shows *During data analysis*. The Orange arrow shows (1) a query to the

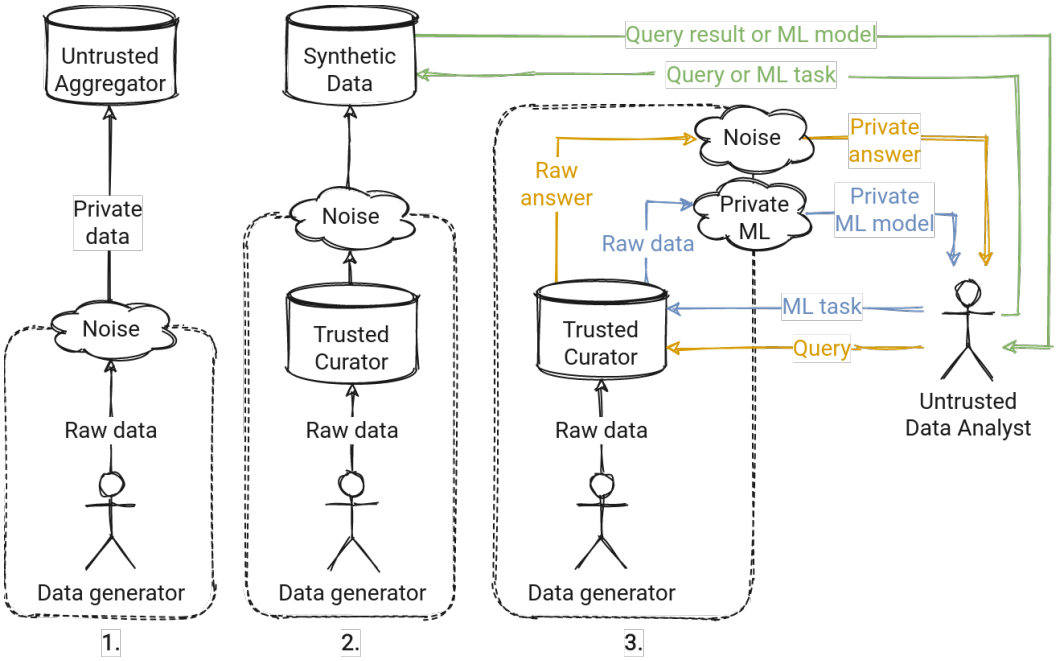

Figure 2.1: Stages of data analysis where DP is applied. **1.** During data collection. **2.** Before data release. **3.** During data analysis.

trusted curator who (2) adds noise to the raw answer and outputs a differentially private (DP) answer. Blue arrows show a request for an ML task to the trusted curator who trains a DP-ML model. Finally, green arrows show queries running directly on the synthetic data and ML model training directly with the synthetic data.

Additionally, since DP tools for ML and synthetic data generation use a variation of DP, referred to as *Rényi differential privacy* (RDP) [27], we briefly introduce its central idea. Essentially, RDP makes use of the Rényi divergence to measure the distance between distributions. Rényi divergence works in the notion of a parameter referred to as its *order*, or α. The idea is to search for the order that optimizes ϵ. This is further elaborated in Section 2.4.4 and 2.6.1.

2.4.1 Privacy Budget

Essentially, noise is added to each access to the underlying dataset and is regulated by a *privacy budget*. This noise is commonly generated from the Laplace or Gaussian distribution (Section 2.4.4) and added into a function to perturb the data. The central idea of the privacy budget is to guarantee bounds on how much information may be revealed by someone's participation in a database.

These bounds are described by two numbers, ϵ, i.e., the multiplicative bound, and δ, i.e., the additive bound. Together they produce (ϵ, δ)-differential privacy, defined in Definition 2.1. δ is added to the multiplicative bound and represents essentially that the multiplicative bound does not apply to everyone. I.e., the possibility that a few individuals may stand to lose more privacy than the rest. The risk is basically small

if δ is small. However, in this thesis, we are primarily interested in the multiplicative bound described by ϵ. This parameter is roughly the amount of information an analyst might gain regarding an individual and is commonly referred to as the privacy budget.

Describing the privacy budget from an attacker's perspective gives intuitive reasoning of the level of privacy it represents and paints a generic scenario for how one could approach it [35]. For this purpose, we assume that we have a mechanism A which is ϵ-differentially private. We run it on some database D, and release the output $A(D)$ to an attacker. Furthermore, we assume a powerful attacker that knows all of the data in the database except the data of the target. The attacker wants to determine which database is the real one between two options D_{in} (with the data of the target) and D_{out} (with the data of the target).

Under differential privacy, the attacker can only have an initial suspicion, represented by a probability P, of what database the target is in. We suppose that the attacker sees that the mechanism A returns an output O. By looking at O, the attacker's initial suspicion P is guaranteed (by ϵ-DP) to change by at most an upper and lower bound represented by the ϵ parameter. This is essentially the fundamental property of DP.

2.4.2 Sensitivity

The *sensitivity* of a function f measures essentially how much its output can change (in the worst case) when we add one record in the database. I.e., the sensitivity of f is a measure of how revealing f might be. It gives an upper bound on how much its output must be perturbed to preserve privacy. Another way of thinking about it is that it parametrizes how much noise perturbation is required by the DP mechanism. E.g., if Δf is the sensitivity of a function f, then adding Laplace noise with scale $\frac{\Delta f}{\epsilon}$ preserves (ϵ,0)-DP [5].

Global sensitivity refers to the consideration of all possible neighboring databases,[1] while *local* sensitivity refers to the change in one neighboring database. Moreover, the distance we need to use depends on which noise function we are adding. For instance, Laplace noise is scaled by the $L1$ sensitivity, which is based on the Manhattan distance. One central parameter that determines how accurately we can answer numeric queries is their $L1$ sensitivity. By contrast, Gaussian noise is scaled by the $L2$ sensitivity, which is based on the Euclidean distance.

2.4.3 Composability

Composability provides an essential property of differential privacy (DP). Given the quantification of privacy loss, we can analyze and control cumulative privacy loss

[1] Differing in at most one element

over multiple computations. In other words, composability permits programmability, i.e., simple DP primitives can be combined and as a result, preserve privacy in a DP manner for more complex analytical tasks. This property is the key to why any analysis is DP if it is run on a synthetic dataset that has been generated with a process that is DP.

Composability implies essentially that publishing the result of two mechanisms, ϵ-DP M_1 and ϵ_2-DP M_2 is $(\epsilon_1 + \epsilon_2)$-DP [35]. E.g., if we query an ϵ-DP mechanism t times the result would be $(t\epsilon)$-DP, given that the randomization of the mechanism is independent for each query. This is referred to as *sequential composition*. The variant where M_2 depends on the value of M_2 is called *adaptive composition*. If M_1 and M_2 are computed on disjoint subsets of the private database, then a function of them would be $(\max(\epsilon))$-DP instead. This is referred to as *parallel composition*, which allows us to build locally differentially private mechanisms (Table 2.1).

2.4.4 Mechanisms

Generally, three essential mechanisms are used to apply differential privacy (DP) in statistics, and are implemented by default in the tools considered in our evaluation, i.e., *Laplace mechanism*, *Exponential mechanism*, and *Gaussian mechanism*. However, any mechanism meeting Definition 2.1 can be considered as DP.

The Laplace mechanism [5] involves adding random noise that adjusts to the Laplace distribution. It can result (ϵ)-DP when the scale of the noise for a query q is calibrated as (sensitivity of q)$/\epsilon$, where $\delta = 0$. Notice that this DP privacy guarantee is independent of each query response. Given a strongly desired privacy guarantee, more sensitive q corresponds to more noise to achieve that guarantee.

However, since programming the Laplace mechanism involves real values, precautions must be taken due to present vulnerabilities in the implementation of floating-point numbers [5, 43]. Otherwise, DP can be attacked [43]. Furthermore, the Laplace mechanism is mainly suitable for low-sensitivity queries. It needs low ϵ values for a large number of queries, and low ϵ values produce less accurate results to achieve the privacy guarantee. It also cannot by itself be applied to the non-numeric valued queries like *"What is the most common dish in Chalmers restaurants?"*.

To address the issues with the Laplace mechanism and help to extend the notion of DP, the Exponential mechanism [5] is commonly applied in addition to the Laplace mechanism. It randomizes the results and is paired with an application-dependent score function that evaluates the quality of an output [44]. Furthermore, it works for any queries whose output spaces are discrete, which enables DP solutions for problems where the outputs are not real numbers. It can therefore be applied for synthetic dataset generation where the label of data is not numerical [45].

The Gaussian mechanism [5] on the other hand protects privacy by adding randomness with a Gaussian distribution and provides (ϵ, δ)-DP. Since Gaussian noise is

scaled by the $L2$ sensitivity, it allows adding less noise for applications in which $L2$ sensitivity is lower than $L1$ sensitivity.

When it comes to whether to use Laplace or Gaussian noise, we can consider mainly two things: (1) whether we want a non-zero δ and (2) how many statistics a single individual can influence. Consequently, Gaussian is not an option given a non-zero δ. Furthermore, Laplace is preferable if a single person can impact at most one statistic. Gaussian on the other hand is preferable if a single person can impact many statistics.

Gaussian noise is also commonly used in DP-ML. It can be used to reduce the noise magnitude if many statistics are computed on the same data, even if the statistics are unrelated. For instance, to make the stochastic gradient descent DP, where each data point influences many coordinates of a vector at each iteration, noise is added to all coordinates, which makes Gaussian noise a suitable choice. This is further elaborated in Section 2.6.

Nonetheless, both Laplace and Gaussian mechanisms come with shortcomings, and Rényi Differential Privacy (RDP) has been suggested as a generalization of DP to prevent them [27]. RDP is a notion of privacy that is in between ϵ-DP and (ϵ, δ)-DP. It was introduced by Abadi et al. who found a way to track the privacy loss of Gaussian mechanisms. In contrast to Definition 2.1, which evaluates DP in terms of distance measures between distributions, RDP measures distance by the Rényi divergence. It is bounded by ϵ in (α, ϵ)-RDP, where α denotes the order of the divergence. Furthermore, this definition can be extended to (α, δ)-RDP that particularly allows analysis of Gaussian noise and is useful in DP ML.

2.4.5 Applications

Differential privacy (DP) allows probability distributions of aggregated data to be recovered, which makes it useful for tasks such as data analyses and machine learning (ML). Moreover, given the possibility of injecting noise in data in a data-independent way makes it a suitable privacy layer for existing applications.

Since DP can be applied during three different stages of the data analysis (see section 2.1), DP applications can practically be divided into these categories. In regard to our evaluation, we aim our attention at eHealth applications that benefit from data release (also referred to as data publishing) and data analysis.

DP supports a broad spectrum of analytical applications in these categories. I.e., applications that make use of count queries, averages, and histograms for clinical studies, for instance, ML, e.g., linear regression models from patients' health-monitoring devices, and generating synthetic data.

The main distinction between these applications, and ultimately the advantage of synthetic data generation (SDG), is that SDG allows data analyses without con-

sidering privacy. I.e., it allows using existing tools on the DP synthetic dataset that can be conducted any number of times without increasing the privacy risk. Consequently, it enables collaboration between parties and ultimately democratizes knowledge. It is therefore a compelling and interesting option for companies in practice.

2.5 Statistical Queries

Statistical queries are generally conducted in statistical analysis, which is the process of collecting and analyzing data to identify patterns and trends. Protecting individuals' privacy during this process requires a strong notion of privacy since it is demonstrated by Dinur and Nissim [9] that it is impossible to publish arbitrary queries on a private statistical database without revealing any private information. Differentially private (DP) statistical query tools, however, allow users to run a sequence of queries, with a robust level of privacy guarantees of individuals' privacy. In these settings, the database is typically responsible for protecting an individual's privacy.

Essentially, results in DP statistical queries are perturbed to each access to the underlying dataset to protect individuals' privacy. This is commonly attained with noise-based techniques. Implementing DP statistical query tools involves typically a trusted curator, illustrated in Figure 2.1. The curator's primary job is to control if queries are DP and to keep track of the total privacy budget spent, i.e., to check if too many queries have been made already, which would compromise the privacy guarantee.

Privacy budgeting is integrated and available in the tools considered for our evaluation and is an important functionality for tracking and limiting information disclosure. The privacy budgeting tracks queries to ensure individual queries and their aggregations with other queries do not violate desired privacy demands.

The DP statistical query tools that we considered for this evaluation provide a SQL data access layer that allows users to conduct analysis using SQL language. It makes the tools intuitive to use and integrate into existing analytics workflows.

2.6 Machine Learning

Machine learning (ML) generally means obtaining knowledge from data. The knowledge is often represented in the form of a model and the process of obtaining it is commonly referred to as an ML task. When we train a model, the goal is essentially to learn the general patterns of the data by minimizing a *cost function*. It is worth noting that an error or loss function is just a part of a cost function, which is a type of objective function. This is sometimes used interchangeably. Nonetheless, in linear regression, which was the main ML task in this evaluation, this is done by fitting a curve, which in this case is a line. The goal is, given some data points, to

find the best line that fits the data points. Generally, the cost function in linear regression is formalized as the *Mean Squared Error* (MSE).

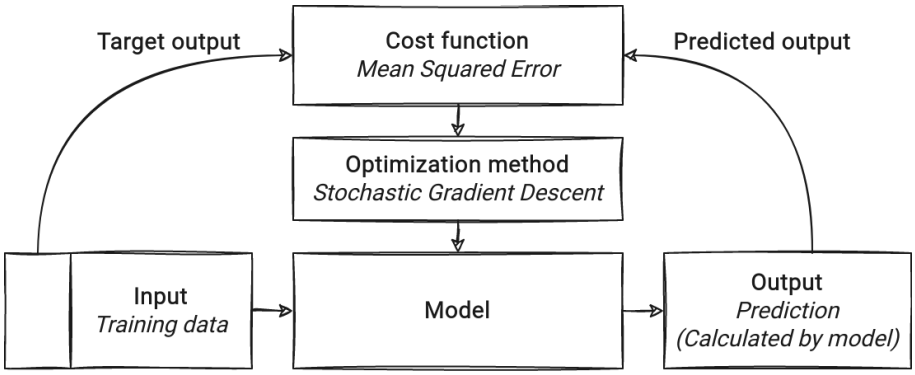

Figure 2.2: A general outline of the iterative process of finding a minimum of the cost function of an ML model for the given training data.

For clarity, Figure 2.2 illustrates a general outline of the iterative process of finding a minimum of the cost function of an ML model for the given training data. It shows how the cost, also referred to as error or loss, is commonly expressed as a difference between output prediction by the model and actual/target output which is given as a part of training data. The loss is calculated by a cost function and the minimum of the cost function is found by an optimization method, SGD in this case. Different ML algorithms can be implemented using different models, cost functions, and optimization methods.

We can implement a linear regression model by solving the model parameters analytically, which is also referred to as closed-form equations. The closed-form solution directly computes the model parameters that best fit the model to the training set. It is commonly applied on small datasets since there is computational costly matrix inverse computation involved. Another way of doing it is by using an iterative optimization algorithm, such as *Stochastic Gradient Descent* (SGD), that gradually tweaks the model parameters to minimize the cost function over the training set. This approach converges eventually to the same set of parameters as the first method and can be seen in Figure 2.2.

The main difference between these methods is that the training dataset for the closed-form solution has to fit into RAM, while for SGD one can run the training set one data point at a time and update the parameters according to the error gradient. The model can be updated with a new batch of data without retraining using the whole dataset. It is better suited for cases where there are a large number of features, or too many training instances to fit in memory.

Moreover, SGD tells in which direction and how far a model should update its internal state. This is also referred to as optimizing the cost function, i.e., the process of minimizing loss by following the gradients of the cost function. Since gradient

descent is an algorithm that minimizes a convex function, and because linear regression cost function MSE is convex and therefore differentiable, we can denote SGD as a cost function for simple linear regression tasks. This can be attained by using only one linear layer in a deep learning model for instance to minimize the squared distance.

There are two main hyperparameters that affect the outcome of an SGD, the learning rate, and batch size. The learning rate controls how much the model changes in response to the estimated error each time the model weights are updated. The batch size controls the number of training samples to work through before the model's internal parameters are updated. Batch size is basically a slider on the learning process. For instance, small batch sizes converge the learning process quickly at the cost of noise in the training process, while large batch sizes converge the learning process slowly with accurate estimates of the error gradient. Generally, a larger batch size gives a higher learning rate.

2.6.1 Differentially Private Machine Learning

Recently, Fredriksson et al. [46], among others, demonstrated that sensitive data can leak from ML models. To address this issue and protect the privacy of the individuals, who's data the ML models are trained on, differential privacy (DP) can be incorporate in the ML process. There are primarily three techniques at different stages of the optimization process (ML task) where this can be done, A) *objective perturbation* [47], B) *output perturbation* [48] and C) *gradient perturbation* [23].

Technique A) runs a standard optimization algorithm, on a modified objective function, to which noise is added. B) considers the whole optimization process as a single algorithm and adds noise directly to its output, and C) applies to SGD where Gaussian noise is added to each stochastic gradient. For the evaluation in this thesis, we focus on techniques A) and C) that are implemented by the ML tools (Section 1.7) to achieve DP during model training. In the case of linear regression tasks, Diffprivlib incorporates [45] to enforce ϵ-DP guarantee of the linear regression model. This mechanism perturbs the objective function of the optimization problem rather than its results. Tensorflow Privacy and Opacus leverage technique C).

Technique C), gradient perturbation, was originally proposed by Abadi et al. who applied it in a deep learning algorithm [23]. Its variations are the main algorithms for private training of large-scale neural networks [24]. They developed a technique called the moment accountant that later motivated the introduction of Rényi Differential Privacy (RDP) [27]. Basically, they developed a gradient perturbation mechanism, DP Stochastic Gradient Descent (DP-SGD), where noise is added at every step of the stochastic gradient descent.

Similar to the idea presented by Zhang et al. (technique A), the privacy in DP-SGD is achieved by perturbing parameter gradients that the model uses to update its weights rather than the data directly [45]. Basically, instead of adding noise to a

result of an analysis to achieve DP, DP is integrated into the training of the ML models [49]. This prevents the model from memorizing its training data while still enabling learning in aggregate. However, outliers that have larger gradients than most samples are at risk of being memorized by the model during training. The goal is therefore to limit how much each sample can contribute to the gradient and to achieve this, the gradient is computed for each sample in smaller batches called mini-batches. The gradients computed for each of the mini-batches are then clipped and accumulated into a single gradient tensor to which noise is then added.

However, while DP-SGD has been shown to achieve practical accuracy-privacy trade-off [23], its real-world application was limited due to considerably higher training time compared to non-privacy protected SGD. This problem has led to different optimized implementations of DP-SGD. Originally, the gradient is computed for the entire batches due to performance optimization. This introduced a distinct challenge as most differentiation functionality provided by most software frameworks did not support computing gradients with respect to individual examples in a mini-batch at the time. This challenge was solved using an efficient technique for obtaining all of the desired gradient norms when training a standard neural network [50]. Others have done further improvements to the technique. We mention recent work that has mapped current techniques in Section 1.8.2.

Furthermore, in regards to DP-SGD, RDP allows for tighter analysis of cumulative privacy loss, which basically reduces the noise that must be added during the training process to satisfy a particular privacy budget [27]. It is therefore especially well suited to analyze the DP guarantees provided by sampling and Gaussian noise addition in DP-ML.

2.7 Synthetic Data Generation

Synthetic data generation (SDG) has become a practical way to release fake data for analysis purposes. In addition, it is also applied as a statistical disclosure technique, aiming to preserve an individual's privacy. However, these techniques lack formal privacy guarantees that differential privacy (DP) can provide. They are vulnerable to re-identification and/or membership attacks, and must thus be formally privatized [51, 52, 53].

In recent years, DP has been incorporated into SDG to allow the release of data while preserving an individual's privacy [54, 55, 56, 57]. The DP synthetic dataset is generated from a statistical model that is based on the original dataset. I.e. it represents a synthetic sample, derived from the original dataset, while retaining as many statistical properties as possible. Different techniques exist to generate DP synthetic data, and in our evaluation, we focus on SGD for tabular data.

The benefit of SDG is that the whole synthesized dataset could potentially be disclosed, even to untrusted parties, without concerns of disclosing private data. We could then conduct ML tasks and other types of analysis on the synthetic dataset.

It allows the consumer of the dataset to use the same tools and data pipeline as if the data was not DP. This eases the adoption when a consumer wants to leverage DP in practice. E.g., in the case of statistical analysis, the query number is no longer restricted by a privacy budget under SDG. In other words, SDG takes into consideration all possible queries that can be issued on the dataset.

Smartnoise Synthetics provide three techniques that were considered for our evaluation:

- *Multiplicative Weights Exponential Mechanism* (MWEM): achieves DP by combining multiplicative weights and Exponential mechanism techniques [58].
- *DP Conditional Generative Adversarial Network* (DPCTGAN): takes the state-of-the-art CTGAN for synthesizing tabular data and applies DP-SGD (Section 2.6.1) [12, 59].
- *Private Aggregation of Teacher Ensemble CTGAN* (PATECTGAN): takes the state-of-the-art CTGAN for synthesizing tabular data and applies PATE [12].

The tool of Gretel Synthetics applies a different approach that leverages Tensorflow Privacy for DP-SGD [60]. However, they replace the SGD optimizer with an RMSProp optimizer specifically modified for Recurrent Neural Networks (RNN). While no Whitepapers are available at the moment, Gretel states the following in their FAQ: [61]

> *"Gretel-synthetics utilizes a sequence-to-sequence architecture to train on a text dataset and learn to predict the next characters in the sequence. Gretel-synthetics uses a Long-Short Term Memory (LSTM) artificial neural network to learn and create new synthetic examples from any kind of text or structured data."*

Nevertheless, the theoretical aspect of these techniques, including Smartnoise Synthetics, is outside the scope of this thesis. However, it is worth noting that DP-CTGAN and PATE-CTGAN are specifically suited for tabular data [12].

3
System Architecture of Tools

In this chapter, we explain the inner workings of the open-source libraries that are considered for our evaluation. We elaborate on how they incorporate differential privacy, essential mechanisms they implement and compare their functionalities. For a complete list of leveraged tools, see Table 1.7.

3.1 Statistical Queries

For this evaluation, we implement *OpenDP Smartnoise* (also referred to as Smartnoise) and *Google Differential Privacy* (DP) (Table 1.7). They provide functionality for conducting aggregated statistics, such as `SUM`, `COUNT`, `AVG`, `HISTOGRAM` queries, and guarantee (ϵ,δ)-differential privacy (Section 2.4). Both of the tools use the SQL language to query the data. The usage of the SQL language offers compatibility and enables the queries to fit into the structure of PostgresSQL, an established and popular open-source database system, with over 30 years of active development.

3.1.1 OpenDP Smartnoise

The OpenDP community has its roots in Harvard University Privacy Tools Project [62]. Their experience in building and deploying PSI [63], a system developed for sharing and exploring privacy-sensitive datasets with privacy protections of differential privacy (DP), has contributed to their efforts towards Smartnoise. Moreover, OpenDP [43] has incorporated insights from other DP tools such as as PinQ [21], Ektelo [13], PrivateSQL [64], Fuzz [65] and LightDP [66]. While most of the tools like PSI and PinQ are research prototypes, OpenDP is now putting in efforts into further developing DP concepts into production-ready tools.

In this thesis, we use the SmartNoise Python bindings, i.e., `smartnoise-sdk` library, to access DP functionality from the `smartnoise-core` library. The `smartnoise-sdk` library provides compatibility and can be integrated with a range of databases, including PostgresSQL, MySQL, SQL server, and plain CSV files through Python's `Pandas` module. A high-level diagram in Figure 3.1 shows how data flows in this integration. Notice that Smartnoise currently assumes that the user is trusted by the data owner and implements a shared privacy budget within a service to compose queries.

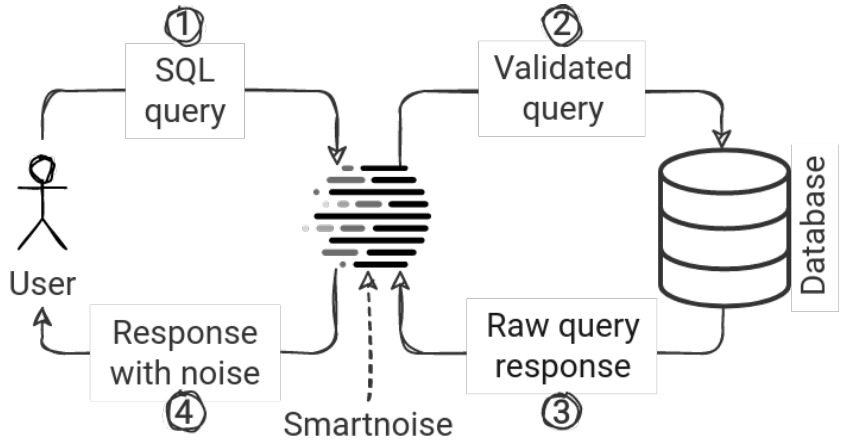

Figure 3.1: This figure shows a high-level overview of Smartnoise data flow.

The system has three layers: analysis construction, validation, and execution. They communicate via Protocol Buffers (Protobuf) messages that encode an abstraction called an *analysis*, illustrated in Figure 3.3. The `smartnoise-core` [67] written in Rust is responsible for the validation and execution. It consists of the main library `validator-rust` and `runtime-rust` were the analysis is executed, shown in Figure 3.2. The `validator-rust` provides utilities for deriving and checking if conditions are sufficient for an DP analysis, and the `runtime-rust` is an execution engine for DP analyses.

The third layer is the `smartnoise-sdk` [68] library, which contains Python language bindings and provides a programming interface for building and releasing analyses.

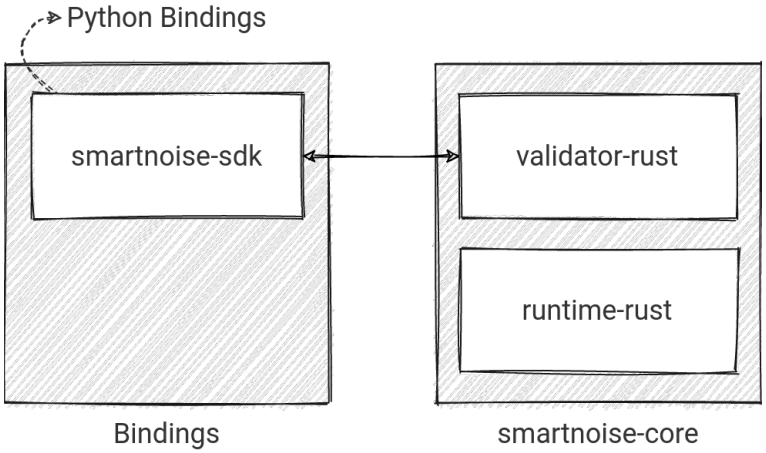

Figure 3.2: Diagram of high-level system layers, showing the three sub-projects that address individual architectural concerns and communicate via Protobuf messages.

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., `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.

Note that no data is stored in the analysis, instead, it is only stored in the *release*. An analysis graph can be validated and executed to produce DP releases of data. Releases on the other hand include metadata about the accuracy of outputs and the complete privacy cost of the analysis.

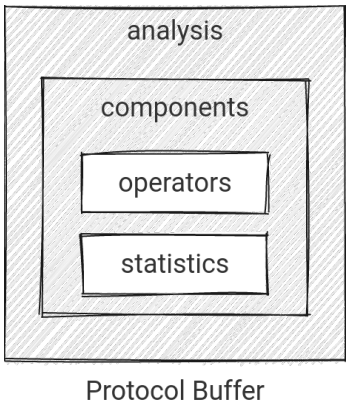

Figure 3.3: Diagram illustrating a Protobuf that encodes an analysis composed of components, which are in turn made up of either operators or statistics.

Essentially, `smartnoise-sdk` constructs and releases analyses, which are composed of components, i.e., operators and statistics such as mechanisms. The main library which is called `validator-rust`, in the `smartnoise-core` project, then determines if the components release DP data, as well as the scaling of noise, property tracking, and accuracy estimates. Finally, the `runtime-rust` library, which is also part of the `smartnoise-core` project, evaluates the components in the analysis on an arbitrary dataset.

The system is designed to be able to evaluate components on different run-times. It might be more practical to conduct non-privacy-protected aggregations and transformations on different run-times for very large datasets for instance. There may also be many sets of language bindings in contrast with the one-and-only validator. Furthermore, the validator is designed such that it never has access to private data.

Besides the Laplace and Gaussian mechanism provided by the `smartnoise-sdk` library, `smartnoise-core` provides implementations of a variety of other mechanisms,

including the Exponential mechanism.

3.1.2 Google Differential Privacy

Recently, Google released an open-sourced version of the differential privacy (DP) library that helps power some of their core products [69]. It captures years of Google's developer experience and offers developers and organizations potential benefit from their implementation, even with fairly low expertise in DP.

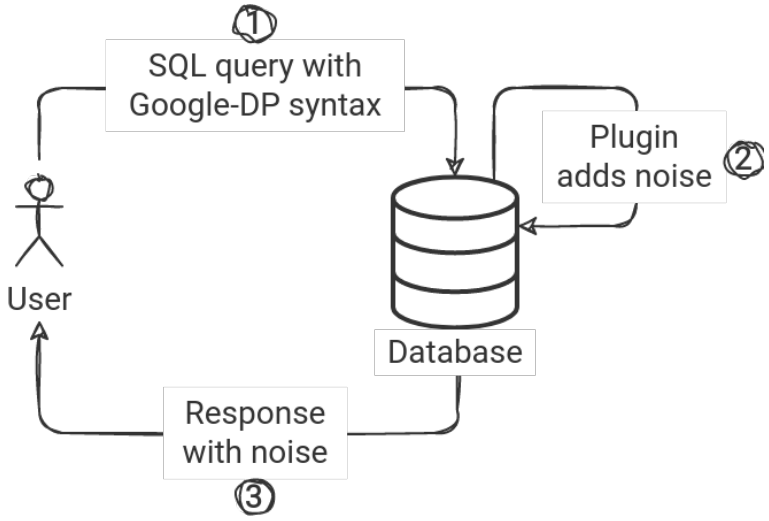

Figure 3.4: High-level overview of Google DP query flow.

In contrast to Smartnoise, Google DP offers a PostgreSQL extension that is installed within the database engine itself instead of running as a separate server, illustrated in Figure 3.4. The PostgreSQL layer is a sub-module in the C++ core library that implements noise addition primitives and DP aggregation. In addition, the core library is also available in Java and Go.

The `postgres` package implements a custom SQL syntax that requires a new semantics of queries. It provides a collection of private aggregation operators, referred to as anonymous functions or `ANON`s, that map to their corresponding DP mechanism in the core library. For instance, `sum` aggregation is passed using the `ANON_SUM`.

Moreover, all mechanisms inherit from the `Algorithm` base class in the core library and are constructed using the builder pattern. 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 [70].

An important property of Google DP is that it does not require that each user is only present in one single row [22] of the data. Google DP uses a technique called *user bounded contribution* that, instead of assuming that each user only contributes once, bounds how much each user can contribute. However, it is important to note

that `ANON`s, like their algorithms, assume that each entry is owned by a distinct user. I.e., all algorithms assume a limited number of input elements per user (1 per user by default).

