CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

WebDP – A Modular Differential Privacy API

Enhancing Accessibility and Flexibility of Differential Privacy Frameworks

Bachelor's Thesis in Computer Science and Engineering

DAVID AL AMIRI
BENJAMIN HEDE
ADAM NORBERG
SIMON PORSGAARD
SAMUEL RUNMARK THUNELL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024
www.chalmers.se | www.gu.se

Bachelor's Thesis 2024

WebDP – A Modular Differential Privacy API

Enhancing Accessibility and Flexibility of Differential Privacy Frameworks

DAVID AL AMIRI
BENJAMIN HEDE
ADAM NORBERG
SIMON PORSGAARD
SAMUEL RUNMARK THUNELL

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

WebDP – A Modular Differential Privacy API
Enhancing Accessibility and Flexibility of Differential Privacy Frameworks
DAVID AL AMIRI
BENJAMIN HEDE
ADAM NORBERG
SIMON PORSGAARD
SAMUEL RUNMARK THUNELL

Supervisor: Alejandro Russo, Department of Computer Science and Engineering
Graded by teacher: Niklas Broberg, Department of Computer Science and Engineering
Examiners: Arne Linde and Patrik Jansson, Department of Computer Science and Engineering

Bachelor's thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

WebDP – A Modular Differential Privacy API
Enhancing Accessibility and Flexibility of Differential Privacy Frameworks
DAVID AL AMIRI
BENJAMIN HEDE
ADAM NORBERG
SIMON PORSGAARD
SAMUEL RUNMARK THUNELL
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

Building on previous work by DPella, this paper describes the implementation of an open API server, WebDP, that provides *differential privacy* (DP) operations on datasets. With the initial goal of implementing a DP engine connector alongside an existing one written by DPella, the proposed WebDP server implements connectors to several DP libraries. By leveraging a microservice architecture, the proposed API and the modules are language agnostic, and administrative tasks can be implemented separately from computation calls to external DP libraries. The solution constitutes a platform that allows further DP framework additions with minimal changes required to the platform itself: The proposed architectural design works toward adding, removing, or updating both existing and new DP libraries to the platform, laying the groundwork for a modular system that adapts to – and respects – any arising changes within the emerging field that is differential privacy. By successfully implementing three DP libraries – OpenDP, Tumult Analytics, and GoogleDP – the platform reaches the goal of unifying several frameworks into one platform.

Keywords: differential privacy, WebDP, OpenDP, API, data, privacy, microservices, software development

WebDP – A Modular Differential Privacy API
Enhancing Accessibility and Flexibility of Differential Privacy Frameworks
DAVID AL AMIRI
BENJAMIN HEDE
ADAM NORBERG
SIMON PORSGAARD
SAMUEL RUNMARK THUNELL
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Sammandrag

Med tidigare arbete utfört av DPella som grund beskriver den här rapporten implementationen av en öppen API-server, WebDP, som tillhandahåller *differential privacy* (DP) -operationer på datamängder. Med initialt mål att implementera ytterligare en *DP engine connector* vid sidan av den befintliga skriven av DPella, tillhandahåller den föreslagna WebDP-servern *connectors* till fler DP-bibliotek. Genom att nyttja en mikrotjänstarkitektur kan både den föreslagna API-servern och dess moduler vara språkagnostiska, och administrativa uppgifter kan implementeras separat från beräkningsanrop till externa DP-bibliotek. Lösningen utgör en plattform som tillåter att ytterligare DP-ramverk läggs till med minimala ändringar i plattformen själv: Den föreslagna designen av WebDP eftersträvar att kunna lägga till, ta bort, samt uppdatera både befintliga och nya DP-bibliotek till plattformen, vilket lägger grunden för ett modulärt system som anpassar sig till — och respekterar — eventuella förändringar inom *differential privacy*, ett framväxande fält. Under projektet har tre DP-bibliotek framgångsrikt implementerats: OpenDP, Tumult Analytics och GoogleDP. Utifrån resultatet har plattformen nått målet att förena flera bibliotek under en och samma plattform.

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

API	Application Programming Interface
CRUD	Create, Read, Update and Delete
CSV	File type for Comma-Separated Values
DFA	Deterministic Finite Automata
DP	Differential Privacy
IoT	Internet of Things
IQR	Interquartal Range
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NFA	Non-Deterministic Finite Automata
OAS	OpenAPI Specification
PII	Personal Identifiable Information
REST	Representational State Transfer
SQL	Structured Query Language
URL	Uniform Resource Locator

Contents

1

Introduction

Although the notion that "privacy doesn't matter if you have nothing to hide" is wide-spread in society [1], very few would wish to share their doctor's appointment notes with their colleagues. To some extent, privacy is built into the foundation of all social interactions.

Thinking of the "privacy issue" as a question of herd immunity reveals some interesting aspects of responsibility. Even though some would argue adamantly that they truly have nothing to hide, we all have a responsibility to protect the privacy and integrity of those who do. Subjecting the population as a whole to intrusive data collection might expose certain demographics (victims of stalkers, ethnic and religious minorities, journalists, whistle blowers, political activists, etc) to grave risks, especially in cases where private data can be derived or inferred from publicly available data. To avoid these risks, it is crucial for those who hold sensitive information to take necessary precautions to keep the data private and secure.