This row ownership is propagated through all sub-queries for all SQL queries. Consequently, this requires that any `join` in a sub-query also joins on the user identifier. To apply this functionality, the Google DP PostgreSQL extension makes use of `ANON`s and a query rewriter. 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.

The query rewriter is responsible for performing the anonymization and the validation of the query. Applying it also requires that the sensitivity of the operators is known. This is intuitive for the `COUNT` query since each user can only affect the result with a maximum of C. However, it might get complex for operators such as `SUM`, since we have to find out how much a single user can contribute to the result. If nothing is known about the column in question, then the contribution is potentially infinity. However, such a situation can be improved if the analyst has some knowledge about the column. For instance, if the column tracks the age of the users within a database, then it is reasonable to set the bounds between 0 and 100.

To specify the bounds, the anonymous functions require lower and upper bounds of columns. If a value is outside these bounds, it will be clamped. I.e., if the value is larger than the upper bound, it will be set to equal the upper bound (same applies for the lower bound) [22]. Nevertheless, in practice, it is often hard for analysts to know the bounds of the data. The rewriter can therefore use some of the privacy budgets to calculate the bounds automatically by using histogram bin enumeration to estimate the bounds.

Google DP system design comes with some advantages. For instance, it saves the intermediate communication bandwidth between the database and the server running the privacy application. Intuitively, doing the privatization inside of the database itself might offer increased performance (as can be seen in Chapter 5). However, letting the database handle DP computations might be undesired in systems with a heavy load on the database. In comparison, Smartnoise is easier to scale horizontally, since it is stateless. Furthermore, Google DP's PostgreSQL extension is incapable of assessing normal SQL queries, since it makes use of `ANON`s in order to apply DP and thus requires that any existing queries are rewritten. Google DP is also dependent on PostgreSQL 11 and does not work on any other database engines.

3.2 Machine Learning

In this thesis, we implement *Opacus*, *Tensorflow Privacy* (TFP) and *Diffprivlib* (Table 1.7) in differentially private machine learning (ML) task. Opacus and TFP apply differential privacy (DP) by the same principle of Rényi Differential Privacy (RDP) (Section 2.4, 2.6.1), and implement DP-SGD (Section 2.6.1) into their existing op-

timizers, to preserve the privacy of each training sample during the learning of ML model [71, 72, 27]. Both libraries are equipped with privacy accounting to keep track of the privacy guarantee. The computation of ϵ is dynamic, meaning no ϵ privacy bound is provided before conducting ML tasks. Instead, an estimated ϵ is computed and tracked throughout the model training, based on provided hyperparameters, showing the strength of the privacy guarantee which comes with the trained model.

Opacus and TFP provide two relevant methods for this computation, `compute_rdp` and `get_privacy_spent` that are found in the `opacus.privacy_analysis` and `tensorflow_privacy.privacy.analysis.rdp_accountant` modules. The Rényi divergence is computed based on a list of orders, also referred to as alphas (α). For each of these orders, `compute_rdp` returns the RDP achieved by the Gaussian mechanism that is applied to gradients in DP-SGD. Thereafter, `get_privacy_spent` computes the best ϵ for a given `target_delta` value of delta by taking the minimum over all orders. [73] Opacus provides a smaller range of orders than TFP does. According to TFP [73] there is very little downside in expanding the list of orders for which RDP is computed.

In order to enforce the desired privacy guarantees, Opacus and TFP introduce two core parameters for DP-ML tasks:

- `noise_multiplier`, used to control how much noise is sampled and added to gradients before they are applied by the optimizer.
- `l2_norm_clip` (TFP) or `max_grad_norm` (Opacus), used to bound the optimizer's sensitivity to individual training points.

Additionally, TFP requires another parameter, `num_microbatches`. It basically splits mini-batches of data into multiple micro-batches and clips them on a micro-batch basis to improve the performance [73]. Opacus achieves this by clipping the per-sample gradients [74].

To clarify how TFP and Opacus incorporate differential privacy during the model training process, we present a high-level diagram in Figure 3.5. Their method to achieve DP is referred to as gradient perturbation. Note that the denotation "Cost Function" in the figure groups the actual cost function and the gradient computation, which is technically an optimization of the actual cost function (Section 2.6). It shows how the DP-SGD algorithm works, corresponding to the following: (1) sample a mini-batch of training points and computes cost, (2) compute the gradient of the cost, (3) clip gradients (per training example included in the mini-batch), (4) aggregate them back into a single parameter gradient, (5) add random noise to the clipped gradients, and (6) multiply these clipped and noised gradients by the learning rate and apply the product to update model parameters [73].

Diffprivlib, on the other hand, provides ML tasks such as linear regression and logistic regression, where DP is achieved by adding noise to the coefficients of the objective function (Figure 3.6). This method is mainly inspired by Zhang et al. [45].

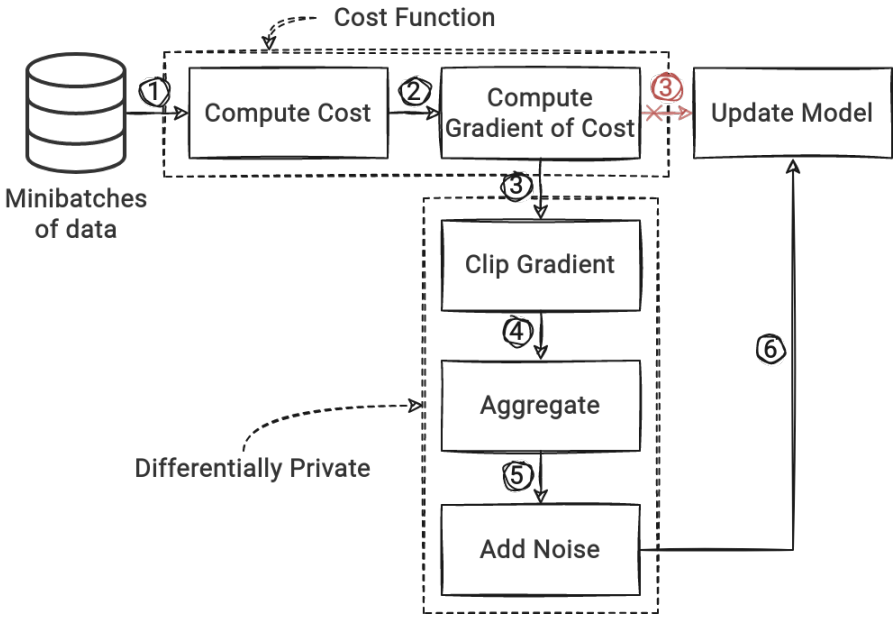

Figure 3.5: High-level overview of DP-SGD implementation during the training process of ML models.

In comparison, Opacus and TFP provide DP optimizers, which allow users to construct both linear regression and logistic regression tasks, by implementing custom layers for the ML models. Moreover, Diffprivlib has to fit the training dataset into RAM, while Opacus and TFP can run the training set one data point at a time. For this reason, Diffprivlib might ultimately be limited with an upper bound of the dataset size.

Conclusively, the main difference between Diffprivlib and Opacus and TFP, is how the ϵ parameter is set and computed. In Diffprivlib, it is straightforward, we set an ϵ value, which is the amount of noise that is added into the entire computation. However, in Opacus and TFP, noise is added multiple times, and the final ϵ value depends on other parameters, such as sample size, batch size, and the number of epochs. So target epsilons, corresponding to the `noise_multiplier` parameter, have to be manually calculated.

3.2.1 Opacus

Opacus is Facebook's library for differentially private (DP) ML, built on top of PyTorch. It was developed in collaboration with Facebook AI Research (FAIR), the PyTorch team, and OpenMined that is an open-source community dedicated to developing privacy techniques for ML and AI [75].

Opacus is built on PyTorch and enables the training of DP models by applying DP-SGD. Its main component is the `PrivacyEngine`, which is attached to the optimizer before training the PyTorch module. The module is validated by `DPModelInspector` to meet the requirements for attaching the `PrivacyEngine`. Gradients are clipped by

the `max_grad_norm` parameter and multiplied with the `noise_multiplier` parameter to calculate the noise and obtain DP models that protect the training sample's information from disclosure.

3.2.2 Tensorflow Privacy

TensorFlow is an ML framework developed and released by Google, originally inspired by the work of Abadi et al. [76, 49] who implemented a similar optimizer for TensorFlow and a privacy cost tracker. It is configurable and developers can define custom neural networks. With this flexibility, developers can implement their own operators in their applications. As a result, TensorFlow Privacy (TFP) emerged and adapted differentially private (DP) mechanisms to TensorFlow in order to allow users to leverage differential privacy in the training of ML models.

Keras is a submodule and high-level framework of Tensorflow that was used in our evaluation since the mechanisms that TFP provides are also available 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. It is now a suitable combination for users in a diverse range of industries, and provides user-friendliness while having access to all low-level classes of TensorFlow.

TFP's service also wraps existing Tensorflow optimizers, e.g., `SGD`, `Adam`, etc., into their DP counterparts. With those components, Gradients in TFP's model training are clipped by the `l2_norm_clip` parameter and the calculated amount of noise is added by the `noise_multiplier` parameter. The optimizer adds Gaussian noise to the gradient at each round of training to achieve a DP ML model.

3.2.3 Diffprivlib

Diffprivlib applies the Python module Numpy and inherits functionality from the SciKit Learn library. It extends its functionality by adding differentially private mechanisms on top of it.

Diffprivlib originally implemented The Wishart mechanism, which was shown fairly recently not to satisfy differential privacy (DP). The mechanism was used for producing positive semi-definite perturbed second-moment matrices. It is currently deprecated, as of Diffprivlib version 0.4, and will be removed in version 0.5 [77].

To clarify how Diffprivlib currently incorporates DP during the model training process, we present a high-level diagram in Figure 3.6. The Functional Mechanism [45] is an extension of the Laplace mechanism that (1) adds noise to the coefficients of the objective function, and (2) derives the model parameter that minimizes the perturbed function.

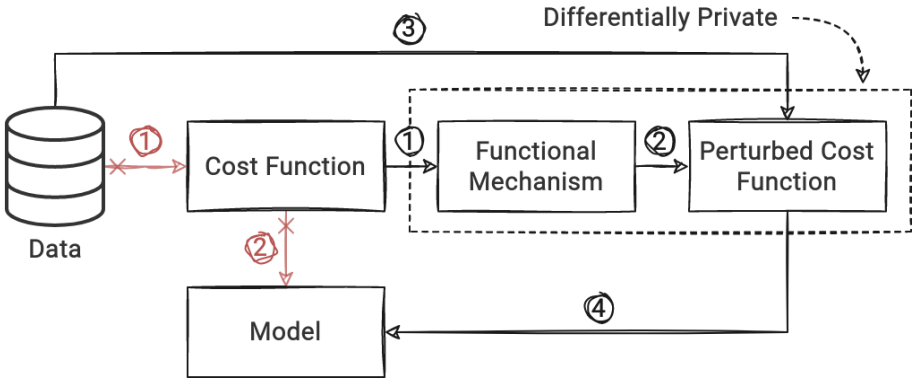

Figure 3.6: High-level overview of how differential privacy is applied to the cost function

Additionally, Diffprivlib provides `PrivacyLeakWarningCustom` warning to capture possible privacy leaks, as a result of 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 originally specified.

3.3 Synthetic Data Generation

The considered tools for statistical queries (Section 3.1) and machine learning (Section 3.2) add noise to the analytics task when the task itself is defined (the third bullet in Section 2.4). However, curators often want to release data to analysts before the task itself is defined (which requires the second bullet in Section 2.4). This is where synthetic data tools may provide a solution, by applying privacy to the dataset before the task is defined and creating a private version of the entire dataset. This is achieved by training ML models able to learn patterns in the data and generate new synthetic data points, using algorithms described in Section 2.7.

In this thesis, we look closer at two tools that can be used for generating differentially private synthetic datasets, namely Smartnoise's synthetics module (referred to as *Smartnoise Synthetics*) developed by OpenDP, and *Gretel Synthetics* developed by Gretel.ai.

3.3.1 Smartnoise Synthetics

At a high level, the synthetics module of Smartnoise consists of synthesizers and a sampler. To generate a synthetic dataset, an implementation of the abstract class `SDGYMBaseSynthesizer` can be applied that creates an instance of a synthesizer, which in turn trains a synthetic model [78]. New rows are then generated by sampling the synthetic model. To clarify the procedure, we present a high-level diagram in Figure 3.7, showing how data is used by the synthesizer to train a synthetic model. The model can then be sampled for new synthetic rows.

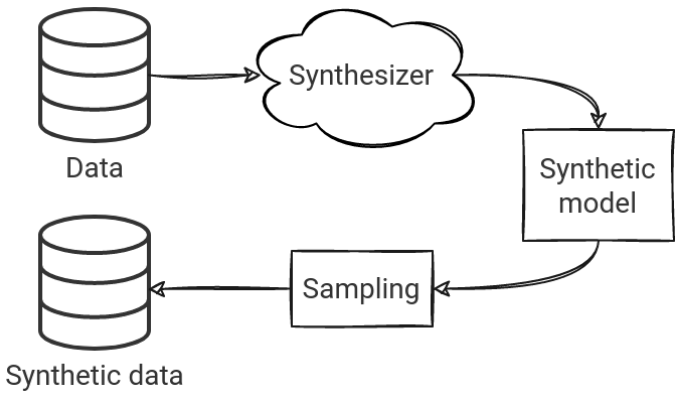

Figure 3.7: Overview of the synthetic data generation module of Smartnoise, showing the flow of data.

Smartnoise comes with three implementations of Synthesizers, (1) `MWEMSynthesizer`, without any external dependencies in Smartnoise, (2) `DPCTGANSynthesizer` and (3) `PATECTGANSynthesizer`, which both use PyTorch and the CTGAN module from SDV for the generators [79] and Opacus for the discriminator [4]. The visualization of the cooperation between the generator and the discriminator is shown in Figure 3.8. While the technical details of these algorithms are outside the scope of this thesis, some details are provided in Section 2.7 for better understanding.

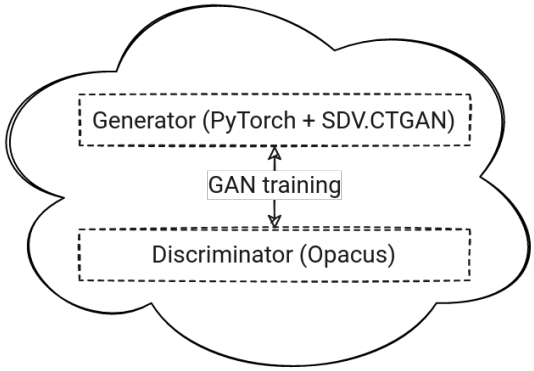

Figure 3.8: Overview of the inner working of the DPCTGAN synthesizer in Smartnoise Synthetics

3.3.2 Gretel Synthetics

Gretel synthetics try to learn patterns in data by encoding the data as rows of text and learning patterns in the text. This is done by using tokenizers, which split the text characters or subwords in the text with integer-based id:s [60]. Together with configuration for the underlying machine learning engine, the tokenizers will output

`TokenizerTrainers` that trains a model by learning patterns in the dataset. This model can then be used to generate new data by generating synthetic rows of text.

Since the model learns patterns directly from the raw text without any knowledge about what values are valid, there is a risk rows contain invalid data or that rows are badly formatted. To address this problem, Gretel synthetics provides a row validator that post-processes all rows and only allows values that are present 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 3.9.

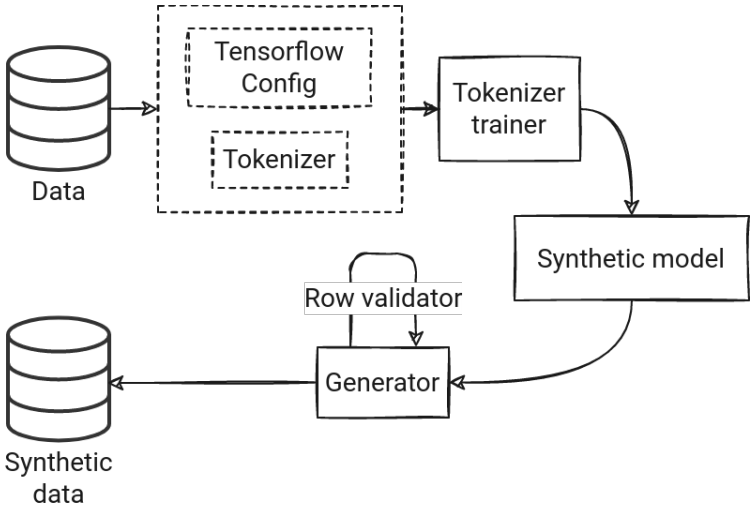

Figure 3.9: Overview of the synthetic data generation with Gretel synthetics, showing flow of data.

4
Methodology

This chapter details the methodology for evaluating differential privacy (DP) tools, and establishes the foundation for our experiments and implementation, to ultimately answer the following research questions that were introduced in Section 1.4):

- ***How much do DP tools impact data utility and system overhead in DP applications?***
- ***Can DP service practitioners apply these tools and how can they configure them?***

In search of concrete answers, we categorize DP applications that the tools constitute to guarantee that the evaluation results can be analyzed and compared within their relevant domains. We also devise the criteria to quantitatively assess the evaluation results. The evaluation plan is listed with detailed settings to show what will be conducted in the experiment, followed by how the implementation of this evaluation is achieved.

4.1 Evaluation Criteria

We adopt data utility (also referred to as accuracy) and system overhead as evaluation metrics for the considered tools in this thesis. Utility refers to whether the data is still useful to conduct a specific functionality on, after the data is perturbed with differential privacy (DP) measures. I.e., how much an outcome deviates from the true quantity it attempts to estimate. E.g., what degree of accuracy reduction is incurred, when querying on the perturbed dataset compared with that on the original result. 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 two metrics of accuracy and overhead illustrate what deviations can be expected from the DP results of the tools, and allow for comparison amongst tools' performances.

Data utility is measured by the deviation of the prediction error over several runs of the same experiment, using the *Root Mean Square Percentage Error* (RMSPE) [80]. This is essentially the percentile difference between the results from the DP and NP values shown in Equation 4.1.

$$\text{RMPSE} = \sqrt{\frac{\sum_{n=1}^{N}\left(\frac{NP\ -\ DP}{NP}\right)^2}{N}} \cdot 100\ \%, \tag{4.1}$$

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

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 an 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 explained below, we obtain 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 use the RMSPE (Equation 4.1) between the NP and the DP task.

This thesis also uses *Memory overhead* as another metric for overhead, which is measured by comparing the worst-case memory usage between DP and NP query or ML tasks. I.e., the measurement shows the percentile difference between the worst-case memory usage of DP vs NP query, or DP-ML vs NP-ML tasks, denoted as δ. Memory is measured by the worst-case, contrary to run-time and utility that is measured in RMSPE since this is the minimum system requirement for using the tools reliably. Memory usage is recorded during the time that the tools conduct DP and NP query or ML tasks. This is measured by retrieving a stream of the memory usage of the Docker containers that run the evaluation process in our evaluation (See Section 4.4). This criterion of memory overhead is considered in this evaluation since it might affect the 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 which means that we will present two different memory evaluations for these tools, both for the evaluation container and the database container. This 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.

4.2 Data Sources

When evaluating the performance of differential privacy (DP) tools (Table 1.7) with different functionalities, we carefully select datasets with the following criteria. Datasets that are (1) in the domain of eHealth, and (2) have characteristics that allow for fair evaluation, i.e., provide qualities for conducting functionalities that the tools provide.

For this evaluation, we use two public datasets: *Parkinson* (Section 4.2.1) and *Health Survey* (Section 4.2.2). Having two different datasets in the field of life-science gives

us a nuanced picture of how well the considered tools perform when DP is applied, in regard to our evaluation criteria (Section 4.1).

Datasets
2018 Massachusetts Health Reform Survey[1]
Parkinsons Telemonitoring[2]

Table 4.1: Publicly available datasets.

Both datasets are single table datasets, i.e., they can in their entirety be placed in a single CSV file. Both datasets consist of numeric data from user contributions or device usage. The *Health Survey* contains data of type `integer`, and the dataset is predominantly categorical. In comparison, the *Parkinson* dataset has continuous attributes that are of type `float`. It is important to note that *Parkinson* has only two categorical columns, which have significantly fewer categories than the majority of the *Health Survey* ones. Specifically, it implies that conducting `HISTOGRAM` queries on *Parkinson* produces fewer bins in our following evaluations.

4.2.1 The Parkinson dataset

Parkinsons Telemonitoring, or just "the Parkinson dataset", is an open dataset that was created in collaboration with 10 medical centers, who developed a telemonitoring device to record speech signals. In our evaluation, the dataset was used for conducting queries and performing ML tasks, i.e., linear regression tasks. It was originally used to predict Parkinson's disease symptoms in studies of linear and nonlinear regression methods, where the primary aim was to predict the UPDRS[3] scores. The dataset is composed of measurements from 42 people with early-stage Parkinson's disease.

Columns in the table contain subject ID, age and gender, number of days since the start, two score UPDRS scores, and 16 biomedical voice measures. There are in total 5,875 voice recordings from the participants, where each row corresponds to one recording, and generally, there are around 200 recordings per patient. The data is in CSV format.

While the Parkinson dataset is more suited for ML tasks, it contains insufficiently few individuals to obtain any useful results when conducting differentially private queries on it. We, therefore, remove user IDs and assume that each row holds records that correspond to one individual. This is due to the negative impact that differential privacy has on the accuracy of statistics that contain few individuals: given its

[1] `https://www.icpsr.umich.edu/web/HMCA/studies/37411`
[2] `https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring`
[3] Unified Parkinson's Disease Rating Scale [81]

inverse correlation between accuracy and privacy, more noise has to be injected into the obtained results when fewer individuals exist, to guarantee individual privacy in our experiments. Thus the DP result on the dataset with few individuals can run the risk of unacceptably low accuracy.

4.2.2 The Health Survey dataset

2018 Massachusetts Health Reform Survey, or just "the Health Survey dataset" is an open dataset, collected from a 2018 survey, with 10,000 individuals from Massachusetts, USA. Each row in the data corresponds to the Q&A from one participant. For this evaluation, the dataset was used for conducting queries (Table 4.4), and performing ML tasks, i.e., linear regression tasks.

This survey data contains general characteristics of participants, e.g., age, gender, income, race, etc., and information regarding access to and use of health care, health and disability status, medical treatment, etc. Each question corresponds to one column, where answers are bounded by a finite set of categories. E.g., the answer to the question *"In general, how would you say your health is?"* is constrained by six numerical options, i.e., $\{-1, 1, 2, 3, 4, 5\}$, where -1 corresponds to refused answer, 1 is excellent, 2 is very good, 3 is good, 4 is fair, and 5 is poor.

4.3 Experiment Plan

Information of individual data might be disclosed from data analyses. However, tools that prevent this by leveraging differential privacy might ultimately come with a cost of accuracy and increase the system resources [4, 12]. Our focus is therefore to study the difference between the differentially private (DP) and non-privacy protected (NP) results that these tools produce, as well as how this difference varies between different tools.

To determine the impact of privacy measures that the leveraged DP tools have on accuracy and system resource usage, we measure the difference between DP and NP results using our evaluation metrics in Section 4.1, i.e., data utility and system overhead. We vary two parameters that affect these metrics: the privacy budget (ϵ) and dataset size, to further illuminate the privacy-utility trade-off induced by differential privacy measures.

We study the impact of ϵ in correlation to dataset size since these are the parameters that trade between privacy and data utility. The amount of noise that ϵ introduces in DP computations is also dependent on the underlying dataset size. I.e., DP increases the minimal dataset size required to produce accurate statistics, which makes it challenging to obtain high data utility from relatively small datasets [82]. 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 the respective dataset considered for

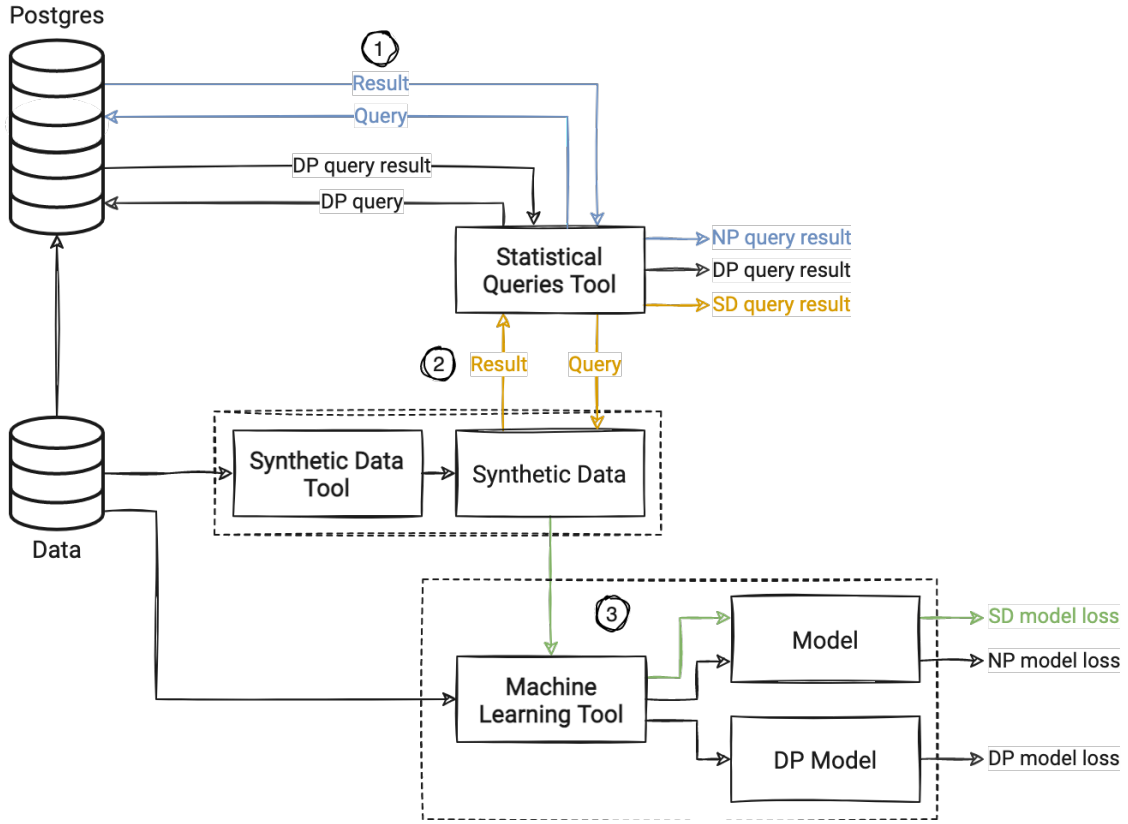

Figure 4.1: High-level overview of the experiment flow, showing *1)* DP queries and non-privacy protected (NP) queries being conducted on Postgres, *2)* queries being conducted on synthetic data, and *3)* ML tasks being conducted on synthetic data.

all the experiments 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 4.2: List of dataset sizes for each dataset.

When it comes to the justification of ϵ values, much of related literature is agnostic to the choice of ϵ. DP algorithms have previously been evaluated with ϵ ranging from 0.01 to 7, often with no justification [63]. However, Apple Health in iOS `10.2` use $\epsilon = 2$ for gathering what health data types are being edited by users [42]. Additionally, Microsoft, in collaboration with OpenDP, explains that privacy budgets are typically set between 1 and 3 to limit the risk of re-identification [83]. They state that ϵ values below 1 provide full plausible deniability and that values above 1 come with a higher risk of exposure to the actual data. Nevertheless, they recommend ϵ values between 0 and 1 when implementing DP systems.

To cover a wide range of privacy-utility trade-off results, we use the practice of ϵ in literature and industry as guidelines for selecting a set of ϵ. The set of ϵ values considered for all the experiments is listed below (Table 4.3).

Epsilon (ϵ) 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 4.3: List of ϵ values.

Furthermore, 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 allows for a reasonable comparison of tool performance within each group of tools. Details can be found in Table 1.7.

This thesis tests different functionalities for tools of different domains. We select a set of simple queries for experiments with statistical query tools. These queries are conducted on each column in the dataset, and the set of queries considered for this evaluation is listed below (Table 4.4). The actual queries using SQL in our evaluation can be found in Appendix A.2. Note that `HISTOGRAM` queries are only conducted on categorical columns since only the statistics of the categorical values can be sorted into buckets. For the ML experiments, since we want to produce comparative results, we conduct simple linear regression tasks, being that regression is the only functionalities that all the considered ML tools have in common. The procedures for conducting queries and linear regression tasks can be seen in Algorithm 2 and Algorithm 3 respectively.

Queries
Queries $qs \in \{$`SUM, AVG, COUNT, HISTOGRAM`$\}$

Table 4.4: List of queries.

For clarity on how we conduct our experiments for the different tools, we present a high-level diagram shown in Figure 4.1. It shows how data flows through the different tools and generates results. The following Sections detail and list all experiments, enumerated from 1 to 10.

4.3.1 Experiments for statistical query tools

Non-privacy protected (NP) statistical queries might reveal private information about single individuals within the dataset by aggregate statistics, such as `COUNT, AVERAGE, SUM` [9]. To guarantee the formal privacy of individuals, differential privacy (DP) has been integrated into statistical query tools. However, applying DP

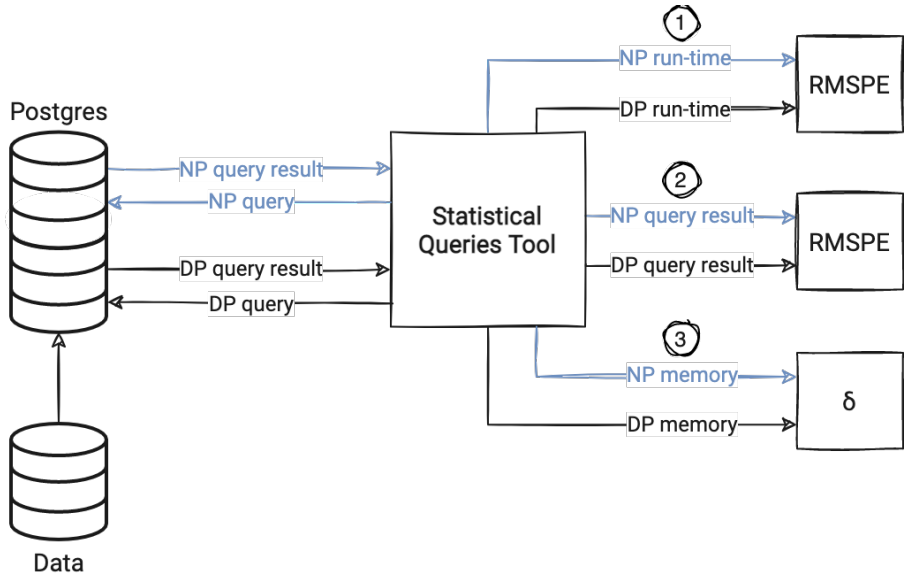

Figure 4.2: Overview of experiments flow for statistical query tools, showing how results are obtained for *1)* run-time overhead, *2)* data utility, and *3)* memory overhead.

into statistical queries can reduce the accuracy of the query results [9], and increase the system resources needed to conduct the analysis.

In these experiments, the focus is to quantify the impact of the tools (Table 1.7) on statistical queries regarding data utility and system overhead (run-time and memory overhead). The definition and calculation of data utility and system overhead are shown in Section 4.1. The two parameters that influence the results the most are dataset size and the privacy budget (ϵ) [4, 18, 58]. These parameters are therefore varied during the evaluation.