This thesis explores the use of differential privacy, a mathematically secure model for data privacy, in a software application for use on datasets.

1.1 The false promises of data anonymisation

Anonymisation is a process in which Personally Identifiable Information (PII) is either removed from a dataset or obscured in some way. Examples of PII include names, social security numbers, places of birth, medical records, etc. [2]. The goal of obscuring or removing PII from data is to facilitate potentially valuable insights without compromising the private information of participants.

However, there are good reasons to believe that anonymisation of PII creates a false sense of security. For instance, a study conducted in the year 2000 revealed that 87% of the US population was likely to be uniquely identifiable by the combination of the attributes ZIP code, sex, and date of birth [3]. Interestingly, prior to this study, attributes like ZIP code, sex, and date of birth were not considered PII [2]. A famous case when anonymisation failed spectacularly was when then-governor of Massachusetts William Weld's hospital records were leaked. The GIC (Group Insurance Commission) had decided to release medical data of state employees, and

Weld had assured the public that patient privacy was not compromised due to the GIC having removed identifiers such as names and social security numbers [2]. A comparison between the released GIC data and another public non-anonymised dataset made it possible to uniquely identify Weld in the medical data [2].

Since combinations of attributes can be used to identify individuals, the question arises: Which combinations, in addition to the aforementioned attributes, should be obscured or removed from a dataset for the data to be sufficiently private? To see why this question cannot be answered, let us consider the nature of data analysis: For data to be analysed, and to be able to draw conclusions from the analyses, there has to be some substance in the data itself. By this principle, you cannot obscure *every* attribute of a given dataset as that would diminish its analytical value, ergo, some attributes will always be non-obscured. Any non-obscured attributes will be exposed for coupling with other, present or future, data sources – making *any* dataset that was anonymised in this way vulnerable to de-anonymisation. From this, we can derive that we cannot give any quantifiable privacy guarantees by simply removing or obscuring attributes. Of course, this does not mean that all private information could just as well be publicised in plain text, but rather it highlights the need for quantifiable security measures when it comes to PII, as anonymisation cannot guarantee the privacy of participants when releasing data.

1.2 Differential privacy: A brief introduction

Differential privacy (DP) is a mathematical mechanism that allows analysts to study populations without revealing information about specific individuals [4, 5]. More precisely, it guarantees that an analyst cannot deduce anything about an individual beyond what can be learned from the study of the entire population [5]. For instance, one could reasonably infer that a smoker generally has poorer health than a non-smoker, all other things equal, and this holds true regardless of whether the smoker in question has participated in a study that came to this conclusion [5].

The mechanism behind DP works by introducing finely calibrated noise to query results on a statistical database. The noise is introduced so that the same query on two database tables that differ on at most one row gives practically indistinguishable results. Therefore, participating in a study reveals just as much about oneself as not participating.

Person	Smoker
a8fdv8	YES
...	...
l44k2n	NO
eo3mf2	NO

(a)

Person	Smoker
a8fdv8	YES
...	...
l44k2n	NO
eo3mf2	NO
t555m1	YES

(b)

Table 1.1: (a) Table of an arbitrary number of rows of sensitive information. (b) Addition of a row to the original table.

For instance, consider Table 1.1 (a). As can be seen, the table's *Person* column has been anonymised to protect the participants' privacy. In its current state, the number of smokers is, let's say, 134, which is what a non-differentially private query would return. Now, consider Table 1.1 (b): Let us assume that we knew the number of smokers before the insertion of the new row, and that we now query the table again to obtain the count of 135. Further assume that we know, for instance, that our colleague has filled in the survey after we made the first query. Observing that both the row count and *Smoker* count have incremented by one since our last query, we can reasonably infer that our colleague is a smoker.

Note that we could make this inference through the simple interface of counting and filtering rows; the anonymisation did not protect privacy due to our auxiliary information. However, if we were to query the table in a differentially private manner, then perhaps we would first obtain the result "142.1", and in our second query "141.5". In this latter case, we could not deduce whether or not our colleague smokes since the results are practically unchanged between queries, but we could still infer that approximately 0.7% of the studied population are smokers, which potentially holds some value to us.

1.2.1 Applications of differential privacy

Researchers, industries and governments alike have a growing interest in differential privacy [6]. The U.S. Census Bureau is a far-reaching example of DP adaptation for confidentiality protection, as they used this technology while collecting and releasing population data bound to political and economic decision-making in their 2020 census [7].

In recent years, tech giants in the private sector in the US have started utilising differential privacy to collect and analyse segments of user data. Google has implemented an assortment of differentially private algorithms, RAPPOR, in its web browser Chrome, where it is used to collect and analyse crowd-sourcing statistics [8]. Apple has implemented a corresponding set of DP algorithms for its systems [9]. Both systems are opt-in.

Differential privacy can be applied to machine learning methodologies, if privacy preservation within crowd-sourced datasets is desired. This can still be achieved "within a modest ('single-digit') privacy budget" [10]. However, DP applications in machine learning have been shown to not be faithful to the original DP formulation, and to instead use slight relaxations to the definitions [11].

1.2.2 WebDP: An interface for differential privacy

Since the inception of DP, a handful of libraries of DP algorithms have been implemented in different general purpose computer languages. However, since differential privacy is a complex – albeit in many cases useful – subject, many potential users might be discouraged from utilising these libraries.