This part of the evaluation combines SQL in the implementation of statistical query services, which is arguably one of the most used tools for statistical analyses. To obtain a diverse picture of how the tools perform in general, we issue a set of queries (Table 4.4), using the standard SQL operators. Moreover, to lower the occurrence of irregularities during the experiments, each experiment is conducted 20 times, since it provides a nice trade-off between the run-time of the evaluation while reducing the occurrence of irregularities. To remove outliers, the extrema [84] are removed from the 20, and the remaining 18 is used to calculate the RMSPE.

With the experiments listed in List 1, we aim at gaining insights into how statistical queries are affected when differential privacy is integrated into the analyses, and how the considered DP tools trade-off accuracy and privacy during the analysis. We are also interested in the correlation between the performance of utility and overhead, the influencing factor of ϵ, and data size and how it differs for the different queries.

For clarity on how we conduct our experiments, we present a high-level overview

of the experiments flow in Figure 4.2. It shows how results are obtained from the experiments. Specifically, it shows how measures for data utility and system overhead are obtained by comparing DP and NP results. Below we show the detailed ingredients for this part of the experiment.

We would like to notice that we evaluate the performance of statistical queries of the following differential privacy tools: Smartnoise and Google Differential Privacy (Section 1.7). The performance criteria to measure are listed as follows:

Experiment list for statistical query tools

1. *Data utility*

    - We show how the performance of statistical query tools differ between DP and NP query results, in term of query accuracy. We display how the tools' privacy measure impacts accuracy by conducting a set of queries, qs (Table 4.4), and show how the performances vary over different combinations of data size and ϵ, using two different datasets. We anticipate finding a direct trend between ϵ and accuracy, as well as between data size and accuracy.
    - We conduct qs for a set of ϵ (Table 4.3) on
        (a) Health Survey dataset of size in Table 4.2
        (b) Parkinson dataset of size in Table 4.2
    - We study the impact on data utility by measuring the RMSPE (Equation 4.1) between DP and NP model performance.

2. *Run-time overhead*

    - With these experiments we investigate how much the run-time overhead the DP functionality adds to the statistical queries. We anticipate to find a direct trend between data size and run-time, and that run-time will generally increase when DP is integrated into statistical queries.
    - We conduct queries for a set of ϵ (Table 4.3) on
        (a) Health Survey dataset of size in Table 4.2
        (b) Parkinson dataset of size in Table 4.2
    - We measure impact on run-time overhead by measuring the difference in time usage between DP and NP queries as shown in Algorithm 2.

3. *Memory overhead*

    - With these experiments we aim to gain insights into how much the memory overhead the DP functionality adds to the statistical queries. We anticipate a marginal increase in memory overhead when DP is integrated.
    - We conduct queries for a set of ϵ (Table 4.3) on
        (a) Health Survey dataset of size in Table 4.2
        (b) Parkinson dataset of size in Table 4.2

- We measure impact on memory overhead by measuring the difference in memory usage in both the container running the database and the container running the evaluation when issuing DP and NP queries to the database.

4.3.2 Experiments for machine learning tools

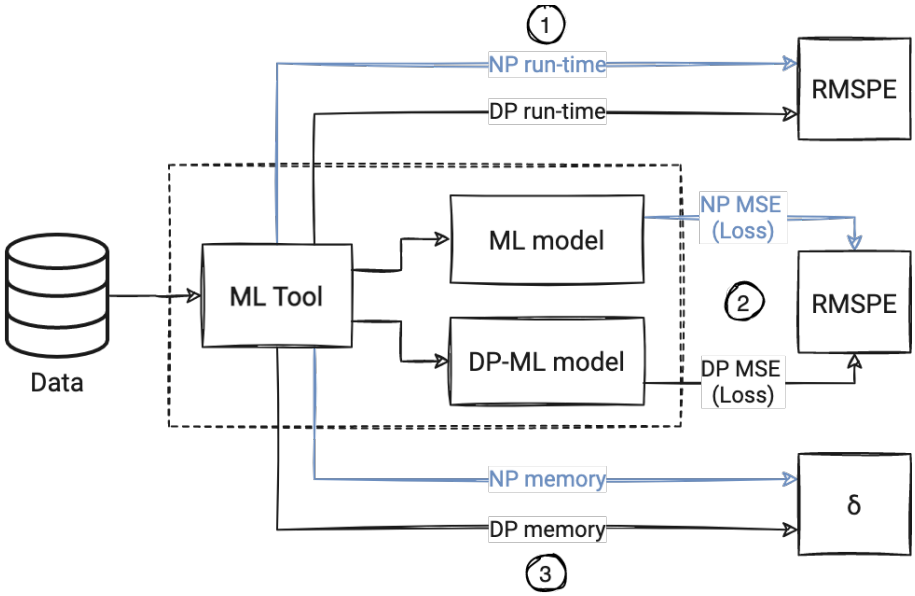

Figure 4.3: Overview of ML experiments flow, showing how results are obtained for *1)* run-time overhead, *2)* data utility, and *3)* memory overhead.

While non-privacy protected (NP) ML models can disclose information of individual data in the training data [46], differentially private (DP) ML models that address this issue can at the same time reduce model accuracy and increase system resource usage. Our focus in these experiments is to quantify how much the performance of ML tools differ between DP and NP ML tasks, as well as how this difference varies between different ML tools.

To determine the impact that the ML tools have on data utility and system overhead (run-time and memory overhead), we measure how much the DP results differ from the NP results that have been obtained with the ML tools. To further illuminate the privacy-utility trade-off, we adopt a range of experiment settings over which we observe how the performance varies. The definition and calculation of data utility and overhead are shown in Section 4.1. Since ϵ and data size are the two parameters that affect the result the most, these two parameters are therefore varied during the evaluation [4, 18, 58].

Moreover, to lower the occurrence of irregularities during the experiments, each experiment is conducted 10 times, since it provides a nice trade-off between the run-time of the evaluation while reducing the occurrence of irregularities. To remove outliers, the extrema [84] are removed from the 10, and the remaining 8 is used for

calculating the RMSPE. Note that due to computational time restraints, we had to deviate from experiment 1, 2 and 3 that run 20 times, more details in Section 5.4.

With the experiments listed in List 4.3.2, we aim at gaining insights into how ML functionality can be affected when differential privacy is integrated into the training, and how the considered DP-ML tools trade-off accuracy and privacy through the experimental results. We are also interested in how the performance of utility and overhead varies with different ϵ and data size.

For clarity on how we conduct our experiments, we present a high-level overview of the experiments flow in Figure 4.3. It shows how results are obtained from the ML experiments. Specifically, it shows how measures for data utility and system overhead are obtained by comparing DP and NP results. Below we show the detailed ingredients for this part of the experiment.

Note that we evaluate the following machine learning tools: Tensorflow Privacy, Opacus, and Diffprivlib (Section 1.7). We illustrate what and how to evaluate the DP tools in machine learning tasks:

Experiment list for machine learning tools

4. *Data utility*

    - Experiments in this part aim to show how the performance of ML tools differ between DP and NP ML tasks. We specify the tasks as linear regression (LR) to show how the performances vary over different combinations of data size and ϵ, using two different datasets. We anticipate to find a direct trend between ϵ and accuracy, as well as between data size and accuracy.
    - We conduct LR for a set of ϵ (Table 4.3) on
        (a) Health Survey dataset of size in Table 4.2
        (b) Parkinson dataset of size in Table 4.2
    - We study the impact on data utility by measuring the RMSPE (Equation 4.1) between DP and NP model performance (loss).

5. *Run-time overhead*

    - We conduct experiments aiming to gain insights into how much run-time overhead of ML functionality can be affected by the ML tools when DP is integrated. We conduct LR tasks to show how run-time varies over different combinations of data size and ϵ, using two different datasets. We measure the impact on run-time overhead by conducting LR tasks. We anticipate the run-time to generally increase when DP is integrated into LR.
    - We conduct LR for a set of ϵ (Table 4.3) on
        (a) Health Survey dataset of size in Table 4.2

(b) Parkinson dataset of size in Table 4.2

- We study the impact on run-time overhead by measuring the difference in time usage between DP and NP LR tasks.

6. *Memory overhead*

- We aim to gain insights into how much the memory overhead of ML functionality can be affected by the ML tools when DP is integrated. We conduct LR tasks to show how the performances vary over different combinations of data size and ϵ, using two different datasets. We measure the impact on memory overhead by conducting LR tasks and anticipate a marginal increase in memory overhead when DP is integrated.
- We conduct LR for a set of ϵ (Table 4.3) on
  (a) Health Survey dataset of size in Table 4.2
  (b) Parkinson dataset of size in Table 4.2
- We study the impact on memory overhead by measuring the difference in memory usage during DP and NP LR tasks.

4.3.3 Experiments for synthetic data tools

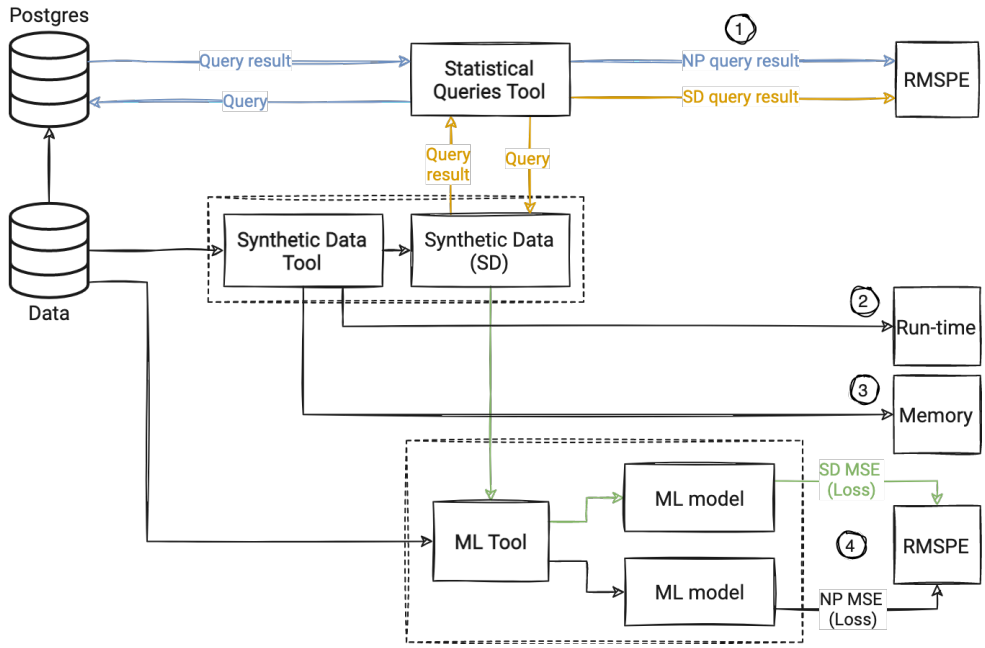

Figure 4.4: Overview of experiments flow for statistical query tools, showing how results are obtained, by comparing results obtained from the synthetic vs the original data, for *1)* data utility of statistical queries, *2)* run-time overhead, *3)* memory overhead, and *4)* data utility of ML tasks.

Synthetic data generation (SDG) offers a way of using substitute data, rather than the raw version in data-related services, to preserve privacy. However, SDG does

not provide any formal privacy guarantees, and information about individual data might thus be disclosed [51, 52, 53]. Differentially private (DP) SDG on the other hand provides formal guarantees for individual privacy. However, while it aims to retain as many statistical properties of the original dataset as possible, conducting statistical queries and ML tasks on DP synthetic datasets might significantly impact the accuracy of the results, since DP comes with an inherited impact on utility.

To determine the impact that the provision of different DP-SDG tools has on data utility, we generate a synthetic version of two different datasets, i.e., *Health Survey*, *Parkinson* (Section 4.2). Since data size and ϵ impact the outcome of DP mechanisms the most, we generate a synthetic dataset with different combinations of the two parameters. We then measure how much the results of ML tasks and statistical queries on synthetic datasets differ from those conducted on the original ones. The definition and calculation of data utility and system overhead are shown in Section 4.1.

With the experiments listed in List 4.3.3, we aim at gaining insights into how ML tasks and statistical queries can be affected when they are conducted on the synthetic dataset, and how the considered synthetic data tools trade-off accuracy and privacy through the experimental results. We are also interested in how the performance of utility correlates with ϵ and data size in SDG.

For clarity on how we conduct our experiments, we present a high-level overview of the experiments flow in Figure 4.4. It shows how results are obtained from the SDG experiments. Specifically, it shows how data utility of statistical queries and ML tasks is measured by comparing results obtained from the synthetic data and original data, along with the absolute run-time and memory overhead. Below we show the detailed ingredients for this part of the experiment.

We would like to remind you that we evaluate the following synthetic data tools: Smartnoise Synthetic, Gretel Synthetics (Section 1.7).

Experiment list for synthetic data tools

7. *Statistical query utility*

   - We carry out experiments to show how the performance of statistical queries on synthetic datasets differ from those conducted on the original dataset. We conduct a set of queries qs (Table 4.4) to show how the performance vary between results obtained from the synthetic vs the original datasets.
   - To show the impact that the SDG tools have on data utility we generate synthetic version, for different ϵ (Table 4.3), of
     (a) Health Survey of dataset size in Table 4.2
     (b) Parkinson of dataset size in Table 4.2

- We study the impact on data utility by measuring the RMSPE (Equation 4.1) between query results obtained from the synthetic dataset vs the original dataset.

8. *Machine learning utility*

   - Experiments in this part aim to show how the performance of ML tasks on synthetic datasets differ from those conducted on the original dataset. We specify the tasks as linear regression (LR) to show how the performance vary between results obtained from the synthetic and the original datasets.
   - To show the impact that the SDG tools have on data utility, we generate synthetic version, for different ϵ (Table 4.3), of
     (a) Health Survey of dataset size in Table 4.2
     (b) Parkinson of dataset size in Table 4.2
   - We study the impact on data utility by measuring the RMSPE (Equation 4.1) between model performance (loss) obtained from the synthetic dataset and the original dataset.

9. *Run-time overhead*

   - Experiments are conducted to gain insights into the run-time overhead for generation synthetic datasets with the leveraged tools.
   - We generate synthetic dataset, for different ϵ (Table 4.3), of
     (a) Health Survey of dataset size in Table 4.2
     (b) Parkinson of dataset size in Table 4.2
   - We study the impact on run-time overhead by measuring the direct time usage of synthetic data generation as shown in Algorithm 3.

10. *Memory overhead*

    - We conduct experiments aiming to gain insights into the memory overhead for generation synthetic datasets with the leveraged tools.
    - We generate synthetic dataset, for different ϵ (Table 4.3), of
      (a) Health Survey of dataset size in Table 4.2
      (b) Parkinson of dataset size in Table 4.2
    - We study the impact on memory overhead by measuring the direct time usage of synthetic data generation.

4.3.4 Experiment Settings

In our evaluating experiments, each dataset comes with a metadata file that contains the experiments parameters and experiment settings (Section 4.3.4). The metadata files contain instructions and settings for each domain of tools and list what parameters to use when conducting different experiments. This is further elaborated in Section 4.4.

4.4 Experiment Implementation

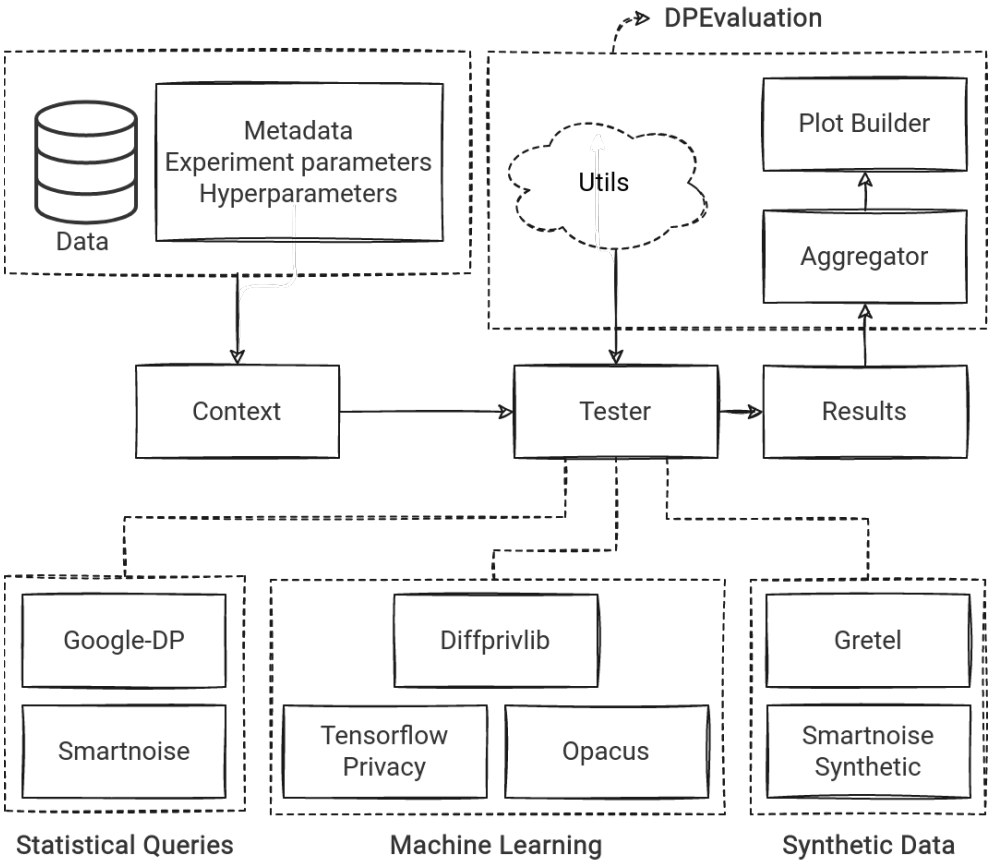

Figure 4.5: High-level overview of experiment implementation, showing how data flows in the evaluation framework.

We aim at enabling the reuse of our framework to the full extent and allowing users to develop and test our code on any environment, without installing and handling dependencies locally. We, therefore, build a collective framework for all the tools, with a focus on usability and portability. The full source code can be found here: `https://github.com/anthager/dp-evaluation`.

To make sure that our code works across different environments, our framework is developed with Docker that packages each tool `package` and its dependencies in a virtual container. Docker also eases memory usage measurements. The Docker daemon exposes a RESTful API, running on the host system through a UNIX socket, from which metrics such as memory usage can be fetched.

Each tool that is being evaluated is packaged in its own docker image together with all the packages the tool depends on, together 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.

To visualise how the framework is constructed, we present a high-level diagram in Figure 4.5. 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) Results are saved after running `Tester` that (5) `Plot Builder` collects with `Aggregator`, both which are parts of the collective utils package.

4.4.1 Data pre-processing, Metadata and Context

To make sure that datasets (Table 4.1) fit certain criteria since the datasets are of different shapes and attributes, we *pre-process the data* by conducting quality control and parsing the data. Moreover, to be able to compare results, we want to use each dataset for all the leveraged tools. Since ML tasks and the synthetic data generators require encoded data, we encode categorical and ordinal columns as strings or integers, as well as continuous columns as floats according to the tools' requirement on the data format.

To enable backward translation for encoded attributes, a *metadata* file is generated for each dataset, containing all column names, their respective encoding, numerical lower and upper bounds, etc. It also contains test parameters, i.e., ϵ values (Table 4.3), dataset sizes (Table 4.2), and hyperparameters for ML tasks and synthetic data generators (Section 4.3.4), which makes it convenient for setting up different parameters for different experiments. The metadata and its corresponding dataset are stored in `Context`, which provides functionalities such as parsing, splitting, or saving the dataset throughout the evaluation process.

4.4.2 Testers

Each tool is assigned with a `Tester` script that, given a `Context` (containing metadata, experiment parameters, and hyperparameters), will run a test for each combination of the configuration parameters, and produce results in CSV format. E.g., for statistical query tools, one test is running one differentially private query on the dataset one time. For ML tools, one test is conducting one ML task on the dataset one time. A full set of tests, given different combinations of ϵ values and data size, correspond to one experiment. In both these cases, we write the test results and system overhead to a CSV file, together with key points of its configuration (see details in Appendix A.3).

4.4.3 DPEvaluation

Since a lot of code can be reused between the testers (Section 4.4.2), it was placed in a package `DPEvaluation` that holds common utils for statistical query tools, ML tools, and synthetic data tools. It includes functionalities for logging events, reading, parsing, and saving data, building metadata files, parsing datasets, bootstrapping PostgreSQL, building ML models, splitting and normalizing datasets for

ML tasks, etc. Besides mentioned functionalities, `DPEvaluation` contains aggregators and plotters for merging, extracting, and plotting results obtained from the DP tools.

4.4.3.1 Aggregators and Plotters

To evaluate the results from our experiments (Section 4.3), we use `Aggregator` scripts to merge the results and compute the RMSPE (Section 4.1). To get insights from how the tools perform, we use *Plotter* scripts to efficiently quantify and visualize our results. I.e., the aggregators are responsible for merging, querying, and extracting insights and data-series from the results, while the plotters process the data-series and generate various plots.

5
Results

This chapter shows our evaluation results of all the considered tools. We illustrate how relevant open-source privacy tools perform in differential privacy (DP) applications. The results are presented with complementary analysis to show what we can learn from them. I.e., insight into the performance of the considered tools, and how the obtained knowledge can benefit the use of DP tools in practice.

This chapter provides insights for readers within developer communities, who are seeking guidelines for selecting and configuring DP tools in their privacy service development. The evaluation results can also provide the academic society with a quantitative assessment of the effectiveness of DP tools, which can ultimately serve as an important role in bridging the gap between theoretical and practical DP research. As detailed below, we obtain results showing *(1)* how much DP tools impact data utility and system overhead in DP applications, and *(2)* if DP service practitioners can apply these tools, and how they can be configured.

5.1 Summary of results

Overall, our evaluation results indicate a positive relationship between high ϵ values and high data utility, as well as large data size and high data utility. However, irregularities appear in most cases, and some tools perform inadequately under certain settings, which might be interesting to notice for practical implementations in industry and warrants further research. Moreover, it seems that most tools perform worse on continuous data than on categorical data. Specifically, Diffprivlib, Gretel Synthetics, and Smartnoise MWEM did not manage to generate any usable results with our continuous data. It is also worth noting that memory overhead is not affected significantly across all tools when compared with the non-privacy-protected results.

The results are presented as the difference in percentage error between differentially private and non-privacy protected results obtained by the tools. The evaluation criteria are defined in Section 4.1, and the list of experiments are shown in Sections 1, 4.3.2 and 4.3.3. Details of the experiments are presented in Sections 4.3, and detailed results are shown in Sections 5.3, 5.4 and 5.5. In the following Sections 5.1.1, 5.1.2 and 5.1.3, we shortly summarize what we found from each experiment before the detailed experimental results.

5.1.1 Summary of statistical query tools results

For the statistical query tools, we observe the expected correlation between ϵ, data size, and data utility. Smartnoise has a substantial increase in run-time overhead ($\approx$ 450%) when DP is applied, across all ϵ values and data size on both datasets (*Health Survey* and *Parkinson*), while the impact on run-time is less for Google DP ($\approx$ 150%). In comparison, memory overhead is marginally increased by the tools, while the increase is negligible in the database engine that holds the data, which the queries are conducted.

5.1.1.1 Experiment 1. Data utility

In experiment 1, we can observe that, when ϵ and data size increase, the queries provide more accurate answers for both query tools. For the simple queries, such as `COUNT`, `SUM` and `AVG`, Google DP performs between 0.1% and 20% worse than the benchmark, while Smartnoise performs between 0.2% and 350% worse than the benchmark over the considered parameter ranges of ϵ and data size. On the other hand, Smartnoise performs better than Google DP on the `HISTOGRAM` queries, i.e., between 0.5% and 60% worse than benchmark, compared to Google DP's between 0.2% and 250%. Looking only at the full data size (9358 rows for *Health Survey* and 5499 for *Parkinson*), and ϵ value of 0.5, Google DP has a maximum error of 1%, while Smartnoise has a maximum error of 2%, which together with mentioned results indicate that Google DP performs better than Smartnoise on the simple queries.

As for complex query, we in the experiment could not see any obvious relationship between ϵ, data size, and accuracy for `HISTOGRAM` queries on the *Health Survey* for Google DP, since the results fluctuate significantly. The fluctuation is not as extreme for Smartnoise, which implies that Smartnoise performs better on `HISTOGRAM` queries than Google DP.

5.1.1.2 Experiment 2. Run-time overhead

In experiment 2, we can observe that the run-time overhead increases for larger datasets when using Smartnoise. Given the smallest data sizes, Smartnoise poses an extra run-time overhead of about 400% to 500%. For the largest data sizes, Smartnoise adds a run-time overhead of about 450% to 550%. Google DP on the other hand induces less run-time overhead when using larger data sizes. Given the smallest data sizes, Google DP adds a run-time overhead of around 110% to 150%, and between about 100% and 130% for the largest data sizes. This means that Smartnoise adds significantly more overhead than Google DP does.

In this experiment, we did not notice any obvious relationship between ϵ and the run-time overhead, and no significant difference between the various queries. There are however apparent fluctuations in the results.

5.1.1.3 Experiment 3. Memory overhead

In experiment 3, the DP tools inflict a small memory overhead in the database less than about 5% for both datasets. For the larger data sizes, Google DP has an overhead between about -1% and 2%, while Smartnoise has between about -2% and 3%, which makes it hard to say which of the tools has less overhead. Specifically, for the *Parkinson* dataset, the memory overhead of the database fluctuates between 0% to 3% for the `HISTOGRAM` query, while between 0% to 1% for the other queries. For the *Health Survey* dataset, all queries fluctuate between about -2% and 3%. These results show that the `HISTOGRAM` query performs worse than the other queries on the *Parkinson* dataset while no such point can be made from the *Health survey* dataset. Moreover, the process that runs the evaluation, i.e., the impact of DP statistical query tools, has a memory overhead between about -30% and 40% for both of the datasets which shows that there is a significantly larger memory overhead in the process running the evaluation than in the database.

In this experiment, we cannot find any relationship between ϵ, data size, and memory overhead. The data fluctuates significantly, but we cannot see any obvious patterns. For *Health Survey*, some queries, e.g., `HISTOGRAM`, use more memory than others, e.g., `SUM`, i.e., about 40% compared to 0%, for the same combination of data sizes and ϵ values. However, what query has a larger overhead differs largely, without any clear patterns.

5.1.2 Summary of machine learning tools results

For the ML tools, experiment 4 indicates that Opacus overall performs marginally better on categorical data (*Health Survey*), in comparison to Tensorflow Privacy and Diffprivlib. However, Diffprivlib achieves better accuracy given high ϵ values and large data size on categorical data, while it obtains undesirable accuracy for low ϵ values and data size. Diffprivlib is also notably faster in absolute time but did not manage to generate any useful DP results on continuous data.

5.1.2.1 Experiment 4. Data utility

In experiment 4, we can observe a trend between high ϵ values, large data size, and high data utility. When we take certain privacy budgets to compare the tools, e.g., for $\epsilon = 0.5$ with larger data sizes of *Health Survey* ($\gtrapprox 9000$), Opacus has an error around 6%, Tensorflow Privacy (TFP) between about 9% and 16%, and Diffprivlib between about 12% and $3.3\dot{1}0^8$%. For the same dataset and data size but $\epsilon = 3$, Opacus has an error between about 3% and 5%, TFP between about 3% and 6%, and Diffprivlib between about 0.5% and 2%. This shows that while Diffprivlib provides good results for larger ϵ values, there is a risk that it will perform undesirably poorly for smaller ϵ values while both Opacus and TFP performs well on those ϵ values. Moreover, on the *Parkinson* dataset, Opacus has roughly about twice the RMSPE as TFP given larger ϵ values ($\gtrapprox 1.5$) and for data sizes of ($\gtrapprox 5000$). Diffprivlib on the other hand has more than a million times the RMSPE, which makes its results impractical on continuous data (*Parkinson*).

5.1.2.2 Experiment 5. Run-time overhead

In experiment 5, given the larger data sizes ($\gtrapprox$ 5000 for *Parkinson* and $\gtrapprox$ 9000 for *Health Survey*), Tensorflow Privacy (TFP) has a run-time overhead of between about 25% and 50%, and Opacus has a run-time overhead of between about 200% and 250%. Diffprivlib on the other hand has a run-time overhead between about 500% and 2000% on *Health Survey* and no useful result on *Parkinson* data, which makes it the slowest compared to benchmark.

Beyond that, we cannot find any clear patterns between ϵ, data size, and run-time overhead for either of the tools or datasets. Moreover, the run-time overhead for Diffprivlib is excluded altogether, since it generated unusable data utility results, as stated in experiment 4.

5.1.2.3 Experiment 6. Memory overhead

In experiment 6 we can observe that TFP has a significant memory overhead, about 90% additional memory consumption for the *Parkinson* dataset and about 20% for the *Health Survey* dataset. Opacus has a marginal memory overhead, less than 5% in both of them. For the *Health Survey* dataset, Diffprivlib has an overhead of about 10% while Diffprivlib are excluded from the *Parkinson* dataset, since it generated unusable data utility results, as stated in experiment 4.

We cannot see any connections regarding the value of ϵ, dataset size, and the memory overhead that would indicate any trends. The memory overhead is fairly consistent for each combination of tool and dataset and independent of the value of ϵ and dataset size. There are however fluctuations against the evenly distributed memory overhead in the results, e.g. both TFP and Diffprivlib observe an abrupt decrease of memory overhead for dataset size of 9358 for the *Health Survey* dataset.

5.1.3 Summary of synthetic data tools results

For the evaluation of synthetic data tools, we observe considerably larger errors when conducting statistical queries and ML tasks on the synthetic data, compared to when DP is applied to the statistical query or ML tools. There are a few examples where analysis on the synthetic datasets provides a reduced accuracy of 20%, compared to benchmark, however, sometimes this reduction is well over 100%. The tools for DP statistical queries and DP ML, on the other hand, frequently obtain an accuracy decrease of less than 10%. Particularly, the data synthesizer Smartnoise PATECTGAN performs marginally better than the other tools on continuous data (*Parkinson*), while Smartnoise DPCTGAN outperforms the other tools on categorical data (*Health Survey*). Additionally, Smartnoise MWEM is by far the fastest but did not manage to generate any useful results on continuous data.

Moreover, as detailed in Section 5.5, there are a few missing data points in these experiments. For instance, we were unable to retrieve any synthetic data for the *Parkinson* dataset with Smartnoise MWEM and Gretel Synthetics. We also did not

obtain any data given some combinations of ϵ values and data sizes with Smartnoise MWEM on *Health Survey*, and Smartnoise DPCTGAN on *Parkinson*.

5.1.3.1 Experiment 7. Statistical query utility

In this experiment, we observe that Smartnoise PATECTGAN obtains better data utility for smaller data size (RMSPE between about 25% to 70%, given data size of 1000, while between about 50% and 70% for data size of 9358), and generally better data utility for larger ϵ values (RMSPE between about 60% and 80%, given $\epsilon = 0.1$, compared with between about 30% and 65% for $\epsilon = 3.0$). In other words, there is a connection between ϵ, data size, and utility for Smartnoise PATECTGAN.