WebDP is an open-source initiative to lower the entry threshold to differential privacy by providing an abstracted, unified interface to the lower-level workings of differential privacy libraries [12]. More importantly, the initiative aims to make it easy for organisations to set up differentially private environments for their analysts without needing to share their sensitive data with a third party.

The WebDP server was developed by DPella [13], a company founded by computer science researchers. To date, the development team has implemented a proof-of-concept WebDP API-compliant server that is powered by Tumult Labs' [14] differential privacy framework Tumult Analytics [15]. This proof-of-concept implementation is described and analysed in Section 2.3. As two bachelor's theses, DPella suggested extending the current WebDP server with another DP framework – OpenDP, developed at Harvard University [16] – and building a front-end for the server [15].

1.3 Web applications: Protocols and terminology

To assist the novice reader or refresh the memory of the more experienced reader, this section provides a brief overview of some of the core concepts in web applications and communication.

Most web applications rely entirely on system-to-system communication: The client sends a request to the server, and the server responds to the request. An API (Application Programming Interface) is an agreed-upon interface that determines how the communication should be formed, so that both parties can interpret the other's messages. A popular paradigm for designing APIs is REST (Representational State Transfer) which focuses primarily on *resources*. A resource is an entity (e.g. a HTML web page, image file or data object) which can be altered with the basic operations Create, Read, Update, and Delete (CRUD). These operations directly correspond to the HTTP methods Post, Get, Put/Patch and Delete [17]. Resources can be accessed by making requests to the exposed API *endpoints*, usually in the forms of HTTP URLs. The endpoints specify what kind of information or resource we want to get from, or send via, the API. For instance, we might send a HTTP Get request to `http://cooking.com/recipes/pies` to retrieve a

list of pie recipes, or send a HTTP Post request containing a *payload* to the same address to post a new pie recipe. The payload contains the new object, which is usually formatted as a JSON object. JSON is a lightweight text format that is completely language independent and can therefore be implemented into any API of any programming language [18]. The data itself is constructed into unordered sets of key-value pairs, which makes it easy for developers to read, and easy for machines to parse [18].

For getting specific resources by their IDs or attributes, APIs support making requests using parameters: In `.../pies/normas_cherry_pie`, the ID of the resource, "normas_cherry_pie", is specified as a *path parameter*. The API might also support functions like sorting and filtering. For instance, a request to `.../pies?type=cherry ?author=norma` might return a list of all the cherry pie recipes that Norma has uploaded. Here, "type" and "author" are *query parameters*.

A standard practice for web application security is to ensure authentication and authorisation of users. In most modern applications, this is implemented through OAuth 2.0 – the industry-standard protocol for authorisation [17]. The protocol utilises two types of tokens for verifying users of an API: Access tokens and refresh tokens [17]. The access token has a limited lifespan and is used across all queries to ensure the user's identity and permissions. It is often generated by the API provider via authentication [19]. The second token, the refresh token, is used to gain a new access token when it has expired. Refresh tokens usually have a much longer lifespan, but can be revoked if necessary [17].

A common way to structure and specify the API contents is using the OpenAPI Specification (OAS) framework. The OAS can be used in all stages of the life-cycle of an API, in order to standardise the language used in agnostic terms, and to decouple them from any programming language. The specification offers a comprehensive "dictionary of terms" that summarises the functionality of an API: It lists the available endpoints and their supported operations, operation parameters, authentication methods, licenses and contact information [20].

1.4 Project description

This project aims to improve WebDP, a software application serving APIs for differential privacy operations on datasets, by extending its functionality to other DP engines and frameworks. An *engine* or a *core* will in the context of this thesis refer to the part of a code library or (micro)service that provides functionality for making and executing differentially private queries. The *connector* is the bridge between the internal WebDP API framework and an external DP library. Depending on the library, the connector may connect to the framework or the engine.

The finished project will be released to the public under the open source license Mozilla Public License (MPL) 2.0 at `https://github.com/dpella/webdp-v2`.

1.4.1 Purpose

The purpose of this project is twofold: There is a strictly practical purpose to extend the functionality of WebDP by making the OpenDP library available through its interface; there is also a more general purpose, which is to find areas of improvement to the WebDP API to lower the barrier of adaptation, and, in doing so, make differential privacy more usable and available to a wider public.

1.4.2 Problem formulation

The primary goal is to build a connector to connect WebDP API to the OpenDP endpoints. The WebDP server itself should be adapted for extensibility and designed in such a way that the OpenDP connector can be switched for connectors to other DP libraries and frameworks. This entails constructing a common, standardised interface for existing (and future) DP framework APIs using appropriate programming designs and principles.

The finished product should follow the specification as provided by WebDP, and carefully consider trade-offs in performance, scalability, adaptability and suitability with respect to its purpose. Since erroneous code might introduce vulnerabilities that break the DP guarantees on privacy, the code base must be verified thoroughly to ensure correctness and rigidity.

During the development process, the project should evaluate and suggest modifications necessary to the WebDP API based on how well it fits the engines, such that it achieves the goals listed above.

1.5 Delimitations

This thesis is not concerned with the technical details of differential privacy itself. Rather, it builds upon the assumption that functional implementations of differential privacy exist, are available, and suitable for the project's purposes, as described in Sections 1.4.1 and 1.4.2. Therefore, only the fundamental notions and concepts will be studied in the scope of this thesis. A mathematical explanation of some important concepts will be given to the reader in Section 2.1.

1.6 Target audience

This work will be of concern to those who are interested in differential privacy (or privacy in general), open-source software and DevOps. Although differential privacy will function as a mere backdrop to our work, this project may inform the reader to what kinds of tools are available and what their limitations are.

2

Background

To gain some insight into the inner workings of the privacy guarantees that differential privacy provides, Section 2.1 is devoted to digest the mathematical foundations of differential privacy as well as providing key terminology. Readers who are not interested in mathematics may skip ahead to Section 2.2 for an introduction to some of the available DP libraries, or to Section 2.3, where the current implementation of the WebDP server is presented.

2.1 Differential privacy

Definition 1 (Differential privacy [21] [22])**.** *A randomised function $\tilde{Q}$ gives (ε, δ)-differential privacy if for all datasets D_1 and D_2 differing in at most one element (row), and all $S \subseteq Range(\tilde{Q})$,*

$$Pr[\tilde{Q}(D_1) \in S] \leq e^{\varepsilon} \cdot Pr[\tilde{Q}(D_2) \in S] + \delta \tag{2.1}$$

For our purposes, the randomised function, $\tilde{Q}$, is considered to be an exact function (such as a summation over a column) with added random noise. The noise is selected to satisfy the inequality in Equation 2.1, which states that the probability of yielding an output in a certain range S when querying D_1 is not greater, by a factor determined by the privacy parameter ε and a small constant δ, than the probability of yielding a result in the same range when querying D_2. Note that D_1 and D_2 are interchangeable with each other since "x and y differs in one row" is a commutative statement. Intuitively, this means that we expect to yield similar results from a differentially private randomised function on both datasets, which hides the effect that the differing row has on the result for appropriate values of ε and δ.

The special case when $\delta = 0$ is commonly known as *Pure Differential Privacy* (Pure DP) and is said to preserve "ε-differential privacy" [23]. When $\delta > 0$, it is referred to as *Approximate Differential Privacy* (Approximate DP) and is said to preserve "(ε, δ)-differential privacy" [23]. Both Pure DP and Approximate DP are instances of *Privacy Notions* in the WebDP API. While other notions of privacy exist, they are not supported by WebDP.

Definition 2 (Neighbouring datasets: Unbounded DP [24]). *Two datasets D_1 and D_2 are considered neighbouring if D_1 can be obtained from D_2 by adding or removing a single row.*

The notion that two datasets "differ in at most one row" has two natural interpretations: Either it is interpreted as in Definition 2 (Unbounded DP), or it is interpreted as the datasets having equal size and one can be obtained from the other by removing a row from and adding a row to one of the datasets (Bounded DP) [24]. Depending on the definition of closeness, some important theorems may not apply [24]. The definition used in this thesis will be unbounded DP, if not explicitly stated otherwise.

Definition 3 (Privacy Loss [22]). *If $\tilde{Q}$ is a (ε, δ)-differentially private randomised function, D_1 and D_2 are neighbouring datasets, and $x \sim \tilde{Q}(D_1)$, then the Privacy Loss incurred by observing x is defined as:*

$$L^{x}_{\tilde{Q}(D_1)||\tilde{Q}(D_2)} = \ln \frac{Pr[\tilde{Q}(D_1) = x]}{Pr[\tilde{Q}(D_2) = x]} \tag{2.2}$$

In Pure DP, the absolute value of the privacy loss has an upper bound of ε. In Approximate DP, the absolute value of the privacy loss has an upper bound of ε with a probability of $1-\delta$ [22]. Or phrased slightly differently: There is a probability of δ that the absolute value of the privacy loss is greater than ε. This is the same thing as breaking the preservation of ε-differential privacy with a probability of δ.

Theorem 1 (Parallel composition of DP queries [24]). *Let $\tilde{Q}_1, ..., \tilde{Q}_n$ be n queries that satisfy $\varepsilon_1, ..., \varepsilon_n$-differential privacy, respectively. Given a partitioning $D_1, ..., D_n$ of a dataset D, publishing $\tilde{Q}_1(D_1), ..., \tilde{Q}_n(D_n)$ preserves ε-differential privacy for $\varepsilon = \max(\varepsilon_1, ..., \varepsilon_n)$*

Theorem 1 applies only under unbounded DP [24]. A natural application for Theorem 1 is histograms, where D_i is a category and $\tilde{Q}_i$ is a differentially private count. The parallel composition also extends to (ε, δ)-private queries.