For the `AVG` query on the *Health Survey*, Smartnoise DPCTGAN has an RMSPE between about 20% and 70%, Smartnoise PATECTGAN between about 35% and 80%, Smartnoise MWEM between about 50% and 130%, and Gretel Synthetics between about 20% and 30%. In comparison, for the `AVG` query on *Parkinson*, the data points that we obtained with Smartnoise DPCTGAN have an RMSPE between about 80% and 100%, while Smartnoise PATECTGAN has an RMSPE between about 60% and 80%. This means that DPCTGAN performs better on *Health Survey*, and PATECTGAN performs better on *Parkinson*, while Gretel performs the best of the tools on *Health Survey*. However, it should be noted that Gretel uses larger values of ϵ (See Table 5.1), since its privacy budget could not be set manually. All of these accuracy reductions are very high and it might be hard to find a scenario where such decreased accuracy would be acceptable in statistical analysis.

5.1.3.2 Experiment 8. Machine learning utility

In this experiment, we cannot find any trends regarding data size and ϵ on *Health Survey*. We can however observe a general trend on *Parkinson*, where larger data size provides better data utility. For instance, Smartnoise PATECTGAN has an RMSPE between about 800% and 1200% for data size of 5499, and between about 2500% and 4000% for data size of 1000. On the other hand, the tools overall perform worse on *Parkinson* than on *Health Survey*. The best results on *Health Survey* across all tools produced an RMSPE of about 20% and about 900% on *Parkinson*. Similar to experiment 7, experiment 8 shows that using synthetic data negatively affects the utility of the analysis to such an extent that the analysis should be considered useless.

5.1.3.3 Experiment 9. Run-time overhead

This experiment shows that Smartnoise PATECTGAN and DPCTGAN perform very similarly, given $\epsilon < 3$. Both tools also generate synthetic datasets in about 5 seconds, given both the *Parkinson* and the *Health Survey* datasets. Moreover, when $\epsilon = 3.0$, it takes a longer time to generate synthetic datasets given larger data sizes. For smaller data size, it takes about 5 seconds for both synthesizers (PATECTGAN and DPCTGAN), given both datasets (*Parkinson* and *Health Survey*).

Furthermore, given the *Health Survey* dataset with a data size of 9358, it takes about 200 and 50 seconds for DPCTGAN and PATECTGAN respectively, and about 100 and 50 seconds for DPCTGAN and PATECTGAN respectively given the *Parkinson* dataset with data size of 5499.

Moreover, Smartnoise MWEM generates synthetic datasets between about 150 and 700 seconds, while Gretel Synthetics consumes between about 400 and over 2000 seconds, which can be seen in Figure 5.19c. This means that Gretel Synthetics and Smartnoise MWEM are more time-consuming in general than the other tools.

5.1.3.4 Experiment 10. Memory overhead

In this experiment, we cannot observe any significant difference between Smartnoise MWEM and Smartnoise PATECTGAN on *Health Survey* and Smartnoise PATECTGAN on *Parkinson*. They have approximately consistent memory consumption of about 200MB. Smartnoise DPCTGAN on the other hand uses between about 200MB and 1000MB of memory, and for Smartnoise PATECTGAN we can observe that higher values of ϵ and larger data size typically result in larger memory consumption. Additionally, Gretel Synthetics is consistently using about 1700MB of memory, which is the highest of the tools.

5.2 Answers to key research questions

By conducting our evaluation we provide answers to our key research questions of *(1)* how much DP tools impact data utility and system overhead in DP applications, and *(2)* if DP service practitioners can apply these tools, and how they can be configured. We generate some guidelines through our evaluation results that might benefit developers' experience with differential privacy tools in the following.

Developers looking at accumulating general statistics about categorical or continuous datasets, with a margin of error from about 0.1% to 2% for simple queries (`SUM`, `COUNT`, `AVG`), with $\epsilon \gtrapprox 1.5$, should consider Google DP. It should be noted that this suggestion assumes that the data is stored in a Postgres database. For integration with other database engines, given the same set of queries and ϵ values, Smartnoise is preferable as able to provide an accuracy of about 0.5% to 5%. Smartnoise also offers better accuracy for `HISTOGRAM` queries, below about 10% compared to Google DP's with about 15% on *Health Survey*.

Furthermore, in the case where run-time overhead is crucial, we would like to note that Google DP is significantly faster than Smartnoise ($\approx 450\%$). However, when it comes to memory consummation, we cannot make any certain suggestions since we measure the memory usage of the Docker container that hosts the database every second, and other processes might have a large impact on to the results during measurement. This is true for all tools and should serve only as an indication of approximate memory overhead tendencies that each tool has.

For developers looking at building DP machine learning models, both Opacus and Tensorflow Privacy (TFP) show promising results, obtaining data utility below 6% error for $\epsilon \geq 1.0$ on both datasets. However, TFP manages to obtain around two times better data utility than Opacus given maximum data size and $\epsilon = 3.0$. Diffprivlib on the other hand outperforms both tools on *Health Survey*, given maximum data size and $\epsilon = 3.0$. It should however be noted that Diffprivlib did not manage to generate any useful results on continuous data and is also limited to linear regression and logistic regression models, while TFP and Opacus offer building custom neural networks, allowing developers to build complex models for a variety of problems.

While Diffprivlib is orders of magnitude faster in absolute time, both with and without DP applied, it comes with the limitations mentioned above. Meanwhile, given that Opacus and TFP utilize the same approach for applying DP during the machine learning process, it is worth noting that Opacus impact on run-time overhead is around 200% worse than TFP. With that said, Opacus consumes noticeably less memory, below around 4% more than the benchmark, across all ϵ values and data sizes for both datasets. In comparison, TFP consumes around 20% more on *Health Survey* and around 80% more on *Parkinson*. However, it should be noted that it drops to about 40% on *Parkinson* and to around 2% on *Health Survey* for maximum data size and $\epsilon = 3.0$.

For developers looking at DP synthetic data generation, our results indicate that, given our data sources, these tools do not provide nearly as good accuracy as the alternatives, which integrate DP into statistical queries and ML tasks themselves. I.e., while DP synthetic data tools might offer flexibility, given that the whole synthesized dataset could potentially be disclosed, for conducting statistical query and ML tasks on, our synthetic datasets in the evaluation produced undesirable data utility. With that said, the potential advantage of future techniques in the field, allowing a non-restricted number of queries on synthetic datasets, and providing DP datasets for ML tasks, build a valid case for continuous development in the field. It should also be noted that this is an emerging field, and the tools we used for this evaluation are only months old.

With our evaluation results, we suggest that the answer to if developers should use DP depends on primarily *(1)* their use-case and *(2)* how accurate results they need to obtain. For the question about how to configure the tools, our results show that this is not trivial since it is highly dependent on both the data, the tool being used, and the use case. The results from our evaluation can serve as guidelines for how the value of ϵ can be configured for different data sizes, and what data utility can be expected in those circumstances.

We detail our evaluation results in the following sections.

5.3 Experiment results of statistical query tools

Non-privacy protected (NP) statistical queries might jeopardize the privacy of individuals within a dataset. Differential privacy (DP) can be integrated into statistical query tools to address this issue and provide formal guarantees for individual privacy. However, applying DP to statistical queries might reduce the accuracy of the query results [9], and increase the system resources needed to conduct the analysis.

Experiment 1, 2, and 3 in this section evaluate the considered statistical query tools using criteria of utility and overhead. The definition and calculation of data utility and overhead are shown in Section 4.1. The organization of this section is as follows: the experiment in Section 5.3.1 studies the impact that statistical query tools have on data utility when conducting DP queries. Section 5.3.2 investigates the impact that statistical query tools have on the run-time when DP is applied, while the experiment in Section 5.3.3 studies their impact on memory overhead. We focus on the impact of DP by measuring how the performance of DP queries differs from NP queries. Specifically, for experiment 3 (Section 5.3.3), we study the impact on memory overhead, which is imposed by *(1)* the statistical query tools, and *(2)* the memory usage in the database, holding the dataset that we conduct queries on. Memory overhead is measured by comparing the worst-case memory usage between DP and NP query or ML tasks and shows the percentile difference, denoted as δ (Section 4.1). Moreover, statistical queries are conducted on a Postgres server, which is compatible with the considered tools. Also note that the frequency by which memory usage was recorded was imposed by the Docker containers, and was recorded once every second.

Experiment results are obtained by conducting queries (Table 4.4) on each column in two different datasets, i.e., *Health Survey*, *Parkinson* (Section 4.2). The privacy budget parameter (ϵ) and the size of the dataset are the parameters that influence the result the most [4, 18, 58], thus the queries (Table 4.4) are conducted using different combinations of ϵ (Table 4.3) and data size (Table 4.2) to see how the results vary under different conditions. Note that `HISTOGRAM` queries are only conducted on categorical columns since only the statistics of the categorical values can be put into buckets to generate the results for `HISTOGRAM`.

In the experiment, each query runs for 20 times, and we present the result using the average of 18 moderate queries. That is, the top and bottom extrema results are removed out of the 20 query results in order to not include outliers, and the remaining 18 are averaged to generate the results to display and analyze. The list of experiments for statistical query tools is found in Section 1. Experiment settings in regard with query types, ϵ, and data size are listed in Table A.1.

The evaluated DP statistical query tools in this section include Google Differential Privacy and Smartnoise, whose performances are shown in the following.

5.3.1 Experiment 1. Data utility

Integration of differential privacy (DP) in statistical query tools comes with an inherent impact on utility. In experiment 1, we study that impact by comparing how DP query results differ from non-privacy protected (NP) ones. The experiments also study how this difference varies between different statistical query tools.

For this experiment, we anticipate that higher ϵ and the larger data size will provide better utility for both datasets since less noise is expected to be injected under such conditions. As we detail below, from results in experiment 1, we can observe that when ϵ and data size increase, the queries perform better. I.e., larger ϵ and larger data size provide better utility. Beyond this expected trend, we notice that there exist some local irregularities that break the trend, specifically for `HISTOGRAM` queries.

In Figure 5.1, we can observe that higher ϵ values decrease the RMSPE (Section 4.1), which implies higher utility of DP queries. I.e., higher ϵ values improve accuracy for all queries (Table 4.4) in both considered datasets. This observation is quite explicit, especially for the performance of DP queries on *Parkinson*. As anticipated, the relationship that data utility grows with the increase in data size can also be observed. Generally, the results on *Parkinson* are more consistent with our anticipation, while those on *Health Survey* data show marginal levels of fluctuation, specifically for `HISTOGRAM` queries, which we cannot explain.

Figure 5.2 shows contour plots [85] for each tool's performance in regard to different ϵ and data size. Darker shades of blue in the plots indicate a lower RMSPE, corresponding to higher utility. Lighter shades indicate a larger RMSPE, corresponding to lower utility. Note that the plots have different RMSPE scales. This implies that a shade in one plot (which indicates an RMSPE value), does not necessarily correspond to the same shade in another plot.

In Figure 5.2, we can observe that higher ϵ values lead to lower RMSPE values, which implies higher utility of the DP queries. This trend is apparent for both datasets, especially for the performance of DP queries conducted by Google DP on *Health Survey* (Figures 5.2a, 5.2b, 5.2c, 5.2d), showing between around 0.1% and 20% error for $\epsilon \gtrapprox 2$. Moreover, looking at the relationship between datasets' size and utility, we oberve that the results on both datasets are adequately consistent with our anticipation. Broadly speaking, we can observe direct correlational trends between data size, utility and ϵ. However, `HISTOGRAM` queries show a notable level of fluctuation (Figure 5.2d, 5.2h, 5.2l, 5.2p), cf. Google DP *Health Survey* (Figures 5.1a, 5.1b, 5.1c, 5.1d), which is surprising and warrants further investigation in the future. The results indicate that Smartnoise performs between 0.5% and 60% worse than the benchmark, compared to Google DP's between about 0.2% and 250%, looking at the `HISTOGRAM` query.

Queries conducted on *Health Survey* (Figures 5.2a, 5.2b, 5.2c, 5.2d, 5.2e, 5.2f, 5.2g, 5.2h) obtain, in general, better accuracy than the corresponding query results on

Parkinson (Figures 5.2i, 5.2j, 5.2k, 5.2l, 5.2m, 5.2n, 5.2o, 5.2p). However, for all queries conducted by Smartnoise on *Parkinson*, we notice a considerable decrease in accuracy in comparison to Google DP (Figures 5.2m, 5.2n, 5.2o, 5.2p).

Overall, `HISTOGRAM` queries have a larger impact on data utility compared with other types of queries for the considered statistical query tools. This is expected, and consistent with the inherent privacy-utility trade-off in DP, since `HISTOGRAM` queries expose more information about the dataset properties. Consequently, more noise is injected into the `HISTOGRAM` results to guarantee the privacy of individuals, which in turn reduces more data utility. Moreover, `HISTOGRAM` queries obtain better results on *Parkinson* than on *Health Survey*. This is also expected, since the categorical columns in *Parkinson* have fewer bins than *Health Survey* (Section 4.2), and thus expose less information about *Parkinson* dataset properties. Consequently, less noise is injected into the results on *Parkinson* dataset in pursuing the privacy of individuals.

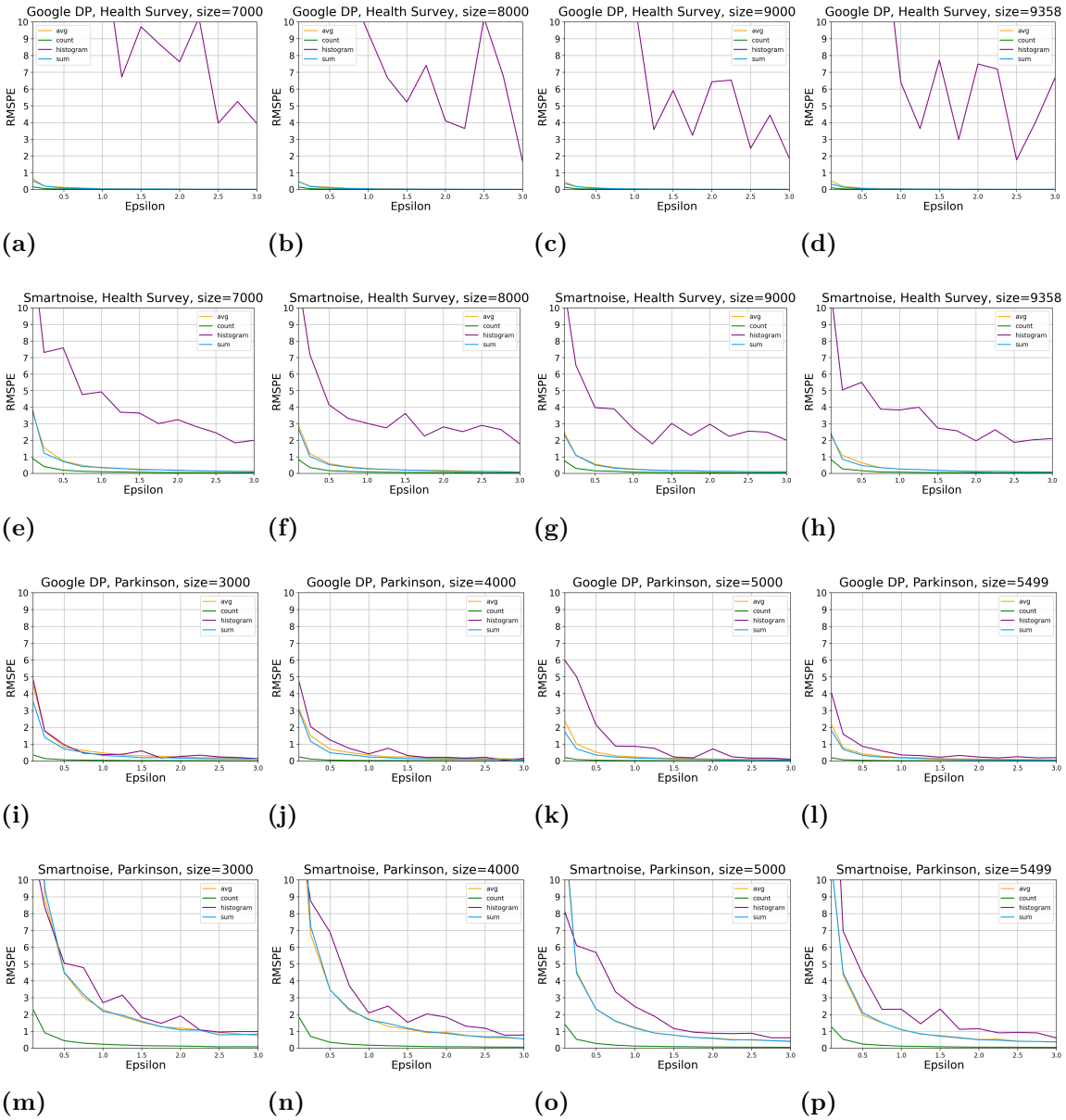

Figure 5.1: The results of experiment 1, showing the impact of statistical query tools on utility for different ϵ (Table 4.3), data size (Table 4.2) and queries (Table 4.4). RMSPE is defined in Section 4.1.

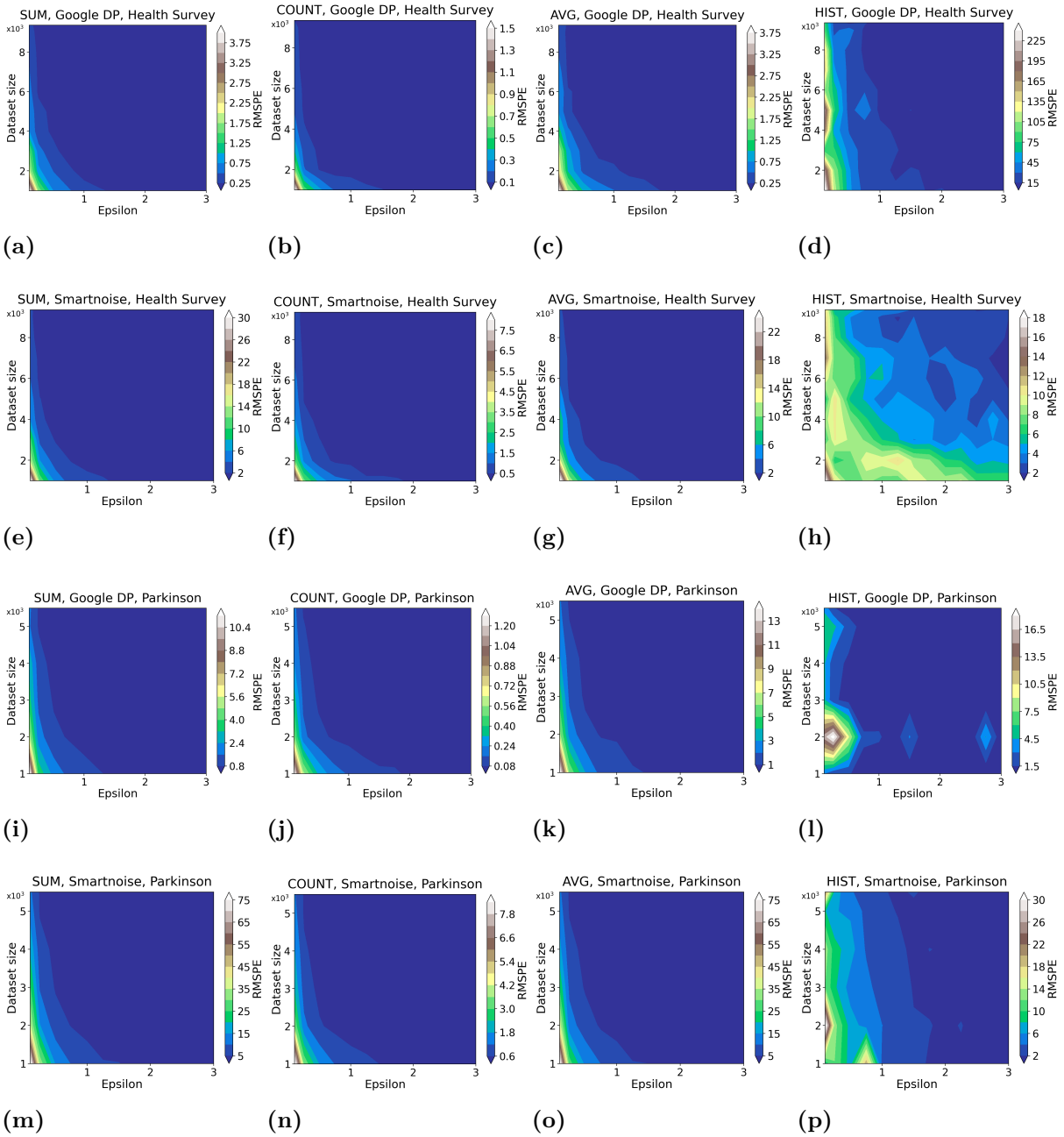

Figure 5.2: The results of experiment 1, showing the impact of statistical query tools on utility when DP is integrated, for different ϵ (Table 4.3), data size (Table 4.2) and queries (Table 4.4). RMSPE is defined in Section 4.1.

5.3.2 Experiment 2. Run-time overhead

Integration of differential privacy in statistical query tools comes with an inherent impact on run-time since differential privacy (DP) requires additional computation to conduct the query. In experiment 2, we study the impact by comparing how the run-time overhead (Section 4.1) differs between DP and non-privacy protected (NP) query tasks. We also study how this difference varies between different statistical query tools.

For this experiment, we anticipate that the run-time overhead might, in general, increase when DP is integrated since DP requires additional computations to conduct the query. Moreover, since the run-time typically increases with data size, we also expect to observe a direct connection between data size and the run-time overhead. As detailed below, in experiment 2, we can generally observe that Smartnoise has a larger impact on the run-time overhead than Google DP, between about 400% to 550% compared to about 100% to 150% respectively. We notice that Smartnoise follows the anticipated trend, where a larger data size increases the run-time overhead. In the contrast, data size has the opposite effect on Google DP, where larger data sizes generally have less run-time overhead.

Figure 5.3 shows contour plots for each tool's run-time overhead in regards to ϵ and data size. Darker shades of blue in the plots indicate a lower RMSPE, corresponding to lower run-time overhead. Lighter shades indicate larger RMSPE, which corresponds to a larger run-time overhead. Note that the plots have different RMSPE scales, which implies that a shade in one plot does not necessarily correspond to the same shade in another plot.

In Figure 5.4, we can observe that higher ϵ values do not necessarily decrease the RMSPE (Section 4.1), which implies that higher ϵ values do not generally impact the run-time of DP queries. This observation holds for both tools and datasets and is quite explicit, especially for Google DP on *Health Survey*. However, we also notice that there exists some local fluctuations that break this trend, specifically for `HISTOGRAM` queries (Figures 5.4j, 5.4n), which is also apparent for `AVG` queries (Figures 5.4f, 5.4g). We cannot explain this behavior.

When it comes to the relationship between data size and the run-time, we notice that larger data sizes typically increase the RMSPE for Smartnoise, i.e., between about 400% to 500% for the smallest data sizes, and about 450% to 550% for the largest. However, larger data sizes usually decrease the RMSPE for Google DP, between about 110% to 150% for the smallest, and about 100% to 130% for the largest. This trend is apparent for both datasets and both tools, especially on the *Health Survey* data, and can be observed more clearly in Figure 5.3. It implies that Smartnoise is running notably slower for large data sizes, while Google DP runs marginally faster as the data size increases. Generally, Smartnoise's increase in run-time is more consistent with our expectation, which is that run-time typically increases as the data size grows. Google DP's decrease in run-time for large data sizes is therefore unexpected, which is an interesting result that warrants further

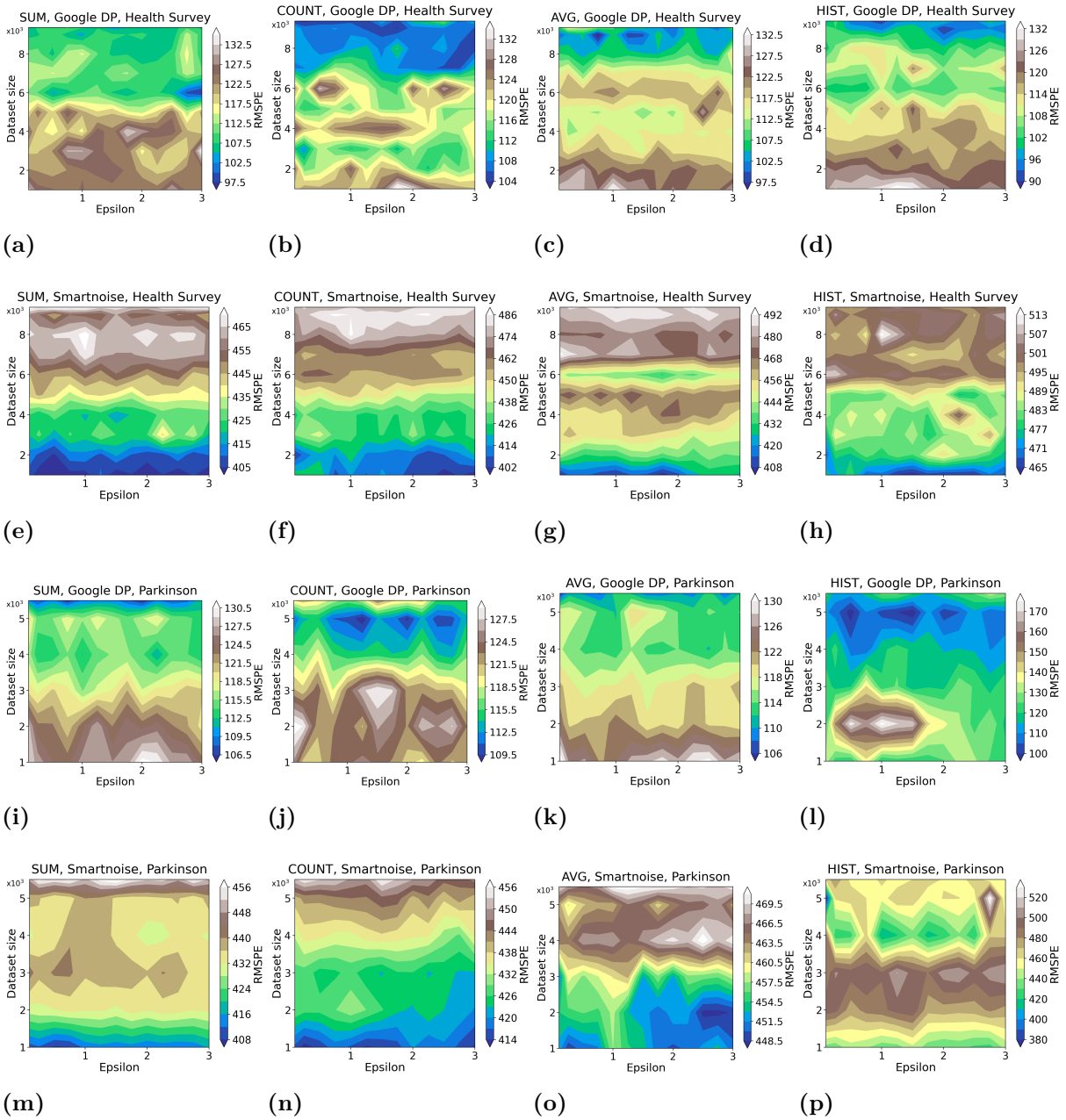

Figure 5.3: The results of experiment 2, showing the impact of statistical query tools on run-time overhead, for different ϵ (Table 4.3), data size (Table 4.2) and queries (Table 4.4). RMSPE is defined in Section 4.1.

study. However, we notice that there exist irregularities that break these trends, e.g., Figure 5.3l, which we cannot explain.

Overall, Google DP outperforms Smartnoise that runs around 400% to 500% slower when conducting DP queries, compared to Google DP that runs around 100% slower. One reason might be that Google DP performs more efficient DP calculations, using a plugin inside the database that is written in C and compiled to native code (Sec-

tion 3.1.2). The run-time might therefore improve, since C is a high-performance language, and no additional layer operates between the database and the analyst that conducts the queries. In comparison, Smartnoise implements pre-processing of queries in Python, before communicating with the database (Section 3.1.1), which might have an impact on the run-time since Python is a high-level language and performs noticeably slower than C.

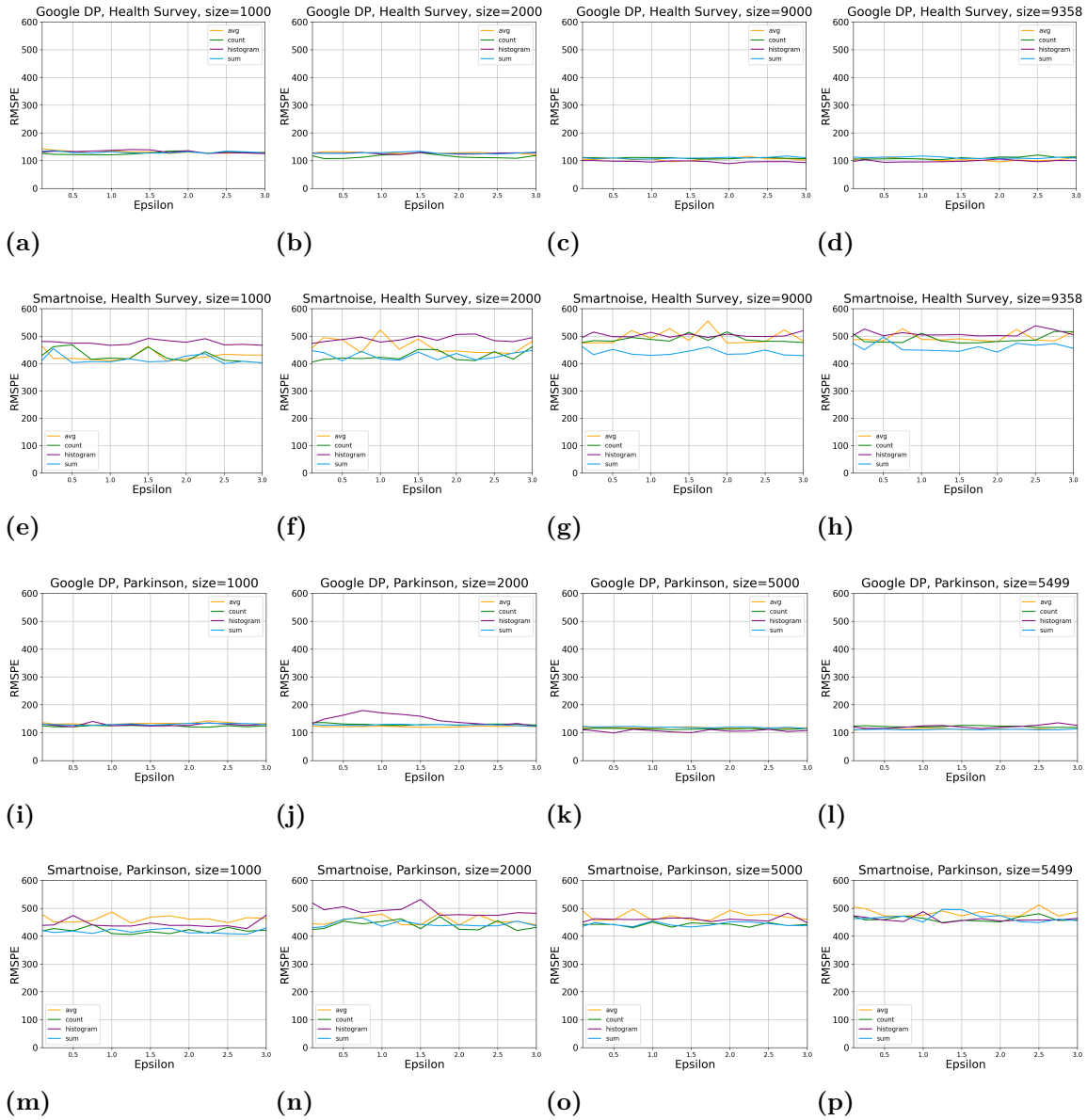

Figure 5.4: The results of experiment 2, showing the impact of statistical query tools on run-time overhead when DP is integrated, for different ϵ (Table 4.3), data size (Table 4.2) and queries (Table 4.4). RMSPE is defined in Section 4.1.