Proof. Let D and D' be neighbouring datasets. Let $D_1, ..., D_n$, and $D'_1, ..., D'_n$ be partitions of the datasets, and let $\tilde{Q}_1, ..., \tilde{Q}_n$ be queries that satisfy $(\varepsilon_1, \delta_1), ..., (\varepsilon_n, \delta_n)$-differential privacy, respectively. Since the datasets differ in at most one row, there exist at most one index $i \in \{1, ..., n\}$ such that $D_i \neq D'_i$. Thus, for all indices $j \in \{1, ..., n\} \setminus \{i\}$, $\tilde{Q}_j$ preserves ϵ-differential privacy for $\epsilon = 0$ since the partitions are identical:

$$Pr\left[\tilde{Q}_j(D_j) \in S\right] = Pr\left[\tilde{Q}_j(D'_j) \in S\right]$$

Thus, we have that:

$$Pr\left[\tilde{Q}(D) \in S\right] \leq (e^{\varepsilon_i} Pr\left[\tilde{Q}(D'_i) \in S\right] + \delta_i) \prod_{j \neq i} Pr\left[\tilde{Q}(D'_j) \in S\right] \tag{2.3}$$

$$= e^{\varepsilon_i} \prod Pr\left[\tilde{Q}(D'_j) \in S\right] + \delta_i \prod_{j \neq i} Pr\left[\tilde{Q}(D'_j) \in S\right] \tag{2.4}$$

$$\leq e^{\varepsilon_i} \prod Pr\left[\tilde{Q}(D'_j) \in S\right] + \delta_i \tag{2.5}$$

$$= e^{\varepsilon_i} Pr\left[\tilde{Q}(D') \in S\right] + \delta_i \tag{2.6}$$

$$\leq e^{\max(\varepsilon_1,...,\varepsilon_n)} Pr\left[\tilde{Q}(D') \in S\right] + \max(\delta_1, ..., \delta_n) \tag{2.7}$$

The inequality in (2.5) is given by the fact that a product of probabilities is at most 1. □

2.1.1 Budgets

For any definition of privacy protection, it is true that a response to a query on a database that gives an answer that is bounded by some error term E is also bound to leak *some* privacy [25]. More specifically, if an unbounded number of queries are answered with an accuracy of E, then an adversary may eventually reconstruct the database with a high precision [25]. This has also been called the "Fundamental Law of Information Recovery" [22].

Theorem 2 (Sequential composition of DP queries [22])**.** *Let $\tilde{Q}_i$ be a $(\varepsilon_i, \delta_i)$-differentially private randomised function for $i \in [1, 2, ..., k] = I$, and let $\tilde{Q}_I$ be defined as $\tilde{Q}_I(x) = (\tilde{Q}_1(x), \tilde{Q}_2(x), ..., \tilde{Q}_k(x))$, then $\tilde{Q}_I$ is $(\sum_{i \in I} \varepsilon_i, \sum_{i \in I} \delta_i)$-differentially private.*

Theorem 2 shows that a sequence of differentially private queries on a dataset will preserve differential privacy under the sum-total of their respective ε and δ upper bounds. For example, if $\tilde{Q}_1$ preserves ε-differential privacy for $\varepsilon = 0.5$ and $\tilde{Q}_2$ preserves ε-differential privacy for $\varepsilon = 0.2$, then the query sequence $\tilde{Q}_1, \tilde{Q}_2$ will preserve ε-differential privacy for $\varepsilon = 0.5 + 0.2 = 0.7$. As seen in Section 2.1, by Definition 3, this value is the upper bound of the privacy loss incurred on the dataset by answering the queries $\tilde{Q}_1$ and $\tilde{Q}_2$. Theorem 2 and Definition 3 thus make it convenient to reason about possible measures against privacy leaks: A curator of a dataset can set its maximum level of tolerable privacy loss in terms of ε and δ, and simply stop answering queries when the accumulated ε and δ values of the answered queries have reached this threshold. This threshold is the *budget* of the dataset. When we say that the budget of a dataset is (ε, δ), we mean that we tolerate a maximum privacy loss of ε with the probability of δ of leaking more privacy than ε.

2.1.2 Accuracy estimations

The OpenDP library and DPella's own framework both support pre-evaluation accuracy estimations of a differentially private query [26, 27].

Definition 4 (Pre-evaluation accuracy estimation [26]). *Let $\mathbb{D}$ be the universe of datasets, $Q : \mathbb{D} \to \mathbb{R}$ and $\tilde{Q} : \mathbb{D} \to \mathbb{R}$, such that for a dataset $D \in \mathbb{D}$, $\tilde{Q}(D) = Q(D) + \lambda$ for some level of noise λ such that $\tilde{Q}$ preserves (ε, δ)-privacy for some ε, δ. Then the accuracy, denoted "a", of $\tilde{Q}$ is defined such that*

$$\forall D \in \mathbb{D}. Pr\left[|Q(D) - \tilde{Q}(D)| \leq a\right] \geq \beta \tag{2.8}$$

where β is the confidence at which the error is bounded by a.

The pre-evaluation accuracy a is a probabilistic measurement of the absolute error of a differentially private query. The statement (2.8) in Definition 4 reads as "for all datasets D in the universe of datasets $\mathbb{D}$, the result of $\tilde{Q}$ will not deviate more than a from the true value at a probability of at least β". "Pre-evaluation" means that an analyst does not have to spend budget on an accuracy estimation, as it does not leak privacy. This is due to the fact that the level of noise needed to preserve differential privacy is not dependent on the data, but rather on the *sensitivity* of $\tilde{Q}$. Section 2.1.4 will explain the sensitivity more thoroughly.

An analyst may request accuracy estimations of queries that preserve (ε, δ)-differential privacy for different (ε, δ)-tuples to gain knowledge of the marginal change in absolute error with respect to the cost of the query. For example, assume that the pre-evaluation accuracy is $a = 100$ for some differentially private row count on a dataset and at some high confidence ($\beta > 0.95$). Further assume that the analyst knows what kind of data the dataset contains. With this knowledge, an absolute error of 100 may or may not be tolerable. If, for example, the dataset contains the names of staff at some kindergarten, then an absolute error of 100 may be intolerable (assuming that there are probably very few kindergartens with more than 100 employees). The analyst would then know that they have to increase the budget expense to get a useful result, assuming a confidence level of β is necessary. On the other hand, if the dataset contains the names of all citizens of Paris, then an absolute error of 100 may be unnecessarily precise, and the analyst may spend less of their budget and still get a useful result.

2.1.3 Post hoc accuracy estimations

Theorem 3 (Bounded absolute error of a post-processed ratio). *Let $\tilde{x}, \tilde{y} \in \mathbb{R}$ be the results of two differentially private queries, where $x, y \in \mathbb{R}$ denote their true values such that the pre-evaluation accuracy estimates*

$$Pr\left[|\tilde{x} - x| \leq a_x\right] \geq \beta_x$$

$$Pr\left[|\tilde{y} - y| \leq a_y\right] \geq \beta_y$$

and it is given that $0 < \tilde{x} - a_x$. Further let $m = \frac{y}{x}$ and $\tilde{m} = \frac{\tilde{y}}{\tilde{x}}$, then we have that

$$Pr\left[|m - \tilde{m}| \leq \frac{|\tilde{x}a_y + \tilde{y}a_x|}{\tilde{x}(\tilde{x} - a_x)}\right] \geq \beta_x \cdot \beta_y \tag{2.9}$$

The proof of Theorem 3 is given in Appendix C. Theorem 3 states that if two differentially private measurements, $\tilde{y}$ and $\tilde{x}$, have been made, and that their pre-evaluation accuracies, a_x and a_y, are known at some levels of confidence, β_x and β_y, then the ratio of the measurements have a probabilistic upper bound at a confidence level of $\beta_x \cdot \beta_y$ – given that the denominator of the measurement one is trying to make, x, is at least β_x likely to be positive. This theorem may be useful for estimating the error of a post-processed mean measurement ($\tilde{x}$ is a DP count and $\tilde{y}$ is a DP sum) or a post-processed measurement of proportions of populations (both measurements are DP counts).

2.1.4 Noise mechanisms

The examples shown so far may have erroneously suggested that an analyst determines the cost of the query after having constructed it. In reality, there are convenient methods for constructing queries that preserve (ε, δ)-differential privacy, with the analyst selecting appropriate values for ε and δ.

Definition 5 (Global sensitivity [24]). *Let $D_1, D_2 \in \mathbb{D}$ be neighbouring datasets. Then for a function $f : \mathbb{D} \to \mathbb{R}$, the global sensitivity of f, denoted Δ_f, is defined as*

$$\Delta_f = \max |f(D_1) - f(D_2)| \tag{2.10}$$

There are similar definitions to Definition 5 where f outputs a vector [24]. Equation 2.10 describes the sensitivity of a function as the impact that the differing rows of the datasets have on the output. For instance, if f is a function that sums values between -1 and 5, then $|f(D_1) - f(D_2)| \leq 5$ for datasets D_1 and D_2 that differ by one row, according to the unbounded DP definition. This is because the maximum value of the row by which they differ is 5. For bounded DP, the sensitivity is 6 since they differ in the rows $r_1 \in D_1$ and $r_2 \in D_2$, and $|r_1 - r_2| \leq 6$.

Theorem 4 (Laplacian noise mechanism [24]). *For any function f, the query $\tilde{Q}(D) = f(D) + Lap(\frac{\Delta_f}{\varepsilon})$ satisfies ε-differential privacy.*

Theorem 4 states that adding random noise from the Laplace distribution centred at 0 and scaled by $\frac{\Delta_f}{\varepsilon}$ to the result of $f(D)$ preserves ε-differential privacy. For example, if f is the function used in the previous example, $f : [-1, 5] \to \mathbb{R}$, then $f(D) + \text{Lap}(5)$ preserves ε-differential privacy for $\varepsilon = 1$. In [22], Dwork and Roth outline a mechanism that samples noise from a normal distribution that preserves (ε, δ)-differential privacy for $\varepsilon \in (0, 1)$.

The WebDP API supports perturbing results of queries with noise sampled from the Laplace distribution as well as the normal distribution, commonly called the Gaussian mechanism. As is evident from Theorem 4, the Laplacian noise is used to achieve ε-differential privacy. The Gaussian mechanism is used for achieving (ε, δ)-differential privacy [22].

2.1.5 The sensitivity of common functions

Filtering rows, selecting columns, and renaming columns all maintain the precondition of differential privacy. In other words, if neighbouring datasets D_1 and D_2 undergo the same transformation (as mentioned above), they will remain neighbouring datasets. The cases for selecting and renaming columns are self-evident, as these operations do not alter the underlying data. When filtering rows, since the differing row of D_1 and D_2 either satisfies the filtering predicate or not, the resulting filtered datasets cannot differ by *more* rows than they did before. This means that these transformations do not affect the amount of noise needed to preserve differential privacy.