5.3.3 Experiment 3. Memory overhead

Integration of differential privacy (DP) in statistical query tools might come with an impact on memory overhead. In experiment 3, we study that impact by comparing how memory usage differs between DP and non-privacy protected (NP) statistical query tasks. The experiments also study how this difference varies between different statistical query tools.

For this experiment, additional memory overhead is anticipated to be slightly increased when DP is integrated into statistical query tasks. As detailed below, in experiment 3, we can observe that DP generally has a marginal impact on the memory usage in the Postgres database with increased memory of less than about 5%. Note that the database holds the dataset on which the queries are conducted.

The impact of DP by the statistical query tools is fairly consistent with our anticipation, possibly except Google DP on *Parkinson* that indicates a low memory overhead of less than 5% for certain data sizes, e.g., 4000 and 5000. Although the impact by DP on memory overhead by the statistical query tools is especially apparent on the *Health Survey*, where it at times is more than 40%. Beyond this, we notice that there exist some local irregularities and fluctuations in each plot. We detail the evaluation results below.

Figure 5.5 shows contour plots for each tool's performance on memory overhead with different ϵ and data sizes. Darker shades of blue in the plots indicate a lower δ, corresponding to lower memory overhead. Lighter shades indicate a larger δ, corresponding to higher memory overhead. Note that the plots have different δ scales. This implies that a shade in one plot (which indicates an δ value), does not necessarily correspond to the same shade in another plot.

In Figure 5.5, we cannot observe any general patterns, e.g., trends between ϵ, data size, and memory overhead. However, it is clear that DP procedures have a larger impact on memory overhead, introduced by the statistical query tools (Figures 5.5a, 5.5c, 5.5e, 5.5g), at times above 40%, compared to the impact it has on the database operations (Figures 5.5b, 5.5d, 5.5f, 5.5h) which is less than 10%. Apart from this, we notice that the RMSPE fluctuates, e.g., in Figures 5.5a, 5.5e, 5.5c, 5.5g, especially for Google DP on *Parkinson* that has an unexpected increase in memory overhead for data size 3000 to about 15%, and maximum data size to over 40% (Figure 5.5e). These irregularities appear more obvious in Figures 5.7i and 5.7l, where we notice that the `HISTOGRAM` query fluctuates between 0% and 2%, seemingly without any pattern. A similar trend appears in Figure 5.5a, e.g., for data sizes 5000, 6000 and 7000, where occasional queries, e.g., `AVG` for data size 7000 (Figure 5.6a), shows a memory overhead of 30%, while other queries are close to 0%. However, in contrast to Figure 5.5e, the memory usage seems to decrease for maximum data size to between 10% and 20%, which is also the case for maximum data size in Figure 5.5c. Incidentally, we can observe similar fluctuations in Figure 5.5g, where memory usage is about 40% for data size of 3000, about 30% for data size 5000, and about 5% for data size 4000. We can also observe an unexpected decrease in memory

usage, specifically in Figures 5.5a and 5.5e, where DP queries seem to occasionally consume less memory than NP queries. While these fluctuations are interesting we have no explanation to why they occur.

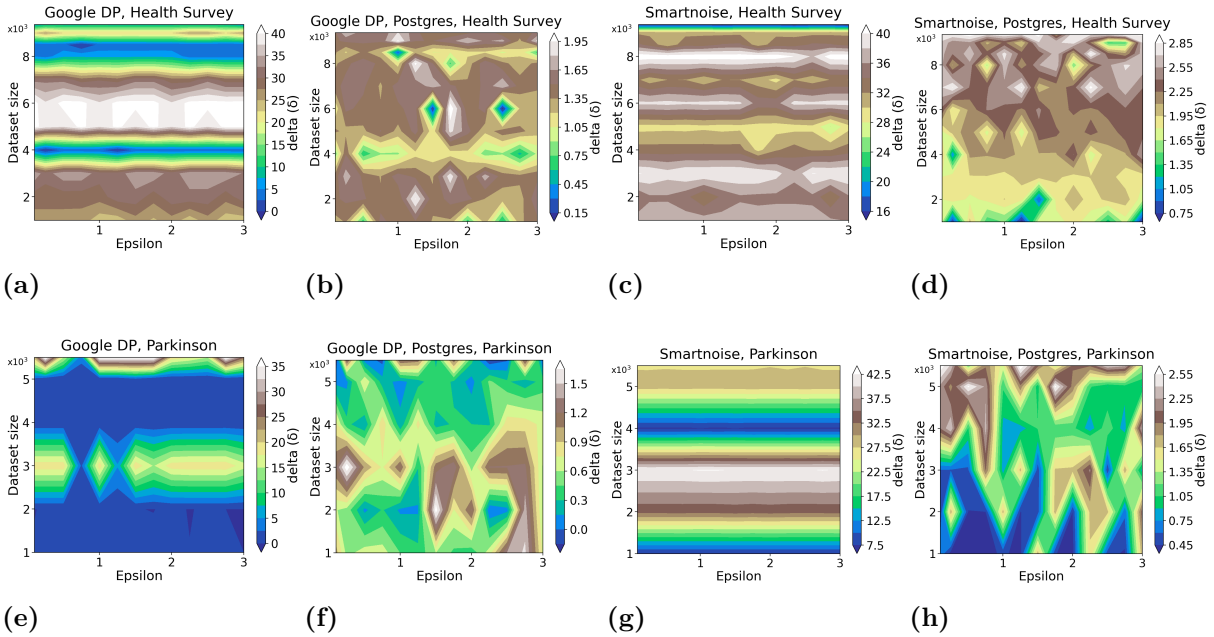

Figure 5.5: The results of experiment 3, showing the impact of statistical query tools on memory overhead for different ϵ (Table 4.3), data size (Table 4.2) and queries (Table 4.4). δ is defined in Section 4.1.

In Figure 5.6, we can observe the impact that DP has by the statistical query tools, for different ϵ and data size, and study the impact of each query. We cannot observe any general relationship between ϵ, data size, and memory overhead. However, we notice that the impact of the `HISTOGRAM` query fluctuates significantly, especially for *Parkinson*, between -30% and 40%. It seems that different ϵ values have a different impact on memory overhead when DP is applied to the `HISTOGRAM` query on *Parkinson*. In some cases, the DP `HISTOGRAM` query has lower impact on memory than its corresponding NP query. However, there seems to be no obvious patterns in this behavior. There are also some apparent irregularities on *Health Survey*, which is obvious in Figure 5.6a (`AVG` query), Figure 5.6c (`SUM` query), Figure 5.6e (`HISTOGRAM`, `AVG` query), Figure 5.6f (`COUNT` query), and Figure 5.6g (`AVG` query).

In Figure 5.7, we can observe the impact that DP has on the Postgres database on which the queries are conducted on. These results are not consistent with our anticipation, and have considerably less impact on memory overhead than DP imposes by the statistical query tools (Figure 5.6). There seems to be no general connection between ϵ, data size, and memory overhead. However, we can observe that the `HISTOGRAM` query fluctuates considerably more than the other queries for *Parkinson*, between 0% and 3%. It also appears to have a different impact on memory usage of the database for different ϵ values, though we cannot observe any general patterns.

In general, memory usage fluctuates considerably for all queries, as can be seen in Figure 5.7. We have no explanation for why this happens. However, it is worth noting that the memory impact is in general notably low ($\lessapprox 3\%$). Additionally, since we measure the memory usage of the Docker container that hosts the database every second, other processes might have a large impact on the results during measurement.

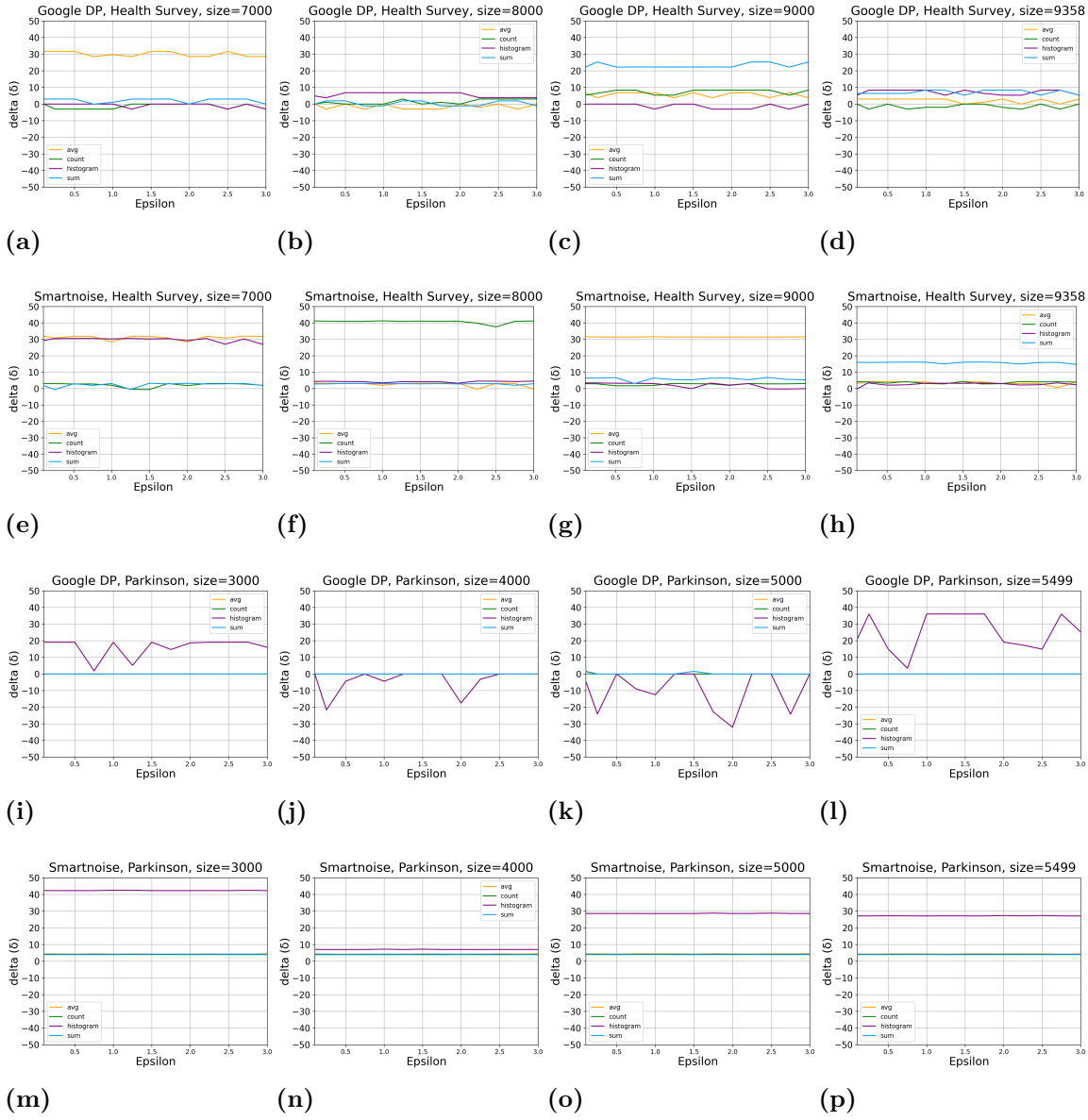

Figure 5.6: The results of experiment 3, showing the impact of statistical query tools on memory overhead when DP is integrated, for different ϵ (Table 4.3), data size (Table 4.2) and queries (Table 4.4). δ is defined in Section 4.1.

Overall, Smartnoise consumes marginally more memory than Google DP in the database on *Health Survey*, while their memory usage on *Parkinson* is approximately the same. Otherwise, Google DP consumes generally less memory on *Parkinson* and

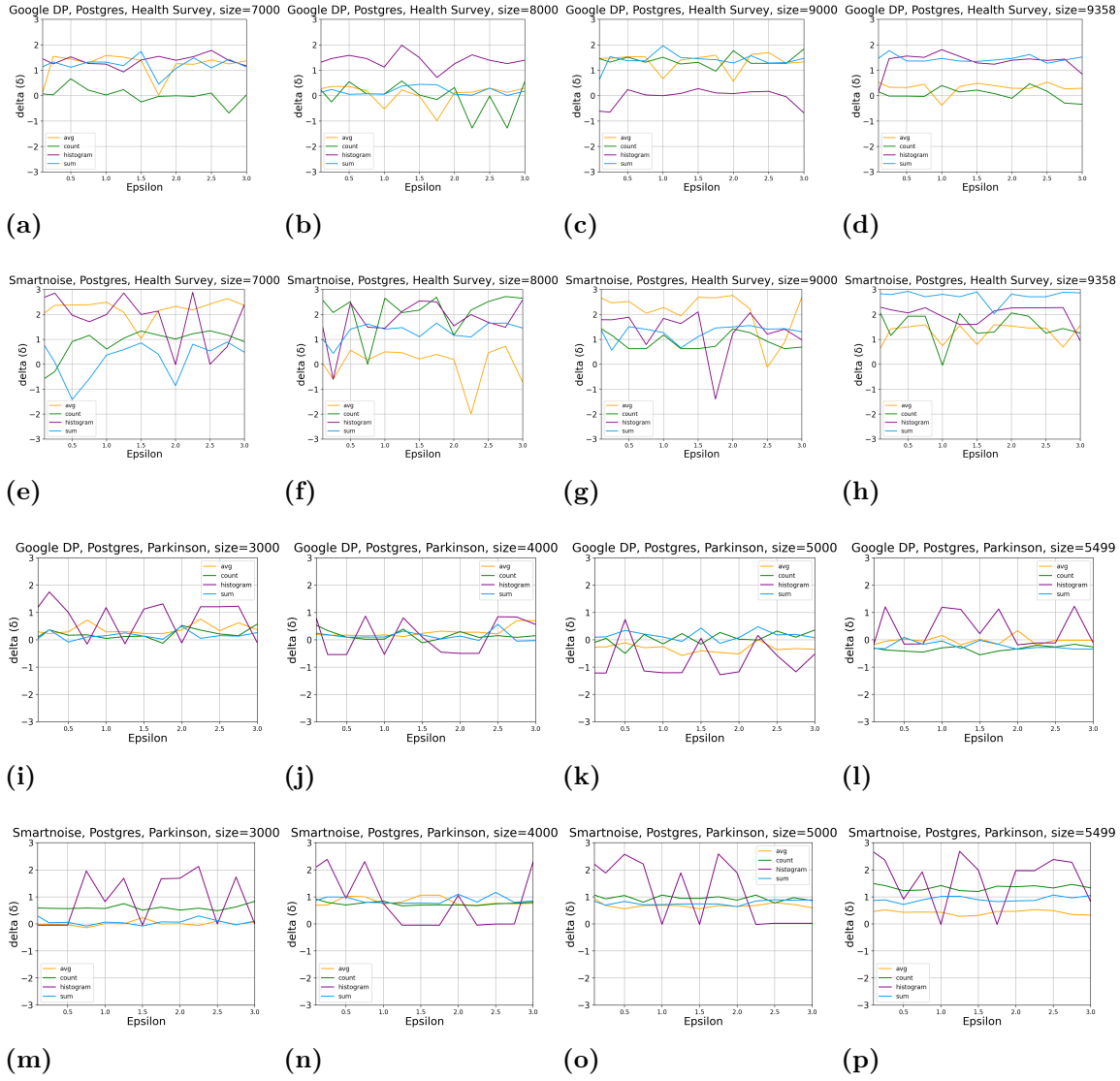

Figure 5.7: The results of experiment 3, showing the impact of statistical query tools on memory overhead when DP is integrated, for different ϵ (Table 4.3), data size (Table 4.2), and queries (Table 4.4). Specifically, the impact on memory overhead in the database (Postgres) that the queries are conducted on. δ is defined in Section 4.1.

Health Survey, than Smartnoise.

5.4 Experiment results of machine learning tools

Non-privacy protected (NP) ML models might jeopardize the privacy of individuals. I.e., information of individual data can be disclosed in the training data [46]. Differential privacy (DP) can be integrated into ML tasks to address this issue, and provide formal guarantees for individual privacy. However, applying DP to ML tasks might reduce the accuracy of ML models, and increase system resources needed to train the DP models [12].

In this section, the experiment in Section 5.4.1 studies the impact that ML tools have on data utility when training DP-ML models. The experiment in Section 5.4.2 studies the impact that ML tools have on the run-time overhead (Section 4.1) when DP is applied, while the experiment in Section 5.4.3 investigates their impact on memory overhead. We focus on the impact that DP has on ML tasks by measuring how the performances of ML tools differ between the DP and NP-ML tasks. Note that Memory overhead is measured by comparing the worst-case memory usage between DP and NP queries or ML tasks and shows the percentile difference, denoted as δ (Section 4.1). Also note that the frequency by which memory usage was recorded was imposed by the Docker containers, and was recorded once every second. The definition and calculation of data utility and system overhead are shown in Section 4.1.

Experiment results are obtained by conducting linear regression tasks on two different datasets, i.e., *Health Survey* and *Parkinson* (Section 4.2). The models are trained using different combinations of ϵ (Table 4.3) and data size (Table 4.2), since these parameters influence the results the most [4, 18, 58].

During the experiment, each model training runs for 10 times. Note that this deviates from the statistical query experiments (Section 5.3) that run 20 times, due to computational time restraints, as detailed below. The result presented here considers the average of 8 queries. That is, the two extrema results are removed to not include outliers, and the remaining 8 are averaged to generate the final results for analysis. Additional experiment settings on the range of ϵ and data size are listed in Table A.2. The list of experiments for ML tools is found in Section 4.3.2.

We run the experiments for approximately 6 days, which in addition to limited time is the primary reason for running each experiment 10 times, in comparison to experiment 1, 2, and 3 that run 20 times. Also, since the models we train are extracted from the data as a whole, the difference might be insignificant, in comparison to each query that might differ considerably. Furthermore, we train around 7000 models, of which approximately 4500 are obtained with Opacus and Tensorflow Privacy (TFP). Due to additional computations of DP, this process consumes time. In addition, for each data size, we have to compute the `noise_multiplier` parameter for every ϵ (Algorithm 1), which is the privacy parameter for for Opacus and TFP. This computation is done on the CPU power, and even though we utilize threads it still takes considerable time to perform the computation, which is another contributing

time factor.

The evaluated ML tools in this section include Tensorflow Privacy, Opacus, and Diffprivlib, whose performances are shown in the following.

5.4.1 Experiment 4. Data utility

The implementation of differential privacy (DP) by ML tools has an inherent impact on data utility. In experiment 4, we study that impact by comparing how the performance of DP-ML models differs from non-privacy protected (NP) ML models. We also study how the impact varies between different ML tools.

In this experiment, we anticipate that higher ϵ values and the larger data size will provide better utility for both datasets since less noise can be added during model training in DP machine learning procedures. As we detail below, from experiment 4, we can observe that as ϵ and data size increase, the learned machine learning models perform better. I.e., higher ϵ and larger data size provide better utility. Beyond this expected trend, we notice that there exist some local irregularities in each plot that break the trend. In addition, Diffprivlib does not produce any useful results for *Parkinson*. We further detail the evaluation results below.

In Figure 5.8, we can observe that higher ϵ values decrease the RMSPE (Section 4.1). This implies the higher utility of the DP-ML model. This trend holds for both the two considered datasets. The trend is quite explicit, especially for the performance of the models trained on *Parkinson*. The *Parkinson* dataset also performs in line with our anticipations when it comes to the relationship between data size and utility, while the model-training results on *Health survey* data show marginal levels of fluctuation.

Figure 5.9 shows contour plots for each tool's performance with different ϵ and data size. Darker shades of blue in the plots indicate lower RMSPE, corresponding to higher utility. Lighter shades indicate a larger RMSPE, corresponding to lower utility. The plots have different RMSPE scales. This means that a shade (which indicates an RMSPE value) in one plot, does not necessarily correspond to the same shade (and RMSPE value) in another plot. The white areas represent over 20% difference from NP results. Note that Figure 5.9f shows RMSPE on a $\text{x}10^{20}$ scale, and Figures 5.8g, 5.8h, 5.8i shows RMSPE on a logarithmic scale.

In the contour plots (Figure 5.9), we can observe that, for experiments conducted on *Health Survey*, Diffprivlib achieves marginally better accuracy than Tensorflow Privacy (TFP) and Opacus for high ϵ ($\gtrapprox 2.5$) and large data size ($\gtrapprox 9000$), between about 0.3% and 2.2% RMSPE. TFP achieves between about 2.9% and 5.8% RMSPE, and Opacus between about 3.3% and 4.5% for the largest data sizes. However, Diffprivlib obtains considerably worse accuracy for small ϵ values ($\lessapprox 1.0$), between about 11% and $3.3\dot{1}0^{8}$% compared to about 9% to 16% for TFP and about 6% for Opacus (Figure 5.9). For experiments on *Parkinson*, we observe that TFP generally

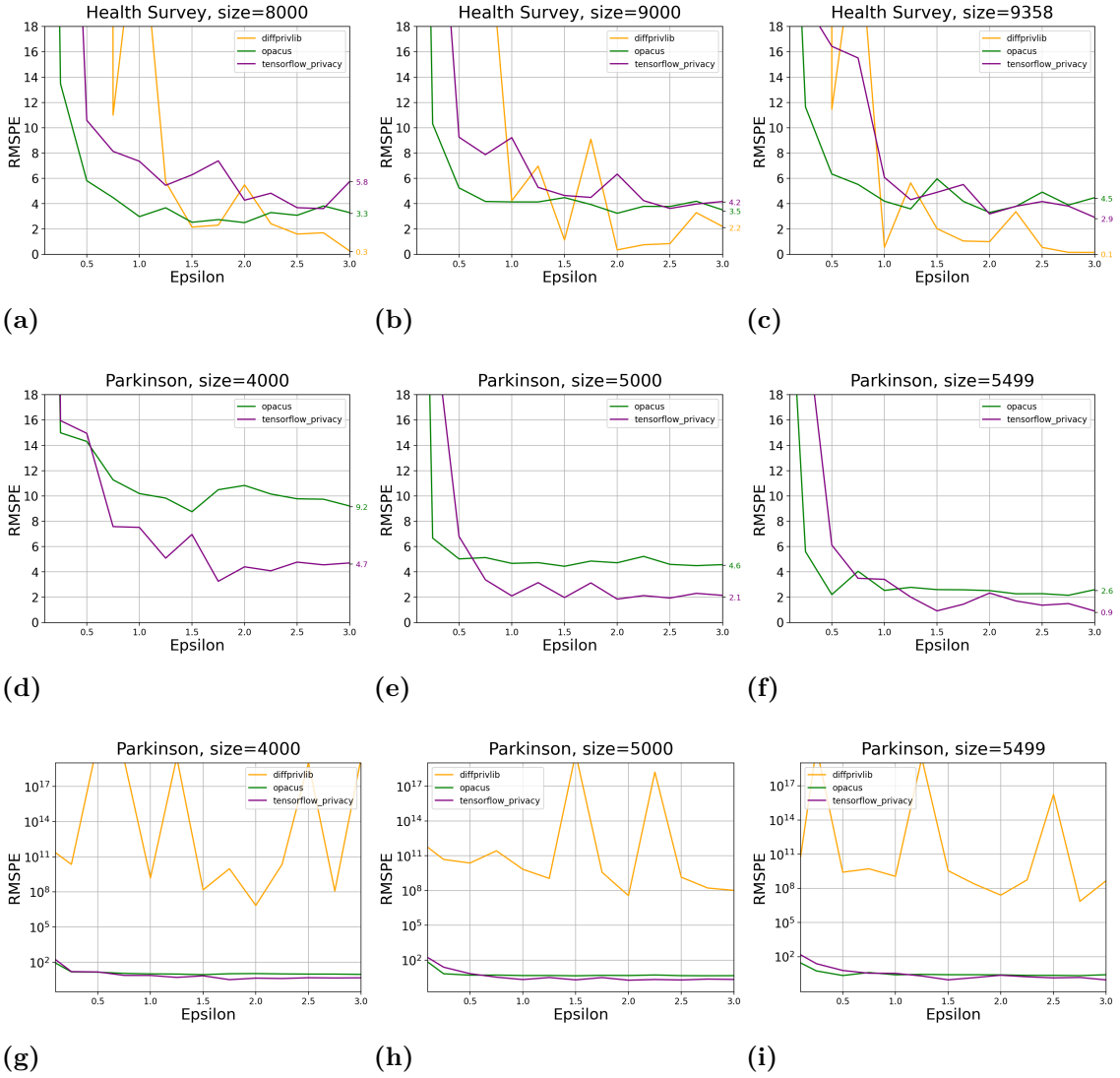

Figure 5.8: The results of experiment 4, showing the impact of ML tools on data utility for different ϵ (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

obtains about 50% less RMSPE than Opacus (Figures 5.8d, 5.8e, 5.8f). We also notice that Opacus does not produce useful results for small data size (Figure 5.9e). Moreover, a decrease in accuracy for Opacus exists when data size is 4000 (Figure 5.9e), which we cannot explain. In comparison, TFP shows a more apparent direct relationship between ϵ, data size, and utility (Figure 5.9d).

We would like to note that Diffprivlib does not produce any useful results, i.e., practical linear regression models (Figures 5.9f, 5.8g, 5.8h, 5.8i). We believe that the different method Diffprivlib uses to achieve DP (Section 2.6.1, 3.2.3), compared to Opacus and TFP, plays a role here. This method difference, in combination with the fact that *Parkinson* has dominantly continuous values, might be the source of this performance. In comparison, Diffprivlib obtains reasonable accuracy for the

Health Survey, with corresponding data size 5000 and $\epsilon = 3.0$. It is also worth noting that Diffprivlib generates comparably equal accuracy to Opacus and TFP when no DP is applied on both datasets.

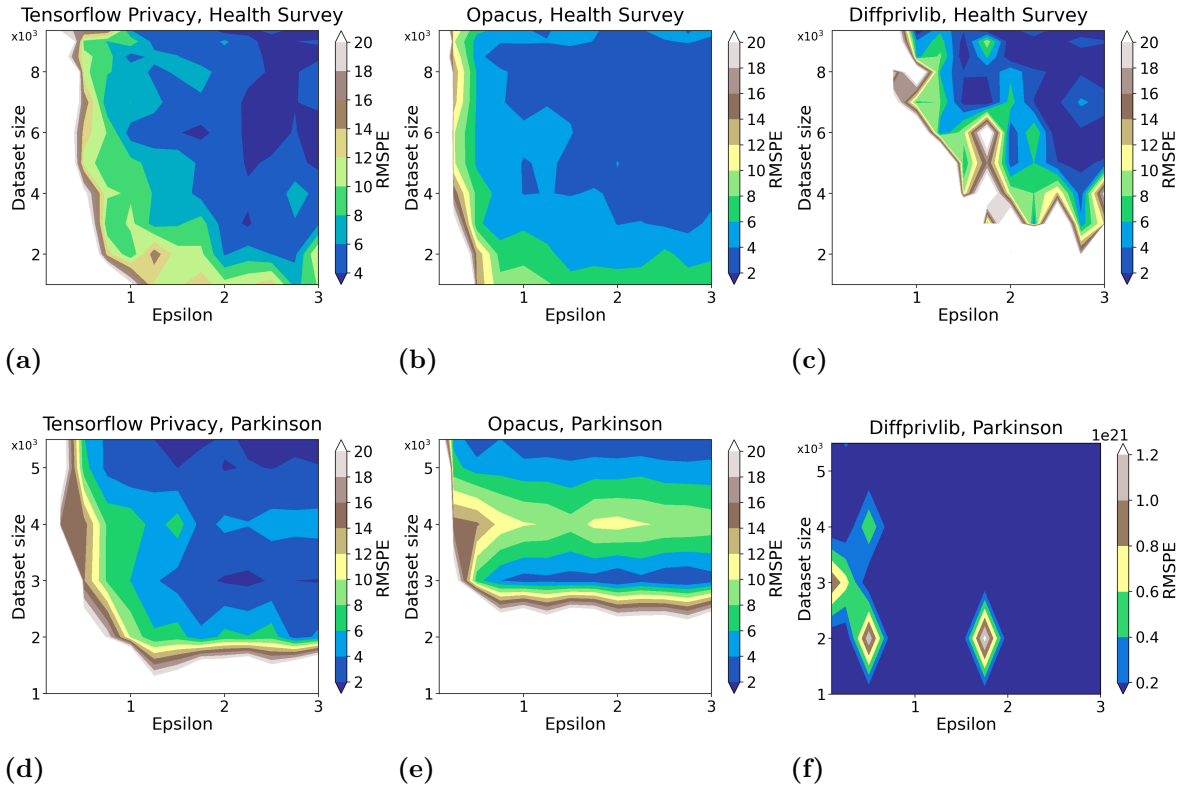

Figure 5.9: The results of experiment 4, showing the impact of ML tools on data utility for different ϵ (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

5.4.2 Experiment 5. Run-time overhead

The implementation of differential privacy (DP) by ML tools inevitably requires additional system resources, and might thus have an impact on run-time overhead (Section 4.1). In experiment 5, we study this impact by comparing how the run-time of DP-ML tasks differ from non-privacy protected (NP) ML tasks, specifically linear regression. We also study how this difference varies between different ML tools.

For this experiment, we anticipate that DP-ML will have a considerable impact on the run-time overhead since additional DP computations take place during the ML tasks (Section 2.6.1). As detailed below, in experiment 5, we observe that run-time has generally increased when DP is integrated into linear regression. This is consistent with our anticipation since the DP-SGD for instance, is the algorithm that Opacus and Tensorflow Privacy (TFP) apply for DP-ML training (Section 2.6.1), has an inherent impact on the run-time in general [24, 25]. Such a trend is apparent in the results of TFP which has a run-time overhead of about between 50% and

25%, and in the results of Opacus which has a run-time overhead of between about 200% and 250%. Beyond the expected general increase in run-time, we notice that there exist some local irregularities. In addition, there are also undesired results, i.e., Diffprivlib does not produce any useful results for Parkinson, which is elaborated in experiment 4 (Section 5.4.1). We therefore do not include Diffprivlib's run-time overhead in Figures 5.10 and 5.11 regarding with its results on Parkinson dataset. The evaluation results are detailed below.

Figure 5.10 shows contour plots for each tool's run-time overhead in regard to different ϵ and data size. Darker shades of blue in the plots indicate lower RMSPE (Section 4.1), corresponding to higher utility. Lighter shades indicate a larger RMSPE, corresponding to lower utility. The plots have different RMSPE scales. This means that a shade in one plot (which indicates an RMSPE value), does not necessarily correspond to the same shade in another plot.

In Figures 5.10 and 5.11, we cannot observe any general patterns, e.g., correlational trends between ϵ, data size, and run-time overhead. However, different ϵ values impact TFP run-time during DP training in an irregular manner. It is quite explicit, especially for *Parkinson* in Figure 5.11, and there seem to be no obvious pattern. In addition, there are several unexpected local irregularities. For instance, Opacus generally consumes around 200% more time for DP training (Figures 5.10c, 5.10e). However, for size 3000 and $\epsilon \geq 2.0$, it consumes around 100% less time (Figure 5.11e). A local fluctuation is also observed for TFP on *Health Survey* in Figure 5.11a when $\epsilon = 2.0$, where TFP consumes over 100% more time, which is an interesting result that warrants further study.

In general, Diffprivlib's DP training has a higher impact on the run-time, in comparison to its NP training (Figure 5.10c). However, it is important to note that Diffprivlib consumes significantly less absolute time in DP model training than TFP and Opacus, due to its different methods for applying DP (Section 2.6.1). Moreover, since TFP and Opacus use the same approach for DP training, it is fair to compare them by how much DP training impacts their run-time, in comparison to their NP training. In regard to that, TFP performs considerably better than Opacus (Figures 5.10a, 5.10d), while Opacus in general performs more consistently. I.e., its run-time does not fluctuate for different ϵ values as frequently as TFP's over different data sizes and ϵ settings (Figure 5.11).

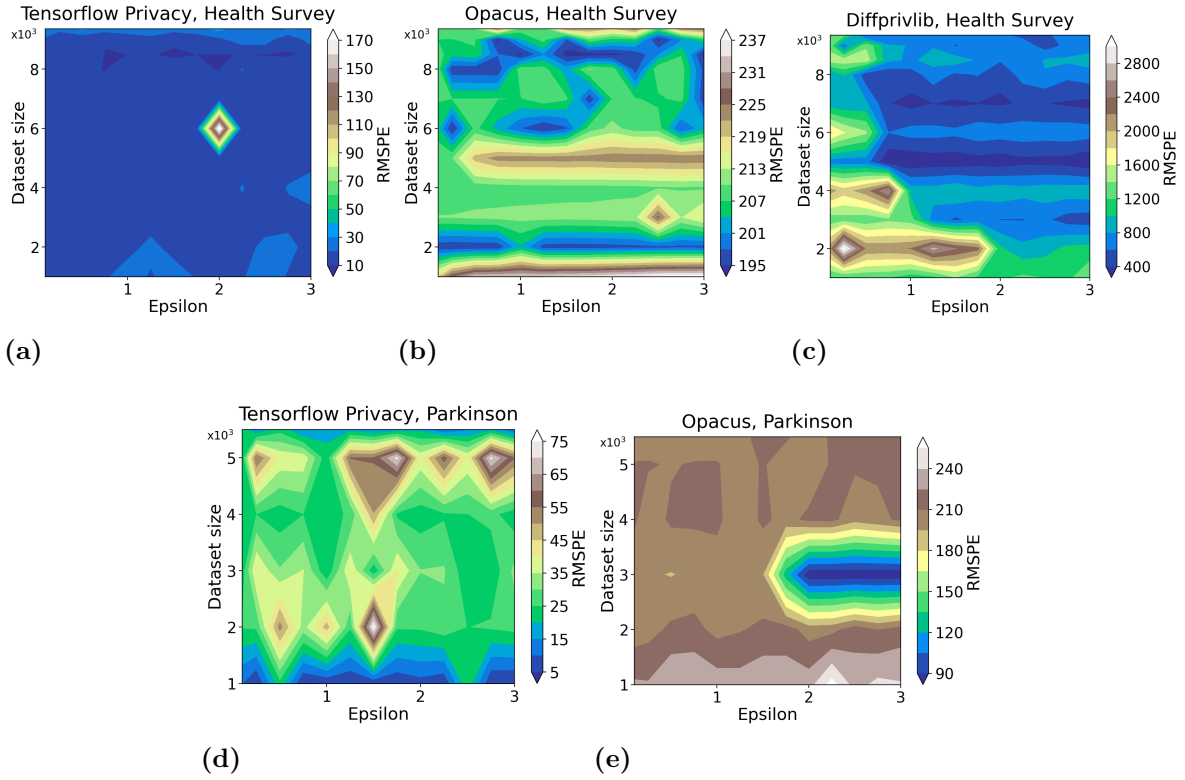

Figure 5.10: The results of experiment 5, showing the impact that DP has on run-time overhead for different ML tools, with different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

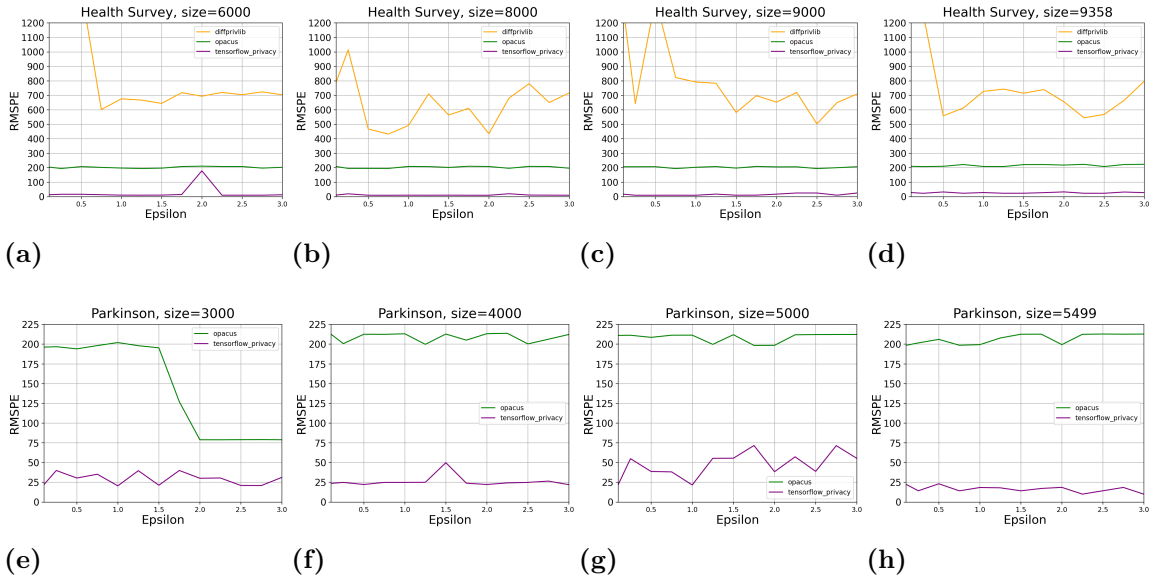

Figure 5.11: The results of experiment 5, showing the impact that DP has on run-time overhead for different ML tools, with different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

5.4.3 Experiment 6. Memory overhead

The implementation of differential privacy (DP) by ML tools might have an inherent impact on memory overhead. In experiment 6, we study that impact by comparing how memory usage differs between DP and non-privacy protected (NP) ML tasks, and investigate how this difference varies between different machine learning tools.

For this experiment, we anticipate a marginal increase in memory overhead when DP is integrated, since methods for applying DP during model training might be memory intensive in practice [24], specifically for Opacus and Tensorflow Privacy (TFP). As detailed below, in experiment 6, we can observe that DP sometimes has a more than a marginal effect on the memory usage, which is especially apparent for TFP on *Health Survey* and *Parkinson* for data size below maximum size. Opacus has a smaller memory overhead for both the considered datasets and performs in line with what we anticipated. We also notice several irregularities and fluctuations in the results, e.g., a sudden decrease in memory usage for TFP and Diffprivlib on *Health Survey* with maximum data size, and TFP on *Parkinson* with maximum data size. Besides, Opacus shows small fluctuations for specific ϵ values on *Health Survey*. There is also an undesired result, i.e., Diffprivlib does not produce any useful results for Parkinson, which is elaborated in experiment 6 (Section 5.4.3). We therefore do not include that case in Figures 5.12 and 5.13. The evaluation results are detailed below.

Figure 5.12 shows contour for each tool's memory overhead in regard to different ϵ and data size. Darker shades of blue in the plots indicate lower δ (Section 4.1), corresponding to higher memory usage. Lighter shades indicate a larger δ, corresponding to lower utility. The plots have different δ scales. This means that a shade in one plot (which indicates a δ value), does not necessarily correspond to the same shade in another plot.

In Figure 5.12, we cannot observe any explicit patterns, e.g., connections between ϵ, data size, and memory overhead. However, we notice that TFP have memory overhead of about 90% for the *Parkinson* dataset and about 20% for the *Health Survey* dataset (Figures 5.12a, 5.12e). Opacus (Figures 5.12b, 5.12d) on the other hand has less than 5% memory overhead in either of the datasets. TFP and Diffprivlib also obtains an unexpected decrease in memory usage for maximum data size on *Health Survey* (Figures 5.12a, 5.12c), while TFP also obtains a decrease for *Parkinson* (Figures 5.12e). Opacus also seems to consume marginally more memory for data size 2000 on *Parkinson* (Figure 5.12d). We cannot explain these irregularities.

In Figure 5.13, we can observe that Opacus generally has less impact on the memory overhead than TFP as detailed in the previous section. It is especially apparent on *Parkinson* (Figures 5.13e, 5.13f, 5.13g, 5.13h). Besides this observation, we notice that Diffprivlib on *Health Survey* has an considerable decrease in memory usage for the maximum data size (Figure 5.13d). This is unexpected since Diffprivlib applies the closed-form solution for computing linear regression, where training data has to fit into RAM (Section 2.6.1). So intuitively, we should observe a slight increase in

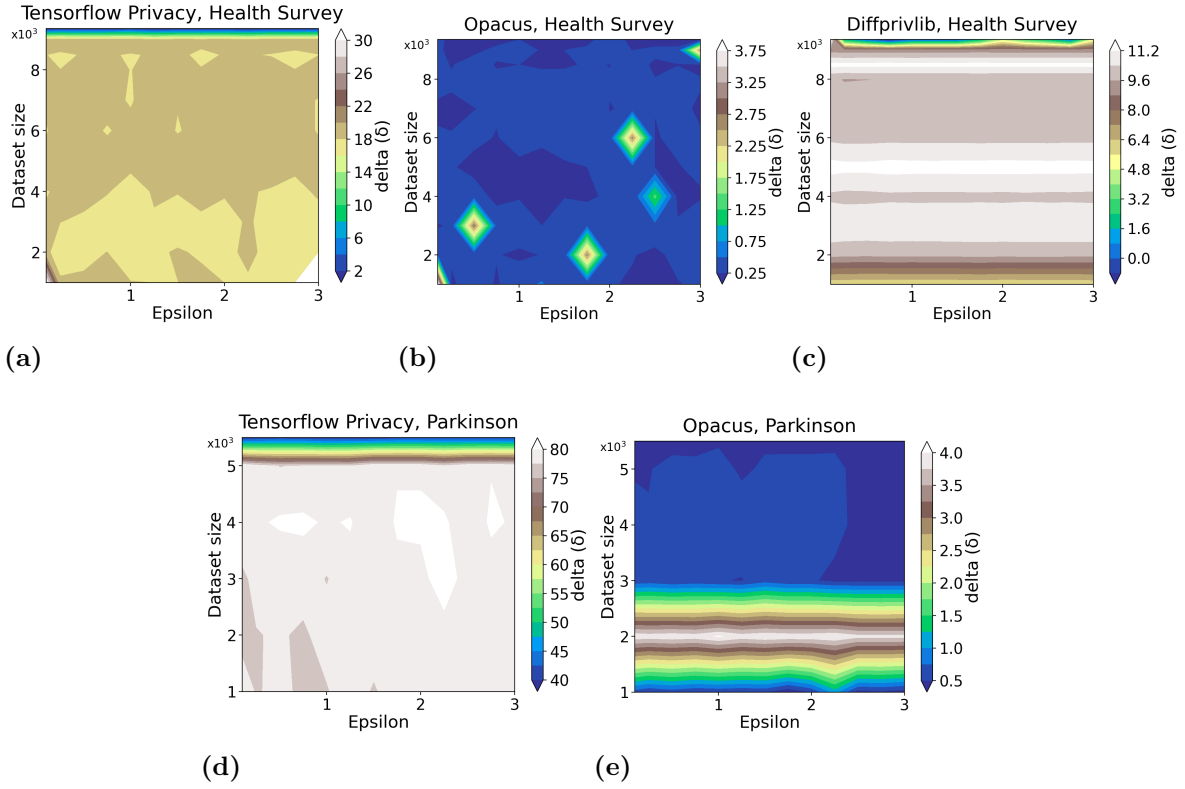

Figure 5.12: The results of experiment 6, showing the impact that DP has on memory overhead for different ML tools, with different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

memory usage for Diffprivlib when data size increases, which is not the case in the results (Figure 5.13d). In addition, Diffprivlib seems to consume marginally more memory for $\epsilon = 0.1$ and maximum data size, where we can observe the memory usage decrease for ϵ values above 0.1 (Figure 5.13d). These are interesting results that warrant further research.

Overall, Opacus performs the best in regard to memory consumption when DP is applied, between around 0.25% and 4% across all data sizes and ϵ and for both datasets. However, it is important to note that Diffprivlib uses a different method for applying DP from the other two tools (Section 2.6.1). Moreover, since TFP and Opacus use the same approach for DP training, it is fair to compare them by how much DP training impacts their memory overhead, in comparison to their NP training. In regard to that, Opacus performs better than TFP, which consumes between about 2% and 40% more memory on maximum data size compared to its NP training.

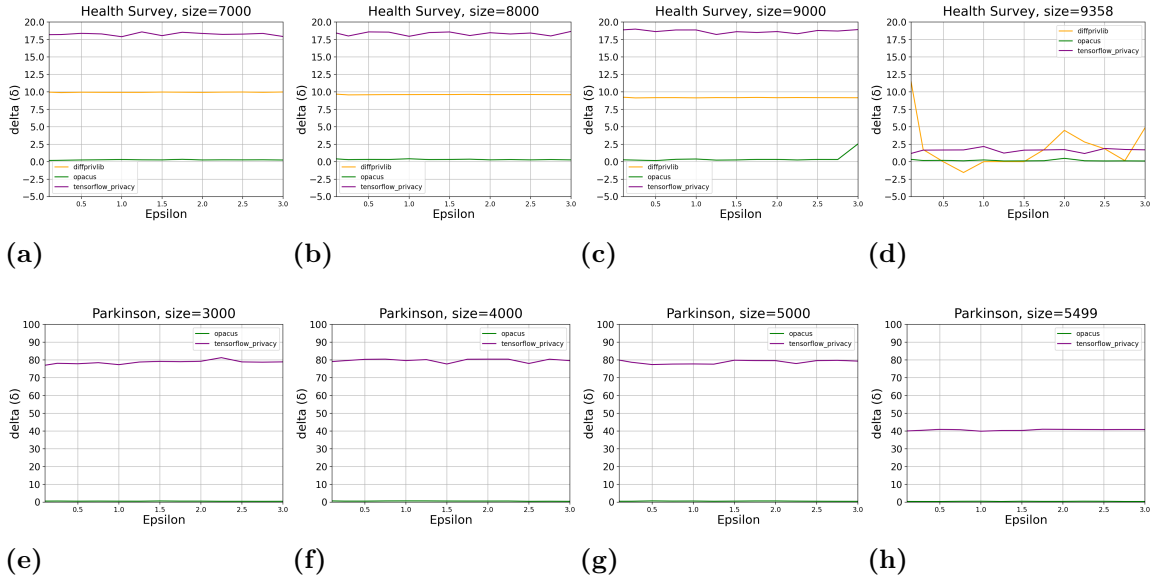

Figure 5.13: The results of experiment 6, showing the impact that DP has on memory overhead for different ML tools, with different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

5.5 Experiment results of synthetic data tools

Synthetic data presents a way of substituting real data to preserve privacy during data analysis. The synthetic data is generated from the original dataset and aims to retain as many statistical properties of the original dataset as possible, while its rows do not belong to any of the rows in the original data. However, techniques aiming to preserve individuals' privacy during synthetic data generation (SDG) lack formal privacy guarantees, and information about individual data might thus be disclosed from the generated synthetic data [51, 52, 53]. To address this privacy problem, differential privacy (DP), which provides formal guarantees of privacy, can be integrated into the SDG process. While we expect to retain statistical properties of the original dataset and preserve privacy, using synthetic data to conduct analysis such as ML tasks and statistical queries on synthetic datasets, as a substitute for the original datasets, might impact the utility of the analysis and increase the system resources needed in order to conduct the analysis.

The experiments in this section study the impact that the DP-SDG tools have on data utility and system overhead (run-time and memory overhead, Section 4.1). We generate synthetic versions of two different datasets *Health Survey* and *Parkinson* (Section 4.2), with different combinations of data size (Table 4.2) and ϵ (Table 4.3), since these are the parameters that influence the result the most [4, 18, 58]. We then measure how much the results of statistical queries and ML tasks on synthetic datasets deviates from those conducted on the original (non-privacy protected (NP)) dataset.

In our evaluation, we consider two tools for generating DP synthetic data: Gretel Synthetics (Section 3.3.2) and Smartnoise Synthetics (Section 3.2). Smartnoise Synthetics uses three algorithms, or Synthesizers as they are named internally, for generating synthetic data. These are MWEM, PATECTGAN and CTGAN. Each of the synthesizers in Smartnoise Synthetics will be evaluated individually in this evaluation and compared.

The content of this section is as follows. Experiment 7 (Section 5.5.1) studies the data utility of the synthetic datasets by comparing statistical queries on the synthetic datasets and the original datasets. Experiment 8 in Section 5.5.2 studies the data utility of ML tasks by training linear regression models on the synthetic data and evaluating them on the corresponding real data, referred to as train-synthetic test-real (TSTR). The system overhead of the tools are evaluated in Section 5.5.3 (run-time) and 5.5.4 (memory). In contrast to previous experiments (Sections 5.3, 5.4), where run-time and memory overhead is measured and compared between DP and non-private (NP) tasks, the experiments for SDG measure absolute run-time and memory since we have no NP baseline to compare the results against for these experiments.

We would like to note that Gretel Synthetics and all the synthesizers in Smartnoise Synthetics provide plenty of hyperparameters for tuning the training process. It is not trivial how to optimally configure them. Our framework, therefore, allows us to provide different values for each hyperparameter and thus generate a synthetic dataset for each combination of the provided values. We thereafter select generated synthetic datasets by using the Chi-squared test, which ranks the datasets by comparing the distributions of each column [86]. Using this method, we do not have to know how to configure the tools. Instead, we can simply 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. The downside of this method is that the number of synthetic datasets grows exponentially, essentially with no upper limit, and might become far too many to generate in a reasonable amount of time. To combat this, we limited the number of hyperparameters (Appendix A.3.1).

We notice in the evaluation results that Smartnoise DPCTGAN fails to generate datasets for some combinations of ϵ and data size. Moreover, both Gretel and Smartnoise Synthetics fail to generate any datasets for the *Parkinson* dataset using MWEM. We believe that those tools might be unfitting for continuous data, however, further study is reasonable for any conclusions. Smartnoise MWEM also failed to generate datasets for large data size with larger ϵ values for *Health Survey*, which we cannot explain.

Additionally, Gretel Synthetics does not provide functionality for targeting specific ϵ values. Even though the library is built around Tensorflow, we could not access the functionality that would allows us to compute `noise_multiplier` values (Section 3.2) for target ϵ values, as we do for Tensorflow Privacy (Algorithm 1). Instead,

it runs with a set of hyperparameters and outputs the final ϵ value at the end of the data generation. We, therefore, do not have a dataset for each ϵ value in experiments on Gretel, as we do for the other tools. Also, note that the ϵ values that Gretel Synthetics outputs are significantly higher than the ϵ values that we obtain for the other tools. To make a reasonable comparison, we generate many synthetic datasets for each data size and with different hyperparameters. We thereafter select the synthetic dataset with the lowest ϵ value and use that for our comparison. Thus, instead of having one synthetic dataset for each combination of ϵ and data size, we have one synthetic dataset for each data size for Gretel Synthetics as shown in Table 5.1.

data size	ϵ
2000	24.6
3000	32.6
4000	32.0
5000	31.0
6000	30.0
7000	30.0
8000	29.0
9000	28.0
9358	55.3

Table 5.1: List of the lowest ϵ values obtained for each data size of the synthetic versions of the *Health Survey* dataset, using Gretel Synthetics.

5.5.1 Experiment 7. Statistical query utility

Running statistical query tasks on synthetic data instead of the real data might have an impact on the utility of the task. In experiment 7, we study that impact by comparing the results between queries conducted on synthetic datasets vs the real data. We also study how the impact varies between different tools for synthetic data generation (SDG).

For the SDG tools application, we generate the same number of rows as in the original dataset, i.e., when a model is trained on 7000 rows of the *Health Survey* dataset, we generate a corresponding synthetic dataset with 7000 rows. This implies that there will not be any difference conducting the `COUNT` query on the synthetic dataset vs the original. We, therefore, exclude any plots of the `COUNT` query in this section. Moreover, since the number of rows in the original and the synthetic datasets are the same, the RMSPE (Definition 4.1, Section 4.1) of the `AVERAGE` and the `SUM` query

will be the same. Consequently, we only include plots for the `AVERAGE` query.

Furthermore, since higher ϵ values represent adding less noise to the data, and more data implies that each individual's contribution to the final result is smaller [22], we anticipate that synthetic datasets generated with higher ϵ values and larger data size, will provide better utility for both datasets, i.e. *Health Survey* and *Parkinson*.

As detailed below, from experiment 7, we cannot see any obvious trends that are exactly consistent with our anticipation. Figure 5.14 shows contour plots for each tool's performance in regard to different ϵ and data size. Darker shades of blue in the plots indicate lower RMSPE, corresponding to higher data utility, while lighter shades indicate a larger RMSPE, corresponding to lower data utility. The white areas indicate that the tools failed to generate datasets for those combinations of ϵ and data size. Also note that the plots have different RMSPE scales, which means that a shade (which indicates an RMSPE value) in one plot does not necessarily correspond to the same shade in another plot. Moreover, since Gretel Synthetics does not explicitly provide functionality for targeting ϵ values, we could not generate datasets for each target ϵ (Table 4.3). We, therefore, omit the Gretel Synthetics results from Figure 5.14. Also, since Gretel Synthetics and Smartnoise MWEM fail to generate any datasets for Parkinson, these cases are also omitted from Figure 5.14.

In Figure 5.14, we can observe that, for experiments conducted on *Health Survey*, Smartnoise PATECTGAN obtains better accuracy for smaller data size (RMSPE between about 35% to about 80% for data size of 1000 rows and between about 50% and 70% for data size of 9358 rows) and generally better data utility for larger ϵ values (RMSPE between about 60% and 80% for the ϵ value of 0.1 and between about 30% and 65% for the ϵ value of 3). Apart from this, we cannot see any regular trends in Figure 5.14. However, we can observe that Smartnoise DPCTGAN fails to generate datasets for some combinations of ϵ and data size on *Parkinson*, and Smartnoise MWEM fails to generate datasets for large data sizes in combination with large ϵ values on *Health Survey*. We think that DPCTGAN might be unfitting for the categorical data in the *Parkinson* dataset, and fails therefore to generate all desired datasets. We have no explanation why MWEM fails on *Health Survey*.

The line plots in Figure 5.15 show how data utility of the tools compare for each data size and $\epsilon \in \{0.1, 1, 2, 3\}$. Since Gretel Synthetics does not support setting ϵ values explicitly, we only show its results for the smallest ϵ value that was obtained, which is still relatively high compared to target ϵ values (Table 4.3). The ϵ values for the synthetics datasets generated by Gretel Synthetics are listed in Table 5.1.

Smartnoise PATECTGAN and Smartnoise DPCTGAN perform comparably for both datasets. For the `AVERAGE` query on the *Health survey*, DPCTGAN has an RMSPE between about 20% and 70%, PATECTGAN between about 35% and 80%, MWEM between about 50% and 130% and Gretel between about 20% and 30%. For the `AVERAGE` on the *Parkinson* dataset, the data points that we obtained on DPCTGAN has an RMSPE between about 80% and 100% and PATECTGAN has

an RMSPE between 60% and 80%. This means that DPCTGAN performs better on *Health Survey* dataset while PATECTGAN performs better on the *Parkinson* dataset. We believe this is due to that DPCTGAN might be better suited for categorical data in the *Health Survey* dataset, while PATECTGAN is better suited for continuous data. Furthermore, Gretel Synthetics performs well for the `AVERAGE` queries on *Health Survey*. However, Gretel Synthetics applies a significantly larger privacy budget ϵ. We also notice that `HISTOGRAM` queries has an RMSPE above 100% for all data points which is significantly worse data utility than `AVERAGE` queries on *Health Survey*. The `HISTOGRAM` queries on the *Parkinson* dataset on the other hand has an RMSPE of between about 20% and 180%, therefore it is difficult to say whether the `HISTOGRAM` or `AVERAGE` queries performs better in this case.

We can observe irregularities in essentially all plots. These are especially apparent for the `HISTOGRAM` queries on *Parkinson* (Figures 5.15m, 5.15n, 5.15o, 5.15p), where synthetic data cannot be generated for some combinations of data size and ϵ by DPCTGAN. While we cannot explain these results, it might indicate a difficulty in dealing with continuous dataset for the considered tools.

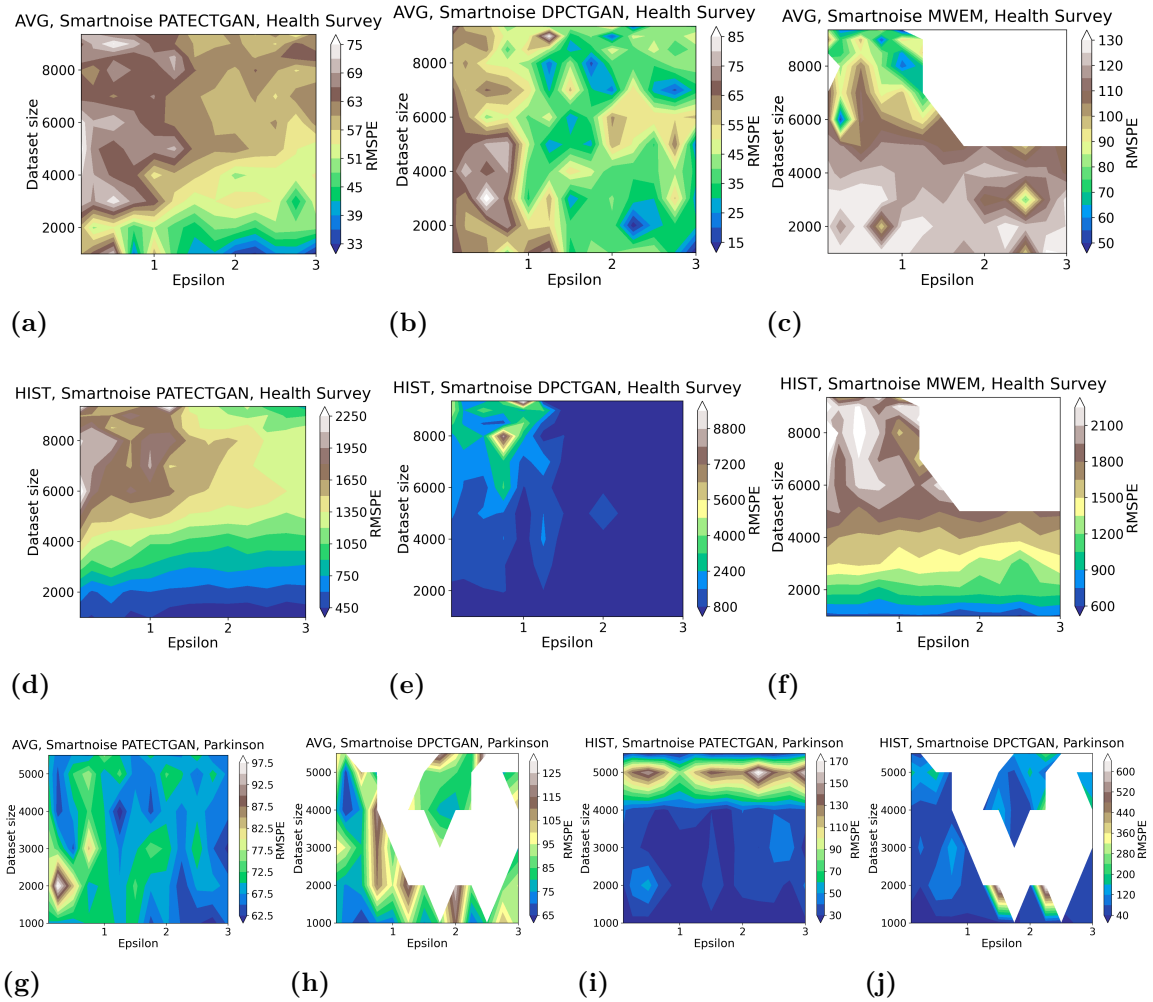

Figure 5.14: The results of experiment 7, showing the impact that DP has on data utility for different SDG tools, with different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

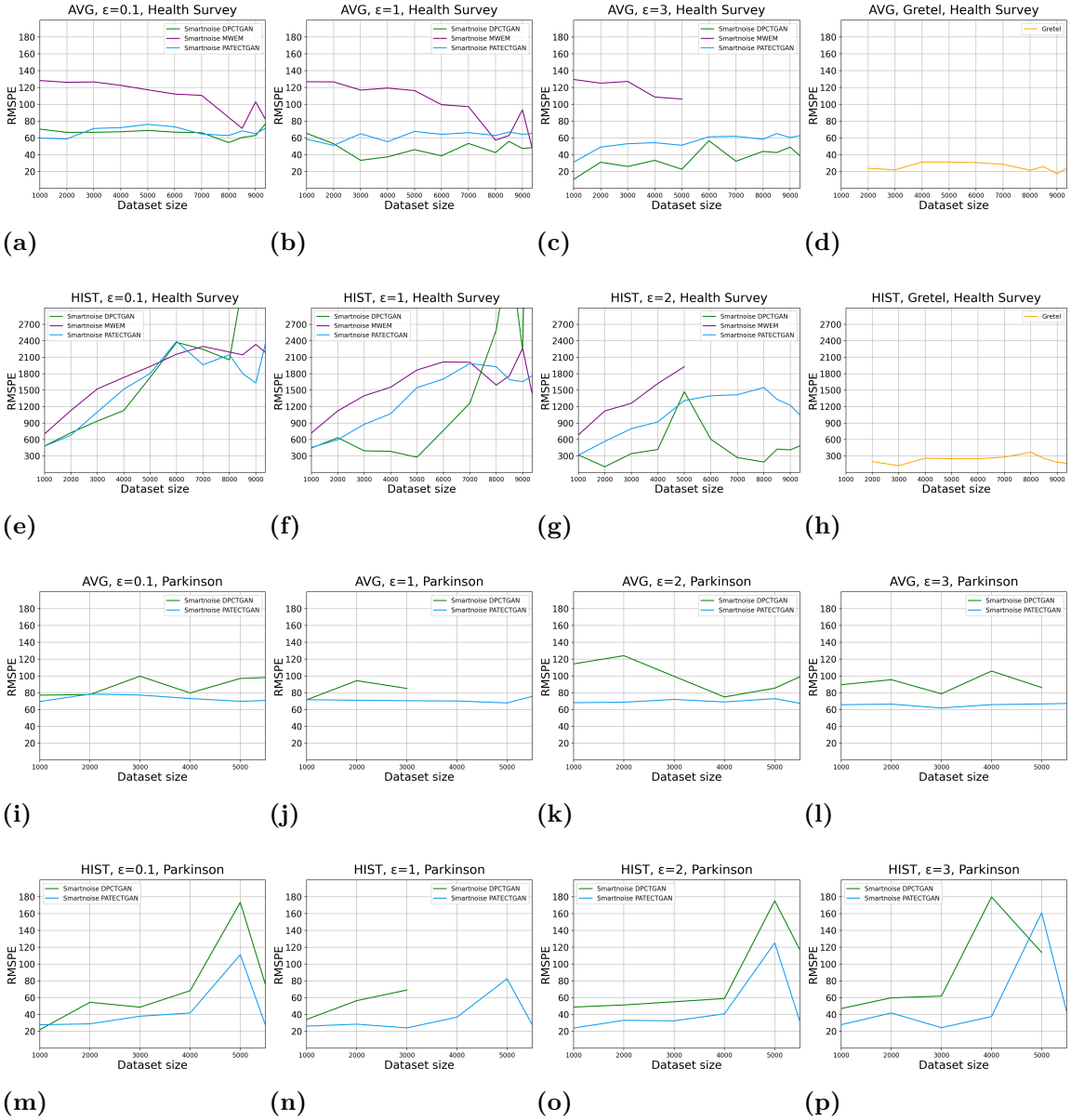

Figure 5.15: The results of experiment 7, showing the impact that DP has on data utility for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2), by conducting statistical queries on synthetic datasets. RMSPE is defined in Section 4.1. Note that Gretel Synthetics shows $\epsilon = 24.6$.