By the definition of neighbouring datasets, the sensitivity of a counting query is 1. The sensitivity of a function that sums numbers on a closed interval $[a, b]$ is $\max(|a|, |b|)$ for unbounded DP and $\max(|a|, |b|, |a - b|)$ for bounded DP.

2.2 Differential privacy libraries and frameworks

There is a variety of library configurations among DP libraries; some only provide functionality for raw DP queries while others implement a framework around the query services. This section will highlight a few of the available libraries.

2.2.1 OpenDP

The OpenDP library is an open source library that provides differential privacy functions [16]. It supports transformations and measurements to be applied onto a given dataset. The library is not intended to be used directly by end-users, but rather for implementation in another user-friendly system [6] – it is a layered structure that can be extended to other programming languages such as Python, R or Rust [28].

2.2.2 Tumult Analytics

The Tumult DP library, developed by Tumult Labs, is an open source library that provides functionality for running differentially private queries. The library contains two separate parts: The Tumult core and Tumult Analytics framework. The core contains the privacy-critical logic and is designed to be modular and extensible, while the framework supports many standard operations such as filters, joins and maps, and aggregations such as counts, averages and quantiles, among others. The framework is designed to be easy to use for non-experts while still providing a lot of functionality [29].

2.2.3 Google's DP libraries

Google has built some libraries to help developers in implementing differential privacy in their own applications. The libraries are available in several programming languages like C++, Go and Java with Python wrappers also available through

third-party sources. The developers have also created several tools to make these libraries available to a wider user base [30]. The core libraries currently only support aggregations. Google has openly stated that there is a vulnerability in the floating-point implementation [30], which could be exploited according to the method described in a report by OpenDP contributors [31].

2.2.4 Other libraries

The aforementioned libraries all focus on providing general-purpose APIs for handling differential privacy, providing a starting point for developers implementing DP into their applications. Due to the amount of crowd-sourced data that has to be privatised (see e.g. [10]), many of the available DP libraries are specifically designed to be used in the learning process of machine and deep learning models. Examples include PyTorch's Opacus [32], Secretflow [33] and IBM's general-purpose library Diffprivlib [34].

2.3 Current state of the WebDP server

The WebDP server code base is written entirely in Python. Most of its modules were generated using OpenAPI Generator [35], which creates an MVC pattern from the specification used as input, where the server-side models and controllers are built from the specification components [36].

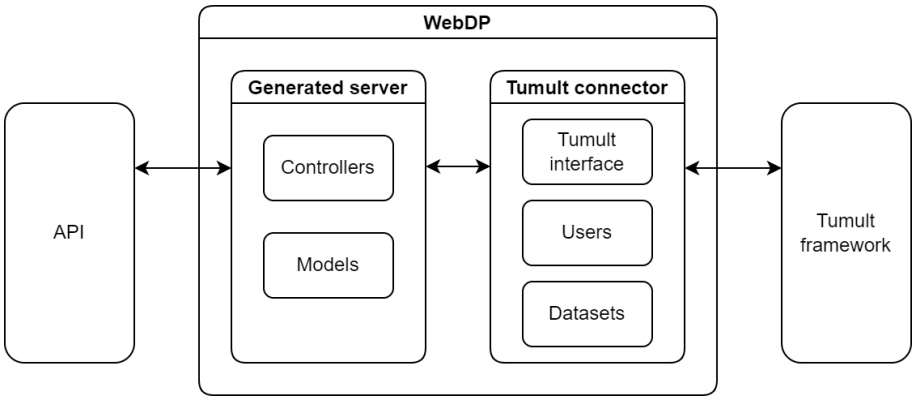

Figure 2.1: Original monolith design of WebDP. User and dataset handling is build into the Tumult connector.

In addition to the generated server skeleton, the WebDP source directory contains an engine connector to the Tumult DP framework (see Figure 2.1). The generated controllers, through which all requests are routed, all connect to the Tumult connector module – a singleton class encompassing the server logic, internal states and the Tumult engine interface. Due to the design of the current implementation, making additions to the code base such as adding a second or third DP engine would take considerable effort and redesign of the application. Additionally, there is no persistence built into the application, which means if the application is restarted, all users, budgets and datasets would be reset.

2.3.1 WebDP API specification

The WebDP API is an attempt toward creating a RESTful interface for interactive differentially private inquiries on sensitive data. *Interactive* denotes that the data does not have to be under the control of the one making the query, and that the exact formulation of the query is not under the control of the one who owns the data. The latter is realised by differential privacy itself: As long as a query preserves (ε, δ)-differential privacy, the query is answered if and only if its associated ε and δ values do not make the accumulated privacy loss of the dataset exceed its budget. The way in which WebDP separates the need to control (own) the data from the ability to query it, is by enabling allocation of a part of a dataset's non-consumed budget to a user. This way, a user may query a dataset as long as there is enough budget allocated for them, and there may be multiple users having budget allocations on the same dataset.

The WebDP API [12] specifies five categories of endpoints: *Users*, *Datasets*, *Budgets*, *Queries* and *Authentication*. The endpoints are listed in Table 2.1.