5.5.2 Experiment 8. Machine learning utility

Running machine learning (ML) tasks on synthetic datasets generated with differential privacy (DP), instead of running the tasks on the original non-privacy protected (NP) dataset, might have an inherent impact on the utility of the task. In experiment 8, we study that impact by comparing the performance of ML models trained on synthetic data vs ML models trained on the real data. The performance is defined as the RMSPE (Definition 4.1, Section 4.1) of the Mean Square Error (MSE) from the linear regression models. We train the ML model on the synthetic data generated based on 80 % of the NP data and use the remaining 20 % of the NP data to test the models. We also study how the performance impact varies between different tools for synthetic data generation (SDG).

Since higher ϵ values represent adding less noise to the data, and more data implies that each individual's contribution to the final result is smaller [22], we anticipate that synthetic datasets, generated with higher ϵ values and larger data size, will provide better utility on both datasets, i.e. *Health Survey* and *Parkinson*. As we detail below, from experiment 8, we can observe that higher data size tends to produce better utility on *Parkinson*, while we see no similar trend on *Health Survey*. We also see no trend regarding ϵ values on both datasets. Generally, Smartnoise PATECTGAN performs the best on both datasets. Moreover, both tools perform significantly worse on *Parkinson* than on *Health Survey*, which might imply that both tools perform better with categorical data for ML tasks.

Figure 5.16 shows contour plots for each tool's performance in regard to different ϵ and data size. Darker shades of blue in the plots indicate lower RMSPE, corresponding to higher data utility, while lighter shades indicate a larger RMSPE, corresponding to lower data utility. The white areas indicate that the tools failed to generate datasets for those combinations of ϵ and data size. Also note that the plots have different RMSPE scales, which means that a shade (which indicates an RMSPE value) in one plot does not necessarily correspond to the same shade in another plot. Moreover, since Gretel Synthetics does not explicitly provide functionality for targeting ϵ values, we could not generate datasets for each target ϵ (Table 4.3). We, therefore, omit the Gretel Synthetics results from Figure 5.16. Since Gretel Synthetics and Smartnoise MWEM fail to generate any datasets for Parkinson, these cases are also omitted from Figure 5.16.

In Figure 5.16 we can observe a trend for *Parkinson*, where larger data size provides better data utility. E.g., Smartnoise PATECTGAN has an RMSPE between about 800% and 1200% for the data size of 5499 rows and between about 2500% and 4000% for the data size of 1000 rows. A similar trend can also be seen for Smartnoise DPCTGAN on *Parkinson*, which is able to obtain better data utility for larger data size (Figure 5.16e). However, no trends are noticeable for any considered SDG tools on *Health Survey*.

In the line plots of Figure 5.17, we notice a trend where larger data size has a positive impact on data utility on *Parkinson*. However, the SDG tools obtain significantly

worse data utility on *Parkinson* than on *Health Survey*. In the plots for *Health Survey* (Figures 5.17a, 5.17b, 5.17c), we can see that Smartnoise PATECTGAN has an RMSPE between about 30% and 70%, DPCTGAN between about 20% and 120% and MWEM between about 30% and above 220%. When also taking the Figure 5.16 into consideration we can see that DPCTGAN has a couple of outliers around 400%. This means that Smartnoise PATECTGAN performs better than DPCTGAN, which is the opposite to what we saw in experiment 7 (Section 5.5.1), where Smartnoise DPCTGAN outperformed Smartnoise PATECTGAN on *Health Survey*. Moreover, Smartnoise MWEM performs reasonable well on *Health Survey*, and for ϵ values of 0.1 and 1 in Figures 5.17a and 5.17b, respectively. Smartnoise MWEM is the best performing tool for large data sizes on *Health survey*, especially when using an ϵ value of 1 where Smartnoise MWEM gets an RMSPE of between 20% and 40%, as can be seen in Figure 5.17f. Furthermore, Gretel Synthetics performs well on *Health Survey*. However, as previously mentioned (Section 5.5.1), Gretel Synthetics uses a significantly higher privacy budget ϵ.

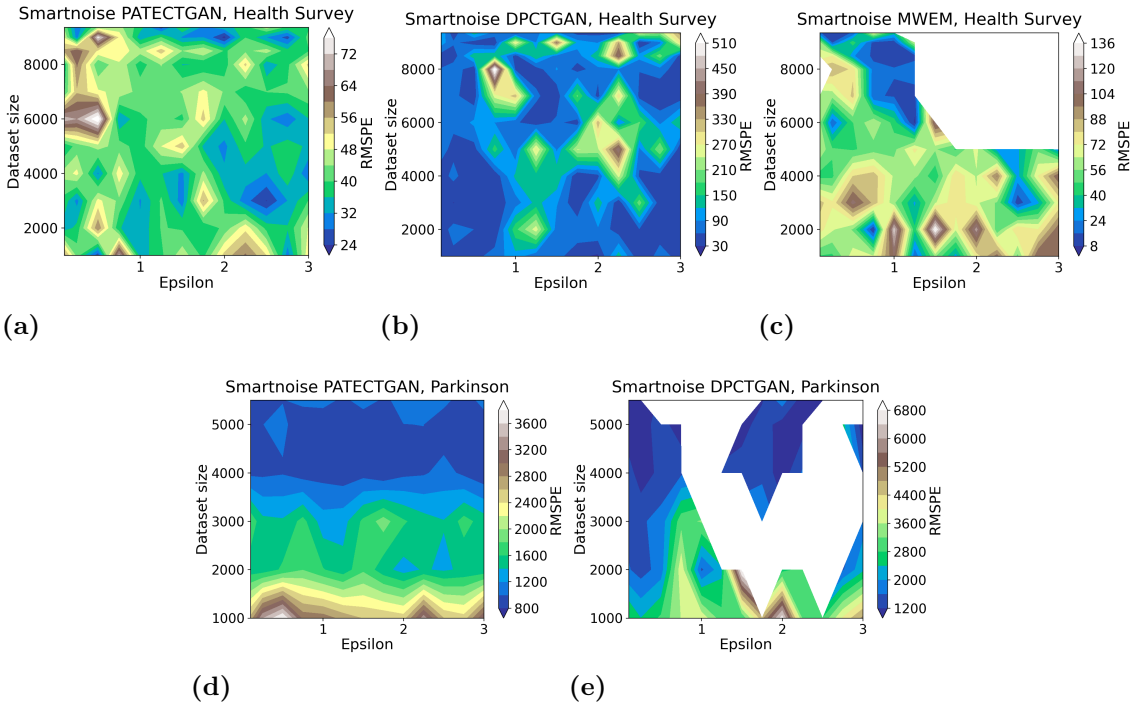

Figure 5.16: The results of experiment 8, showing the impact that DP has on data utility for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2), by conducting linear regression tasks on synthetic datasets. RMSPE is defined in Section 4.1.

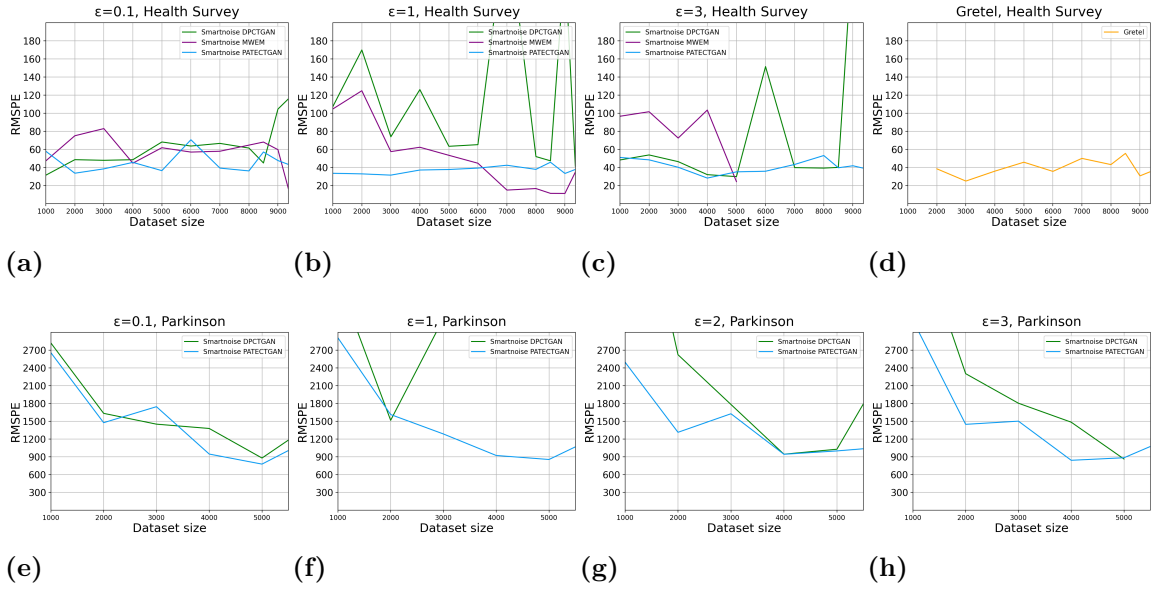

Figure 5.17: The results of experiment 7, showing the impact that DP has on data utility for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2), by conducting linear regression tasks on sythetic datasets. RMSPE is defined in Section 4.1. Note that Gretel Synthetics shows $\epsilon = 24.6$.

5.5.3 Experiment 9. Run-time overhead

Differentially private (DP) synthetic data generation (SDG) clearly takes additional time, compared to using the original dataset directly. In this experiment, we study how much time the generation process takes and how it compares between the tools.

For this experiment, we anticipate that the generation time for Smartnoise MWEM might be significantly less than the generation time of the other tools. The expectation is based on the literature [58], the original MWEM paper that introduced and evaluated the algorithm, and the results from a study [12] that implemented the algorithm in OpenDP's Smartnoise. We also anticipate that generating datasets with larger sizes takes additional time since intuitively more data has to be processed.

As detailed below, the experiment results show that contrary to our anticipation, Smartnoise MWEM takes significantly longer time than Smartnoise's other synthesizers (PATECTGAN, DPCTGAN), while larger datasets take longer time to generate than smaller ones, which was in line with our anticipations.

Figure 5.18 shows contour plots for each tool's run-time in regard to different ϵ and data size. Darker shades of blue in the plots indicate less time for SDG, while lighter shades indicate longer generation time. Note that the plots have different RMSPE scales, which means that a shade (which indicates an RMSPE value) in one plot does not necessarily correspond to the same shade in another plot. Moreover, since Gretel Synthetics does not explicitly provide functionality for targeting ϵ values, we could not generate datasets for each target ϵ (Table 4.3). We, therefore, omit the

Gretel Synthetics results from Figure 5.18. Since Gretel Synthetics and Smartnoise MWEM fail to generate any datasets for Parkinson, these cases are also omitted from Figure 5.18.

In Figure 5.18 we can observe a general trend, where larger privacy budgets (ϵ values) and larger data sizes increase the time it takes to generate the datasets. This is true for all the tools and datasets, except for Smartnoise DPCTGAN for *Parkinson* and except for a few irregularities. The reason why the value of ϵ affects the generation time is not clear to us, however, it is reasonable that it takes a longer time to process and generate larger datasets. Furthermore, the results also show that Smartnoise MWEM for *Health Survey* data takes the longest time for data size between 4000 and 6000 rows, up to 700 seconds, and that Smartnoise DPCTGAN for *Parkinson* takes an especially long time, about 550 seconds to generate with the smallest data size and $\epsilon = 3$, as shown in Figure 5.18e.

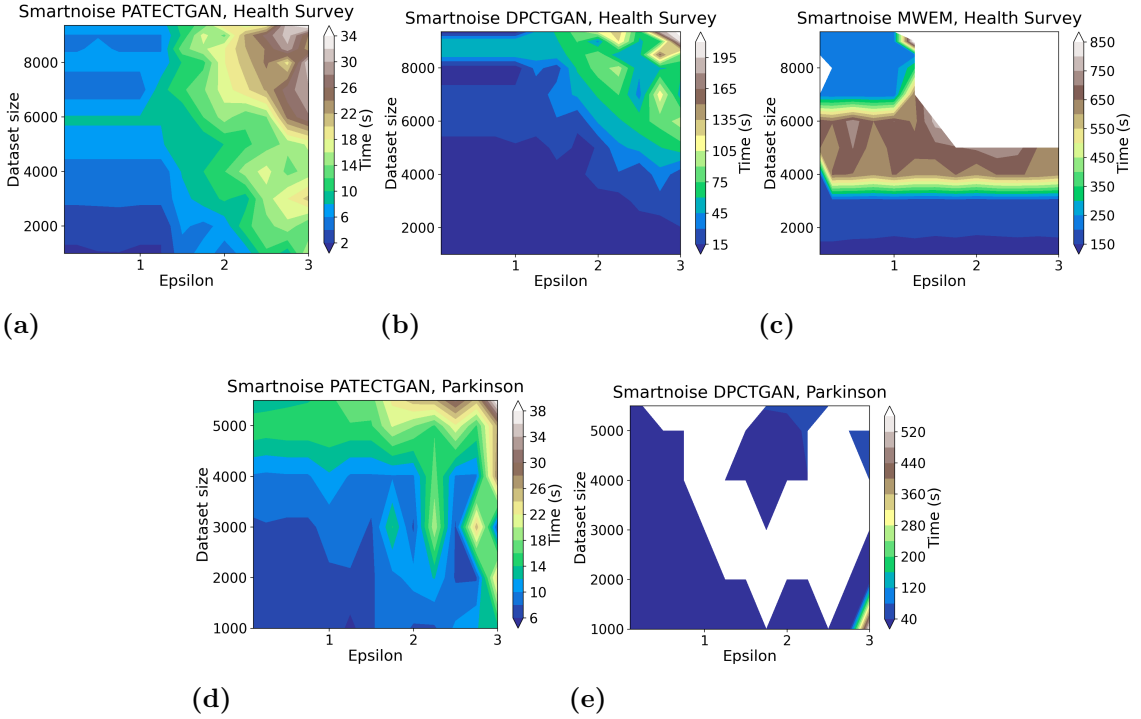

Figure 5.18: The results of experiment 9, showing the impact that DP has on the run-time for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1.

From Figure 5.19, we can observe that Smartnoise PATECTGAN and DPCTGAN both generate synthetic datasets in about 5 seconds (with some exceptions) for both of the datasets for values of ϵ smaller than 3, while it is hard to distinguish whether DPCTGAN or PATECTGAN is the best given ϵ less than 3. For the ϵ value of 3, generating datasets takes a longer time as the size of the dataset grows. For smaller datasets, it takes about 5 seconds for both the synthesizers on both the datasets. For the data size of 9358 rows on the *Health Survey* dataset, it takes about 200

seconds and 50 seconds for DPCTGAN and PATECTGAN respectively, and about 100 seconds and 50 seconds for DPCTGAN and PATECTGAN respectively for the 5499 rows of the *Parkinson* dataset, indicating an advantage of PATECTGAN over DPCTGAN on data synthesis of larger data size. Moreover, Smartnoise MWEM generates synthetic datasets between about 150 seconds and 700 seconds and Gretel Synthetics between 400 seconds and over 2000 seconds, which can be seen in Figure 5.19c. This means that Gretel Synthetics and Smartnoise MWEM are more time-consuming in general than the other tools.

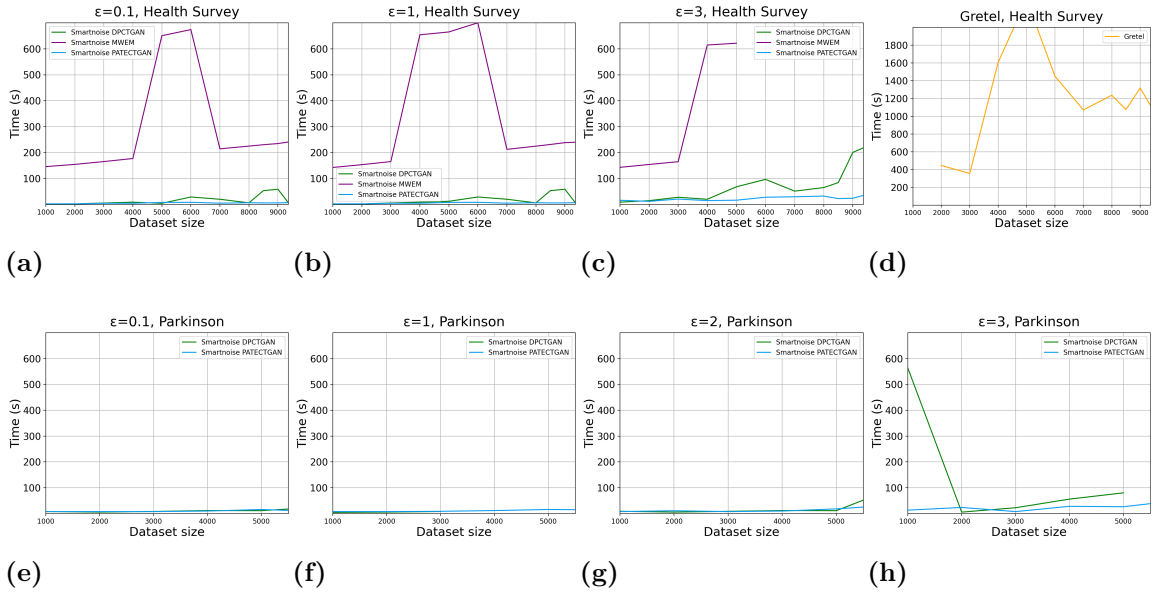

Figure 5.19: The results of experiment 9, showing the impact that DP has on data utility for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1. Note that Gretel Synthetics shows $\epsilon = 24.6$.

5.5.4 Experiment 10. Memory overhead

Differentially private (DP) synthetic data generation (SDG) might require significant memory usage. In experiment 10, we study the memory consumption of the Docker container that runs the SDG. We also study how the different tools compare regarding memory consumption.

For this experiment, we anticipate that larger data size will intuitively increase memory consumption during the process. Since all generated data is kept in memory during the generation process, before being written to disk at the end of the generation, more memory is expected to be consumed for larger data size. As detailed below, Smartnoise DPCTGAN and Smartnoise PATECTGAN on *Parkinson* behave as we anticipated. However, Smartnoise MWEM deviates from our expectations, which is also the case for Smartnoise PATECTGAN on *Health Survey*. Moreover, Smartnoise DPCTGAN has significantly lower memory consumption for smaller privacy

budgets (ϵ values). On the other hand, it performs worse than the other tools in general. We also observe that Smartnoise PATECTGAN and Smartnoise MWEM perform similarly on *Health Survey*, and Smartnoise PATECTGAN performs similarly on both datasets.

Figure 5.20 shows contour plots for each tool's memory consumption in regard to different ϵ and data size. Darker shades of blue in the plots represent higher memory consumption during the SDG, while lighter shades indicate lower memory consumption. The plots have different memory scales, which means that a shade in one plot does not necessarily correspond to the same shade in another plot. Moreover, since Gretel Synthetics does not explicitly provide functionality for targeting ϵ values, we could not generate datasets for each target ϵ (Table 4.3). We, therefore, omit the Gretel Synthetics results from Figure 5.20. Since Gretel Synthetics and Smartnoise MWEM fail to generate any datasets for Parkinson, these cases are also omitted from Figure 5.20.

In Figure 5.20, we can see a trend where both datasets, i.e., *Health Survey* and *Parkinson*, have a smaller memory consumption for smaller ϵ values. However, this trend does not apply for Smartnoise PATECTGAN on both datasets (Figures 5.20a and 5.20d) for ϵ values above 0.5. While for Smartnoise DPCTGAN (Figures 5.20b and 5.20e) the memory consumption continues to increase as the ϵ value increases. Moreover, in Figure 5.20b, 5.20d, and 5.20e, we observe that Smartnoise DPCTGAN, for both datasets, and Smartnoise PATECTGAN for *Parkinson*, consume more memory for larget data size, which is inline with our anticipations. However, we do not see this trend in Figure 5.20a and 5.20c.

From Figure 5.21, we can observe that Smartnoise MWEM and Smartnoise PATECTGAN for both *Health Survey* and *Parkinson* performs similarly, and has a consistent memory usage of about 200MB under different ϵ and data size. In comparison, Smartnoise DPCTGAN and Gretel Synthetics perform significantly worse than MWEM and PATECTGAN; DPCTGAN has a memory usage of between about 200MB and 1000MB while Gretel has a more or less consistent memory usage of 1700MB. This can be observed Figure 5.21. There is also a slight trend that memory cost tends to increase as data size and ϵ grows for Smartnoise DPCTGAN on both datasets.

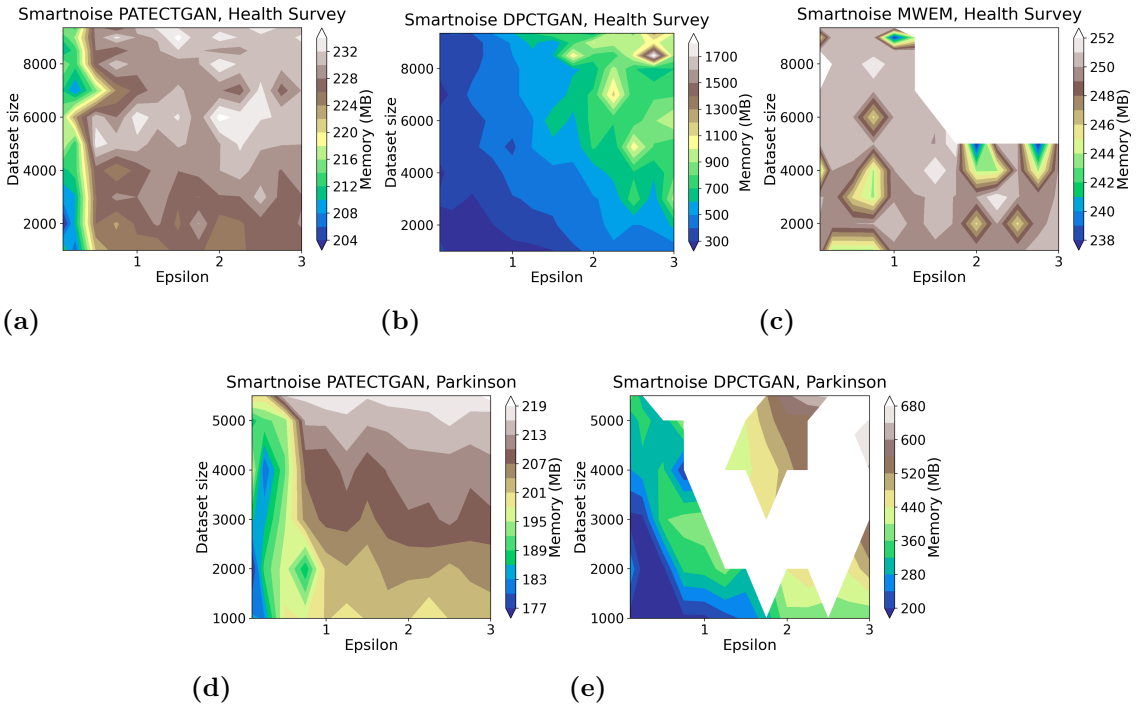

Figure 5.20: The results of experiment 10, showing the impact that DP has on memory overhead for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1 and and memory is measured in mega bytes (MB).

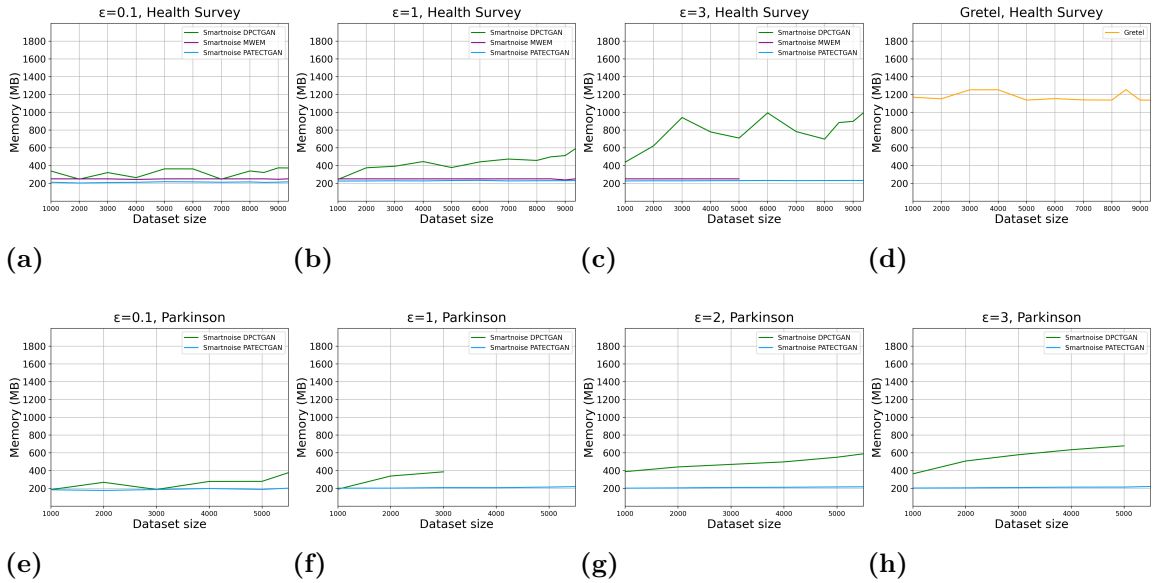

Figure 5.21: The results of experiment 10, showing the impact that DP has on memory overhead for different SDG tools, given different ϵ values (Table 4.3) and data size (Table 4.2). RMSPE is defined in Section 4.1 and memory is measured in mega bytes (MB). Note that Gretel Synthetics shows $\epsilon = 24.6$.

6
Conclusion

In this thesis, we build an experimental evaluation framework and show how to evaluate various differential privacy (DP) tools. Our framework works cross platforms, making it convenient to run and develop, locally and in the cloud. The architecture allows for implementing additional tools, and it can be leveraged by privacy service developers, to examine which DP tools can be used, given the data that is available in their organization.

In the evaluation, we define a set of criteria for the evaluation to quantify the answers to how different DP tools perform, and how they can be optimally configured. Specifically, we evaluate and measure the impact that DP has on different functionalities that the considered tools provide, i.e., statistical queries, machine learning, and synthetic data generation, and how the tools compare to each other. For the evaluation, we use two different data sources in the field of life-science, to obtain a nuanced picture of how well the considered tools perform when DP is applied. The evaluation results demonstrate (1) how much DP tools impact data utility and system overhead in DP applications, and (2) if DP service practitioners can apply these tools with a range of privacy settings.

Generally, our evaluation shows that DP has an inherent impact on data utility, and an apparent impact on system overhead (run-time and memory overhead). We conclude from the evaluation results that the selection of privacy tools should consider developer's purpose, what use-cases might fit them, how accurate the results they might expect to obtain given a certain type of dataset, and what impact the tools might have on system overhead.

Based on our evaluation, when using interactive statistical query tools (like Google DP and Smartnoise), we would recommend to cache query results for statistical query tools, to avoid spending the privacy budget and thus obtaining relatively accurate results. This solution requires a privacy budget bound that limits the number of queries that can be executed on the database, and simultaneously guarantee privacy.

Moreover, while the synthetic data tools offers great flexibility, allowing organizations to leverage the data across different fields of data analysis, it might not offer desirable data utility for data sources across different organizations. However, it is a vibrant field that has shown improvements. Since it is a fundamental challenge to protect individual privacy and obtain adequate results, organizations need to work

towards adjusting suitable data pipelines in the direction of the most promising field of privacy.

Convincingly, DP is currently still the best formal guarantee of individual privacy. However, there are no shortcuts when it comes to privacy-utility trade-offs. I.e., there is a cost for providing privacy that is deeply rooted withing the fabrics of the correlation between privacy and utility. When it comes to statistical queries and machine learning, the considered tools have been fine-tuning the privacy-utility balance, in order to achieve relatively good performance. However, DP synthetic data generation has still a long way to go, and research in how DP tools perform is therefore important. Valuable work has been done in the field, fairly recently, that has gained momentum in the developers' community and industry. It is a crucial field that might allow open-sourcing privacy-sensitive data in the future, ultimately allowing life-science as well as other organizations to collaborate on a global level.

Bibliography

[1] W. N. Price and I. G. Cohen, "Privacy in the age of medical big data," *Nature medicine*, vol. 25, no. 1, pp. 37–43, 2019.

[2] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health Information Science and Systems*, vol. 2, no. 1, 12 2014.

[3] Data protection in the EU. Accessed 2021-02-20. [Online]. Available: https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en

[4] A. Kopp, "Microsoft smartnoise differential privacy machine learning case studies," 2021, Accessed 2021-11-05. [Online]. Available: https://azure.microsoft.com/en-us/resources/microsoft-smartnoisedifferential-privacy-machine-learning-case-studies/

[5] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–487, 2013.

[6] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee, "Toward privacy in public databases," *Lecture Notes in Computer Science*, vol. 3378, pp. 363–385, 2005.

[7] G. J. Kerns and G. J. Székely, "Definetti's theorem for abstract finite exchangeable sequences," *Journal of Theoretical Probability*, vol. 19, no. 3, pp. 589–608, 2006.

[8] D. Kifer, "Attacks on privacy and definetti's theorem," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 127–138.

[9] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2003, pp. 202–210.

[10] M. U. Hassan, M. H. Rehmani, and J. Chen, "Differential Privacy Techniques for Cyber Physical Systems: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 22, no. 1, pp. 746–789, 2020.

[11] L. van der Maaten and A. Hannun, "The trade-offs of private prediction," *arXiv preprint arXiv:2007.05089*, 2020.

[12] L. Rosenblatt, X. Liu, S. Pouyanfar, E. de Leon, A. Desai, and J. Allen, "Differentially private synthetic data: Applied evaluations and enhancements," *arXiv preprint arXiv:2011.05537*, 2020.

[13] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau, "Ektelo: A framework for defining differentially-private computations," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 115–130.

[14] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1895–1912.

[15] D. Knoors, "Utility of differentially private synthetic data generation for high-dimensional databases," Master's thesis, KTH Royal Institute of Technology, 2018.

[16] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang, "Principled evaluation of differentially private algorithms using DPBENCH," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 26-June-20. Association for Computing Machinery, 6 2016, pp. 139–154.

[17] T. Ala Hadi, "Differential privacy: an extensive evaluation of open-source tools," Master's thesis, Chalmers University of Technology, 2021.

[18] N. Johnson, J. P. Near, and D. Song, "Towards practical differential privacy for SQL queries," *Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 526–539, 2018.

[19] Dataflow analysis & differential privacy for SQL queries. Accessed 2021-06-03. [Online]. Available: https://github.com/uber-archive/sql-differential-privacy

[20] Enhance Privacy, Security Tools with Microsoft's New Program. Accessed 2021-05-27. [Online]. Available: https://www.ciobulletin.com/security/microsoft-new-program-enhances-privacy

[21] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 19–30.

[22] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson, "Differentially private SQL with bounded user contribution," 9 2019.

[23] M. Abadi, H. B. McMahan, A. Chu, I. Mironov, L. Zhang, I. Goodfellow, and K. Talwar, "Deep learning with differential privacy," *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 24-28-Octo, pp. 308–318, 2016.

[24] 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," *arXiv preprint arXiv:2102.03013*, 2021.

[25] P. Subramani, N. Vadivelu, and G. Kamath, "Enabling fast differentially private SGD via just-in-time compilation and vectorization," *arXiv preprint arXiv:2010.09063*, 2020.

[26] F. Tramèr and D. Boneh, "Differentially private learning needs better features (or much more data)," *arXiv preprint arXiv:2011.11660*, 2020.

[27] I. Mironov, "Rényi Differential Privacy," *Proceedings - IEEE Computer Security Foundations Symposium*, pp. 263–275, 2017.

[28] L. Rosenblatt, X. Liu, S. Pouyanfar, E. de Leon, A. Desai, and J. Allen, "Differentially Private Synthetic Data: Applied Evaluations and Enhancements," 11 2020. [Online]. Available: http://arxiv.org/abs/2011.05537

[29] E. M. S. Shlomi Dolev, Thomas Petig, "Self-stabilizing and private distributed shared atomic memory in seldomly fair message passing networks," *CoRR*, vol. abs/1806.03498, 2018.

[30] ——, "Brief announcement: Robust and private distributed shared atomic memory in message passing networks," in *PODC*. ACM, 2015, pp. 311–313.

[31] T. Dalenius, "Towards a methodology for statistical disclosure control," *Statistik Tidskrift*, vol. 15, no. 429-444, 1977.

[32] ——, "Privacy transformations for statistical information systems," *Journal of Statistical Planning and Inference*, vol. 1, no. 1, pp. 73–86, 1977.

[33] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[34] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[35] D. Desfontaines and B. Pejó, "SoK differential privacies," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 2, pp. 288–313, 2020.

[36] F. Prasser and F. Kohlmayer, *Putting Statistical Disclosure Control into Practice: The ARX Data Anonymization Tool*. Cham: Springer International Publishing, 2015, pp. 111–148. [Online]. Available: https://doi.org/10.1007/978-3-319-23633-9_6

[37] L. Rocher, J. M. Hendrickx, and Y.-A. De Montjoye, "Estimating the success of re-identifications in incomplete datasets using generative models," *Nature communications*, vol. 10, no. 1, pp. 1–9, 2019.

[38] Art. 4 GDPR. Accessed 2021-06-04. [Online]. Available: https://gdpr-info.eu/art-4-gdpr/

[39] N. Gruschka, V. Mavroeidis, K. Vishi, and M. Jensen, "Privacy issues and data protection in big data: a case study analysis under GDPR," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5027–5033.

[40] Recital 26 - Not Applicable to Anonymous Data | General Data Protection Regulation (GDPR). Accessed 2020-10-01. [Online]. Available: https://gdpr-info.eu/recitals/no-26/

[41] B. Nelson, "Differential privacy-a balancing act," Ph.D. dissertation, Chalmers University of Technology, jun 2021.

[42] Apple Inc., "Apple Differential Privacy Technical Overview," p. 3, 2017. [Online]. Available: https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf

[43] T. O. Team, "The OpenDP White Paper," 2020. [Online]. Available: https://projects.iq.harvard.edu/files/opendp/files/opendp_white_paper_11may2020.pdf

[44] T. Zhu, G. Li, W. Zhou, and S. Y. Philip, "Differentially private data publishing and analysis: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 8, pp. 1619–1638, 2017.

[45] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: Regression analysis under differential privacy," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1364–1375, 2012.

[46] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 2015-October. Association for Computing Machinery, 10 2015, pp. 1322–1333.

[47] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, pp. 1069–1109, 2011.

[48] C. Chen, J. Lee, and D. Kifer, "Rényi differentially private ERM for smooth objectives," *AISTATS 2019 - 22nd International Conference on Artificial Intelligence and Statistics*, vol. 89, 2020.

[49] M. Abadi, A. Chu, I. Goodfellow, B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *23rd ACM Conference on Computer and Communications Security (ACM CCS)*, I2016, pp. 308–318. [Online]. Available: https://arxiv.org/abs/1607.00133

[50] I. Goodfellow, "Efficient Per-Example Gradient Computations," vol. 1, pp. 2–3, 2015. [Online]. Available: http://arxiv.org/abs/1510.01799

[51] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro, "LOGAN: Membership Inference Attacks Against Generative Models," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 1, pp. 133–152, 2019.

[52] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep Models under the GAN: Information leakage from collaborative deep learning," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 603–618, 2017.

[53] J. Zhao, Y. Chen, and W. Zhang, "Differential Privacy Preservation in Deep Learning: Challenges, Opportunities and Solutions," *IEEE Access*, vol. 7, pp. 48 901–48 911, 2019.

[54] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, "Differentially private generative adversarial network," *arXiv preprint arXiv:1802.06739*, 2018.

[55] M. Alzantot, S. Chakraborty, and M. Srivastava, "SenseGen: A deep learning architecture for synthetic sensor data generation," *2017 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2017*, pp. 188–193, 2017.

[56] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1071–1083, 2018.

[57] S. Mukherjee, Y. Xu, A. Trivedi, and J. L. Ferres, "privgan: Protecting gans from membership inference attacks at low cost," *arXiv preprint arXiv:2001.00071*, 2019.

[58] M. Hardt, K. Ligett, and F. McSherry, "A simple and practical algorithm for differentially private data release," *arXiv preprint arXiv:1012.4763*, 2010.

[59] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional GAN," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, 2019.

[60] Gretel Synthetics. Accessed 2021-05-27. [Online]. Available: https://github.com/gretelai/gretel-synthetics#differential-privacy

[61] Gretel-synthetics FAQs. Accessed 2021-06-03. [Online]. Available: https://gretel.ai/blog/gretel-synthetics-faqs

[62] Harvard University Privacy Tools Project. Accessed 2021-01-27. [Online]. Available: https://privacytools.seas.harvard.edu/

[63] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth, "Differential privacy: An economic method for choosing epsilon," *Proceedings of the Computer Security Foundations Workshop*, vol. 2014-Janua, pp. 398–410, 2014.

[64] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay, "Architecting a differentially private SQL engine." in *CIDR*, 2019.

[65] A. Haeberlen, B. C. Pierce, and A. Narayan, "Differential privacy under fire," *Proceedings of the 20th USENIX Security Symposium*, pp. 507–521, 2011.

[66] D. Zhang and D. Kifer, "LightDP: Towards automating differential privacy proofs," *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pp. 888–901, 2017.

[67] smartnoise-core. Accessed 2021-03-10. [Online]. Available: https://github.com/opendifferentialprivacy/smartnoise-core

[68] smartnoise-sdk. [Accessed 2021-01-10]. [Online]. Available: https://github.com/opendifferentialprivacy/smartnoise-sdk

[69] M. Guevara, "Enabling developers and organizations to use differential privacy. developers. googleblog. com/2019/09/enabling-developers-and-organizations. html, 2019," 2019.

[70] Privacy Loss Distributions, Differential Privacy Team Google. Accessed 2021-06-02. [Online]. Available: https://github.com/google/differential-privacy/blob/main/common_docs/Privacy_Loss_Distributions.pdf

[71] Opacus API Reference. Accessed 2021-06-03. [Online]. Available: https://opacus.ai/api/index.html

[72] H. B. McMahan, G. Andrew, U. Erlingsson, S. Chien, I. Mironov, N. Papernot, and P. Kairouz, "A general approach to adding differential privacy to iterative training procedures," *arXiv preprint arXiv:1812.06210*, 2018.

[73] Machine Learning with Differential Privacy in TensorFlow. Accessed 2021-05-27. [Online]. Available: https://github.com/tensorflow/privacy/blob/693dd666c3c05ec09dbc361f303bb045207b8163/tutorials/walkthrough/README.md

[74] Gradient Clipping. Accessed 2021-05-27. [Online]. Available: https://opacus.ai/api/per_sample_gradient_clip.html

[75] Openminded. Openminded.org Website. Accessed 2021-06-04. [Online]. Available: https://www.openmined.org/

[76] H. B. McMahan and G. Andrew, "A general approach to adding differential privacy to iterative training procedures," *CoRR*, vol. abs/1812.06210, 2018. [Online]. Available: http://arxiv.org/abs/1812.06210

[77] Diffprivlib. Accessed 2021-06-02. [Online]. Available: https://github.com/IBM/differential-privacy-library

[78] SDGYMBaseSynthesizer. Accessed 2021-05-10. [Online]. Available: https://github.com/opendp/smartnoise-sdk/blob/a8740d62d7dfb44d0eb7fe02ff4c4a7ef734f9c1/sdk/opendp/smartnoise/synthesizers/base.py

[79] CTGAN. Accessed 2021-06-06. [Online]. Available: https://github.com/sdv-dev/CTGAN

[80] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 10 2006.

[81] Unified Parkinson's disease rating scale. Accessed 2021-06-04. [Online]. Available: https://en.wikipedia.org/wiki/Unified_Parkinson%27s_disease_rating_scale

[82] A. Wood, M. Altman, A. Bembenek, M. Bun, M. Gaboardi, J. Honaker, K. Nissim, D. O'Brien, T. Steinke, and S. Vadhan, "Differential Privacy: A Primer for a Non-Technical Audience," *SSRN Electronic Journal*, no. 1237235, 2019.

[83] What is differential privacy in machine learning. Accessed 2021-05-26. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/concept-differential-privacy#differential-privacy-metrics

[84] Maxima and minima. Accessed 2021-06-04. [Online]. Available: https://en.wikipedia.org/wiki/Maxima_and_minima

[85] Contour Plot. Accessed 2021-05-27. [Online]. Available: https://www.itl.nist.gov/div898/handbook/eda/section3/contour.htm

[86] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

A

Appendix 1

A.1 Algorithms

Algorithm 2 and 3 show the method for evaluating SQ and ML tools respectively. These methods are also applied for the evaluation of the SD tools. Additionally, since Opacus and Tensorflow Privacy (TFP) do not provide a parameter for ϵ, target epsilons have to be computed manually, as shown in algorithm 1. Specifically, we illustrate how `calc_noise_multipliers` of Opacus and TFP work in Algorithm 3 by applying a binary search approach in order compute a `noise_multiplier` (nm), corresponding to a target ϵ value. This computation depends on the settings of `sample_size`, `batch_size` and `epochs` in Opacus and TFP. Each target ϵ is found by narrowing down the difference of the computed ϵ values for each nm below a threshold of 0.01. This method is applied for Gretel as well, since Gretel uses TFP for training.

Algorithm 1
Algorithm for computing a `noise_multiplier` (nm) corresponding to target ϵ value. Note that H stands for high, L for low, and M for mid.

ϵ_L := `calc_epsilon`(`sample_size`, `batch_size`, `epochs`, nm_L)
ϵ_H := `calc_epsilon`(..., nm_H)
while ϵ_H - $\epsilon_L < 0.01$ **do**
  $\text{nm}_M := (\text{nm}_H + \text{nm}_L) / 2$
  ϵ_M := `calc_epsilon`(..., nm_M)
  if $\epsilon_M <=$ target ϵ **then**
    $\text{nm}_L := \text{nm}_M$
    $\epsilon_L := \epsilon_M$
  else
    $\text{nm}_H := \text{nm}_M$
    $\epsilon_H := \epsilon_M$
  end if
end while

Algorithm 2

Algorithms for evaluating statistical query tools: **1)** Each query q runs for **2)** all columns in the dataset. **3)** Each q is then parsed for all combinations *combs* of data size and ϵ. **4)** For each run and all *combs*, bounds are appended and **5)** q is executed on a PostgresSQL database with a set s of the dataset sizes

```
for q in {SUM, SOUNT, AVG, HISTOGRAM} do
  for c in columns ∈ dataset do
    q := parse(q, c)
    for s in {1000, 2000, ..., size(dataset)} do
      for ϵ in {0.1, ...} do
        for i in number of runs do
          q := parse(q, s, ϵ)
          if ϵ = −1 then
            res := run q
          else
            res := run DP q
          end if
          write results := <q, s, ϵ, res, run-time>
        end for
      end for
    end for
  end for
end for
```

Algorithm 3

Algorithm for evaluating ML tools: **1)** For each sub-dataset $ds \in$ dataset, with size $s \in \{1000, 2000, ..., \text{size(dataset)}\}$, we compute a `noise_multiplier` nm for all target $\epsilon \in \{0.1, 0.25, 0.5, ..., 3.0\}$. **2)** For each nm, **3)** for all number of runs, a model is trained for every combination of s and nm, corresponding to target ϵ.

```
target ϵ ∈ {0.1, 0.25, 0.5, ..., 3.0}
dpu := DPUtils(batch_size, target ϵ, epochs)
for ds in dataset, ds ⊂ dataset do
  split and normalize ds
  nms := dpu.calc_noise_multipliers(size(ds), δ)
  for nm in nms ∪ {−1} do
    for i in number of runs do
      if nm = −1 then
        model := build ML model(...)
      else
        model := build DP-ML model(..., nm)
      end if
      loss := train model
      write results := <s, ϵ, loss, run-time, ...>
    end for
  end for
end for
```

A.2 Queries

SQL queries, conducted for the evaluation of statistical query tools. Note that `col_name` is replaced by the column name, since queries are conducted on all columns in the dataset. Likewise, `%s` is replaced by the data size, for each data size that the specific query are conducted on.

```
HISTOGRAM =
 "SELECT COUNT({col_name}) AS result,
  histogram_{col_name} AS query,
  {col_name} AS res_bin FROM data_%s
  GROUP BY {col_name}
  ORDER BY res_bin"

SUM =
 "SELECT SUM({col_name}) AS result,
  sum_{col_name} AS query FROM data_%s"

COUNT =
 "SELECT COUNT({col_name}) AS result,
  count_{col_name} AS query FROM data_%s"

AVG =
 "SELECT AVG({col_name}) AS result,
  avg_{col_name} AS query FROM data_%s"
```

A.3 Experiment Settings

Below we list experiment settings, e.g., experiment parameter and hyperparameter values for different experiments. Full list of ϵ values and dataset sizes can be seen in Table 4.3 and Table 4.2 respectively. Note that Gretel Synthetics (Table 5.1) does not contain ϵ values since it does not provide functionality for targeting specific ϵ values.

Parameter	**Datasets**	
	Health survey	Parkinson
Queries	{`SUM`, `COUNT`, `AVG`, `HISTOGRAM`}	—"—
Epsilon (ϵ)	{0.1, 0.25, ..., 3.0}	—"—
Dataset Size	{1000, 2000, ..., size(dataset)}	—"—

Table A.1: Experiment parameter and hyperparameter values for Smartnoise and Google DP experiments.

Parameter	**Datasets**	
	Health survey	Parkinson
Epsilon (ϵ)	{0.1, 0.25, ..., 3.0}	—"—
Dataset Size	{1000, 2000, ..., size(dataset)}	—"—
Model	Linear Regression	—"—
Optimizer	(`DPKerasSGDOptimizer`, `SGD`)	—"—
Target	"q1"	"total_updrs"
Epochs	30	—"—
Batch size	4	—"—
Learning Rate	(0.01, 0.001)	—"—
Gradient Clipping Norm	(1.0, None)	—"—
delta (δ)	(1/(sample size $\cdot$ 2), None)	—"—

Table A.2: Experiment parameter and hyperparameter values for Opacus and Tensorflow Privacy experiments. Note that the optimizer, learning rate, gradient clipping norm and δ variables are (differentially private, NP).

Parameter	Datasets	
	Health survey	Parkinson
Epsilon (ϵ)	{0.1, 0.25, ..., 3.0}	—"—
Dataset Size	{1000, 2000, ..., size(dataset)}	—"—
Model	Linear Regression	—"—
Target	"q1"	"total_updrs"

Table A.3: Experiment parameter and hyperparameter values for Diffprivlib experiments.

A.3.1 Hyperparameters for synthetic datasets

ϵ	Dataset size	Batch size	Epochs
0.10	8000	500	500
0.10	1000	500	300
0.10	2000	500	300
0.10	3000	500	300
0.10	4000	500	300
0.10	5000	500	300
0.10	6000	500	300
0.10	7000	500	300
0.10	9000	500	300
0.10	9358	500	300
0.25	2000	500	300
0.25	5000	300	300
0.25	9358	500	500
0.25	1000	500	300
0.25	3000	500	300
0.25	4000	500	300
0.25	6000	500	300
0.25	7000	500	300
0.25	8000	500	300
0.25	9000	500	300
0.50	1000	500	300
0.50	2000	500	300
0.50	3000	500	300
0.50	4000	500	300
0.50	5000	500	300
0.50	6000	500	300
0.50	7000	500	300
0.50	8000	500	300

0.50	9000	500	300
0.50	9358	500	300
0.75	1000	300	300
0.75	2000	500	300
0.75	3000	500	300
0.75	4000	500	300
0.75	5000	500	300
0.75	6000	500	300
0.75	7000	500	300
0.75	8000	500	300
0.75	9000	500	300
0.75	9358	500	300
1.00	3000	300	100
1.00	4000	300	300
1.00	5000	300	500
1.00	7000	300	100
1.00	1000	500	300
1.00	2000	500	300
1.00	6000	500	300
1.00	8000	500	300
1.00	9000	500	300
1.00	9358	500	300
1.25	8000	300	100
1.25	9358	300	100
1.25	1000	500	300
1.25	2000	500	300
1.25	3000	500	300
1.25	4000	500	300
1.25	5000	500	300
1.25	6000	500	300
1.25	7000	500	300
1.25	9000	500	300
1.50	1000	300	100
1.50	2000	300	100
1.50	3000	300	500
1.50	4000	500	100
1.50	5000	300	300
1.50	6000	300	100
1.50	7000	300	300
1.50	8000	500	100

1.50	9000	300	500
1.50	9358	300	300
1.75	1000	500	500
1.75	2000	300	500
1.75	3000	500	300
1.75	4000	500	100
1.75	5000	500	300
1.75	6000	500	500
1.75	7000	300	100
1.75	8000	300	300
1.75	9000	300	300
1.75	9358	300	500
2.00	1000	300	100
2.00	2000	300	500
2.00	3000	500	300
2.00	4000	300	500
2.00	5000	500	500
2.00	6000	300	500
2.00	7000	300	500
2.00	8000	300	100
2.00	9000	500	100
2.00	9358	300	100
2.25	1000	300	300
2.25	2000	500	100
2.25	3000	500	100
2.25	4000	300	300
2.25	5000	300	500
2.25	6000	300	300
2.25	7000	300	300
2.25	8000	300	300
2.25	9000	300	500
2.25	9358	300	500
2.50	1000	300	100
2.50	2000	500	500
2.50	3000	300	300
2.50	4000	500	500
2.50	5000	300	100
2.50	6000	300	500
2.50	7000	500	100
2.50	8000	500	100

2.50	9000	500	500
2.50	9358	500	100
2.75	1000	300	300
2.75	2000	500	300
2.75	3000	300	300
2.75	4000	300	500
2.75	5000	300	500
2.75	6000	300	100
2.75	7000	300	500
2.75	8000	500	300
2.75	9000	500	500
2.75	9358	300	500
3.00	1000	300	500
3.00	2000	300	500
3.00	3000	300	100
3.00	4000	500	300
3.00	5000	300	500
3.00	6000	300	500
3.00	7000	500	100
3.00	8000	500	100
3.00	9000	300	300
3.00	9358	300	500

Table A.4: Hyperparameter values for generating synthetic datasets with Smartnoise DPCTGAN on *Health Survey*

ϵ	**Dataset size**	**Batch size**	**Epochs**
0.10	1000	300	300
0.10	2000	500	300
0.10	3000	500	300
0.10	4000	500	300
0.10	5000	500	300
0.10	6000	500	300
0.10	7000	500	300
0.10	8000	500	300
0.10	9000	500	300
0.10	9358	500	300
0.25	2000	300	500
0.25	7000	500	300
0.25	9000	500	500
0.25	1000	500	300

0.25	3000	500	300
0.25	4000	500	300
0.25	5000	500	300
0.25	6000	500	300
0.25	8000	500	300
0.25	9358	500	300
0.50	2000	300	300
0.50	9358	500	300
0.50	1000	500	300
0.50	3000	500	300
0.50	4000	500	300
0.50	5000	500	300
0.50	6000	500	300
0.50	7000	500	300
0.50	8000	500	300
0.50	9000	500	300
0.75	1000	500	100
0.75	2000	500	100
0.75	8000	500	300
0.75	3000	500	300
0.75	4000	500	300
0.75	5000	500	300
0.75	6000	500	300
0.75	7000	500	300
0.75	9000	500	300
0.75	9358	500	300
1.00	4000	500	300
1.00	1000	500	300
1.00	2000	500	300
1.00	3000	500	300
1.00	5000	500	300
1.00	6000	500	300
1.00	7000	500	300
1.00	8000	500	300
1.00	9000	500	300
1.00	9358	500	300
1.25	3000	500	500
1.25	5000	300	100
1.25	7000	300	300
1.25	1000	500	300

1.25	2000	500	300
1.25	4000	500	300
1.25	6000	500	300
1.25	8000	500	300
1.25	9000	500	300
1.25	9358	500	300
1.50	1000	300	500
1.50	2000	300	500
1.50	3000	300	300
1.50	4000	300	300
1.50	5000	300	500
1.50	6000	500	300
1.50	7000	300	100
1.50	8000	500	300
1.50	9000	500	300
1.50	9358	500	300
1.75	1000	300	500
1.75	2000	500	500
1.75	3000	300	500
1.75	4000	300	100
1.75	5000	500	500
1.75	6000	300	300
1.75	7000	300	300
1.75	8000	500	300
1.75	9000	300	100
1.75	9358	500	500
2.00	1000	500	300
2.00	2000	500	100
2.00	3000	300	500
2.00	4000	300	300
2.00	5000	500	500
2.00	6000	300	300
2.00	7000	300	300
2.00	8000	300	100
2.00	9000	500	500
2.00	9358	300	100
2.25	1000	500	500
2.25	2000	300	100
2.25	3000	500	300
2.25	4000	300	300

2.25	5000	500	300
2.25	6000	500	500
2.25	7000	300	500
2.25	8000	300	100
2.25	9000	300	500
2.25	9358	500	500
2.50	1000	300	100
2.50	2000	300	100
2.50	3000	300	500
2.50	4000	300	300
2.50	5000	500	500
2.50	6000	500	300
2.50	7000	300	500
2.50	8000	300	100
2.50	9000	300	500
2.50	9358	500	500
2.75	1000	500	300
2.75	2000	300	100
2.75	3000	300	100
2.75	4000	500	300
2.75	5000	500	500
2.75	6000	300	500
2.75	7000	300	500
2.75	8000	500	100
2.75	9000	300	100
2.75	9358	300	500
3.00	1000	300	100
3.00	2000	500	300
3.00	3000	300	300
3.00	4000	500	300
3.00	5000	500	300
3.00	6000	300	100
3.00	7000	300	100
3.00	8000	300	100
3.00	9000	500	500
3.00	9358	300	100

Table A.5: Hyperparameter values for generating synthetic datasets with Smartnoise PATECTGAN on *Health Survey*

ϵ	**Dataset size**	**Query count**
0.10	4000	200
0.10	5000	200
0.10	6000	400
0.10	1000	400
0.10	2000	400
0.10	3000	400
0.10	7000	400
0.10	9000	400
0.10	9358	400
0.25	1000	600
0.25	2000	600
0.25	3000	600
0.25	4000	400
0.25	5000	600
0.25	6000	600
0.25	7000	400
0.25	8000	400
0.25	9000	400
0.25	9358	400
0.50	2000	200
0.50	3000	600
0.50	4000	400
0.50	5000	200
0.50	6000	400
0.50	1000	400
0.50	7000	400
0.50	8000	400
0.50	9000	400
0.50	9358	400
0.75	1000	200
0.75	3000	400
0.75	4000	200
0.75	5000	200
0.75	6000	400
0.75	2000	400
0.75	7000	400
0.75	8000	400
0.75	9000	400
0.75	9358	400

1.00	1000	600
1.00	2000	200
1.00	4000	200
1.00	5000	200
1.00	6000	400
1.00	3000	400
1.00	7000	400
1.00	8000	400
1.00	9000	400
1.00	9358	400
1.25	1000	600
1.25	3000	200
1.25	4000	600
1.25	5000	400
1.25	6000	200
1.25	2000	400
1.25	7000	400
1.25	8000	400
1.25	9000	400
1.50	1000	600
1.50	2000	200
1.50	3000	200
1.50	4000	400
1.50	5000	200
1.50	6000	200
1.75	1000	200
1.75	2000	200
1.75	3000	200
1.75	4000	400
1.75	5000	600
2.00	1000	400
2.00	2000	200
2.00	3000	200
2.00	4000	400
2.00	5000	200
2.25	1000	200
2.25	2000	200
2.25	3000	200
2.25	4000	400
2.25	5000	200

2.50	1000	200
2.50	2000	200
2.50	3000	400
2.50	4000	200
2.50	5000	200
2.75	1000	200
2.75	2000	200
2.75	3000	200
2.75	4000	200
2.75	5000	200
3.00	1000	200
3.00	2000	400
3.00	3000	200
3.00	4000	200
3.00	5000	200

Table A.6: Hyperparameter values for generating synthetic datasets with Smartnoise MWEM on *Health Survey*

ϵ	**Dataset size**	**Predict batch size**	**RNN units**
2000	24.6	1.0	32
3000	32.6	1.0	32
4000	32.0	1.0	32
5000	31.0	1.0	32
6000	30.0	1.0	32
7000	30.0	1.0	32
8000	29.0	1.0	32
9000	28.0	1.0	32
9358	55.3	1.0	32

Batch size	**Learning rate**	**Noise multiplier**	**l2 norm clip**
4.0	0.005	0.1	0.1
4.0	0.005	0.1	0.1
4.0	0.005	0.1	0.1
4.0	0.005	0.3	0.1
4.0	0.005	0.3	0.1
4.0	0.005	0.3	0.1
4.0	0.005	0.3	0.1
4.0	0.005	0.3	0.1
4.0	0.005	0.3	0.1
4.0	0.005	0.3	0.1

4.0	0.001	0.3	0.1

Table A.7: Hyperparameter values for generating synthetic datasets with Gretel on *Health Survey*

ϵ	**Dataset size**	**Batch size**	**Epochs**
0.10	1000	300	300
0.10	3000	300	300
0.10	4000	300	300
0.10	5000	500	300
0.10	5499	300	300
0.10	2000	500	300
0.25	1000	500	300
0.25	2000	500	300
0.25	3000	500	300
0.25	5000	300	300
0.25	5499	500	300
0.25	4000	500	300
0.50	1000	500	300
0.50	2000	300	300
0.50	4000	500	300
0.50	3000	500	300
0.50	5000	500	300
0.75	1000	300	300
0.75	2000	300	300
0.75	3000	500	300
0.75	4000	500	300
0.75	5000	500	300
1.00	1000	300	300
1.00	2000	500	300
1.00	3000	500	300
1.25	1000	300	300
1.25	2000	500	300
1.25	4000	500	300
1.50	2000	300	300
1.50	1000	500	300
1.50	4000	500	300
1.50	5000	500	300
1.75	1000	500	300
1.75	5499	300	300
1.75	3000	500	300

1.75	4000	500	300
1.75	5000	500	300
2.00	1000	300	300
2.00	5499	300	300
2.00	2000	500	300
2.00	4000	500	300
2.00	5000	500	300
2.25	4000	300	300
2.25	5000	300	300
2.25	5499	300	300
2.25	1000	500	300
2.25	2000	500	300
2.50	5499	300	300
2.50	1000	500	300
2.75	5000	300	300
2.75	1000	500	300
2.75	2000	500	300
3.00	3000	500	300
3.00	4000	300	300
3.00	5000	300	300
3.00	1000	300	300
3.00	2000	500	300

Table A.8: Hyperparameter values for generating synthetic datasets with Smartnoise DPCTGAN on *Parkinson*

ϵ	**Dataset size**	**Batch size**	**Epochs**
0.10	1000	300	300
0.10	3000	300	300
0.10	4000	300	300
0.10	5000	300	300
0.10	5499	500	300
0.10	2000	500	300
0.25	3000	300	300
0.25	4000	500	300
0.25	5000	500	300
0.25	5499	300	300
0.25	1000	500	300
0.25	2000	500	300
0.50	5499	500	300
0.50	1000	500	300

0.50	2000	500	300
0.50	3000	500	300
0.50	4000	500	300
0.50	5000	500	300
0.75	5499	300	300
0.75	1000	500	300
0.75	2000	500	300
0.75	3000	500	300
0.75	4000	500	300
0.75	5000	500	300
1.00	2000	300	300
1.00	3000	500	300
1.00	4000	300	300
1.00	5499	500	300
1.00	1000	500	300
1.00	5000	500	300
1.25	1000	500	300
1.25	2000	500	300
1.25	3000	500	300
1.25	5499	300	300
1.25	4000	500	300
1.25	5000	500	300
1.50	5499	500	300
1.50	1000	500	300
1.50	2000	500	300
1.50	3000	500	300
1.50	4000	500	300
1.50	5000	500	300
1.75	1000	300	300
1.75	2000	500	300
1.75	3000	300	300
1.75	5000	300	300
1.75	5499	500	300
1.75	4000	500	300
2.00	2000	500	300
2.00	5000	500	300
2.00	5499	300	300
2.00	1000	500	300
2.00	3000	500	300
2.00	4000	500	300

2.25	2000	500	300
2.25	3000	300	300
2.25	4000	500	300
2.25	5499	500	300
2.25	1000	500	300
2.25	5000	500	300
2.50	1000	500	300
2.50	5000	500	300
2.50	5499	300	300
2.50	2000	500	300
2.50	3000	500	300
2.50	4000	500	300
2.75	1000	500	300
2.75	3000	300	300
2.75	5499	500	300
2.75	2000	500	300
2.75	4000	500	300
2.75	5000	500	300
3.00	1000	500	300
3.00	2000	300	300
3.00	4000	500	300
3.00	5000	500	300
3.00	5499	300	300
3.00	3000	500	300

Table A.9: Hyperparameter values for generating synthetic datasets with Smartnoise PATECTGAN on *Parkinson*