Category	Endpoints

Budgets	`/budget/allocation/{user}/{dataset}`
	`/budget/dataset/{dataset}`
	`/budget/user/{user}`

Datasets	`/datasets`
	`/dataset/{dataset}`
	`/dataset/{dataset}/upload`

Authentication	`/login` `/logout`
Users	`/users` `/user/{user}`

Queries	`/query/accuracy`
	`/query/custom`
	`/query/evaluate`

Table 2.1: The WebDP API endpoints.

Users: The users of the system have usernames (handles), given names, passwords and one or more roles (*Admin*, *Curator* and *Analyst*). A user's roles determine what actions the user is authorised to take and what functionality is accessible to them. For instance, a data curator may create and update datasets, handle tasks such as managing the dataset schema and overseeing the dataset's budget, etc, while a data analyst may not. An administrator's responsibilities primarily revolve around user management, including adding new users and updating existing ones. The *Users* endpoints are basic CRUD operations which serve as the main way of interaction with the data.

Datasets: In the API specification, the creation of a dataset is separated from the uploading of the actual data. Creating a dataset involves giving it a name, a privacy notion (see Section 2.1), total budget, owner and a column schema. The column

schema is a mapping $Names \rightarrow Types$, that indicates which column is of which type. The available types are integers, doubles, text, booleans and enums. The data may be uploaded as a CSV file to the server after the creation of the dataset. The uploaded data is hidden and immutable to all users – there are no GET, PATCH or DELETE endpoints for the data connected to the dataset. However, the data may be deleted by deleting the dataset. The total budget, name and owner fields in a dataset are mutable.

Budgets: Budgets are intricately linked to users and datasets, facilitating their management and utilisation. The WebDP API specifies basic CRUD operations for handling a user's allocated budget on a single dataset. The allocated budget is the gross budget a user has had allocated, as opposed to the net (consumable) budget that is left after the consumed budget has been subtracted. There are also two more GET endpoints for retrieving all the allocations of a user, which include the amount of budget spent, as well as all budget allocations of a dataset. There is no GET endpoint for retrieving a user's consumed budget on a single dataset – this information has to be retrieved by filtering the result from the endpoint returning all of the user's allocations.

Queries: There is a collection of JSON objects, called *Query Steps*, in the WebDP API specification that are the building blocks of the differentially private queries. The endpoint `/evaluate` evaluates a sequence of query steps on a dataset (see Table 2.2). Apart from the sequence of query steps, and the dataset on which to evaluate the query, the user also specifies how much of their allocated budget to spend on evaluating the query. The API specification does not mention any other restriction on the syntax of the query, other than that it should be built out of the specified query steps.

Query Step	**Description**
`SelectTransformation`	Projection of the dataset columns
`FilterTransformation`	Filter of the dataset rows
`RenameTransformation`	Renaming of the dataset columns
`MapTransformation`	Function mapping over the dataset columns
`BinTransformation`	Mapping of values to predefined bins
`CountMeasurement`	Differentially private count
`SumMeasurement`	Differentially private summation
`MeanMeasurement`	Differentially private mean measurement
`MinMeasurement`	Differentially private min function
`MaxMeasurement`	Differentially private max function
`GroupbyPartition`	Grouping partition over some columns

Table 2.2: A table of the available Query Steps in the WebDP API.

The `/accuracy` endpoint is meant to give a pre-evaluation accuracy estimation of a query at some confidence level β. The `/custom` endpoint is similar to `/evaluate`, apart from expecting a differentially private query as a string and not as a sequence of query steps. As the name suggests, the query is custom and it is therefore up to the one implementing the API to define its syntax.

2.3.2 The Tumult class

Seeing how the Tumult Analytics API was missing some key features, like user management, authentication and convenient handling of multiple datasets, DPella added the missing features to their Tumult connector [35]. While mostly for demonstration purposes in the proof-of-concept connector, the class grew beyond its original mould, making it a "God class" handling all server logic.

Since the coupling of user, dataset and budget management to the Tumult framework interface is very high, adding any new connectors to the application would be cumbersome: Either you need to incorporate the new framework into the monolithic class, or you need to duplicate the additional features into a new connector to create a mirror image of the monolithic class. With both approaches, the code would be hard to maintain and the interfaces would be hard to unify, with each additional framework extension making it even harder. Furthermore, when the data is loaded into the application, the underlying engine, Tumult Analytics, wraps the data in a Tumult-specific wrapper which makes the data inaccessible for use by other engines.

Since clients might not want to upload their sensitive data to a third-party server, a portable solution that is easy to run locally is preferred. This means that another issue with this type of architecture is that implementing the system at clients' premises might entail changes to user or data handling – i.e., adapting it to the clients' own user and data handling – which simply would not be feasible at this level of coupling within the module.

3

Implementation

From the start, DPella's ambition has been to extend the functionality of WebDP and add additional engines to the API; the first step being integrating the OpenDP framework alongside the current implementation of the Tumult connector. However, very little of the functionality in the WebDP API – adapted for use with the Tumult framework – is supported in OpenDP.

Because of the differences in API support between the OpenDP and Tumult libraries, and the limitations of the proof-of-concept implementation (see Section 2.3), constructing another connector to fit WebDP's monolithic architecture would entail duplicating the code used for the Tumult connector and adapting it for use with the OpenDP framework. Implementing additional connectors in a way that would satisfy the desired level of extensibility and scalability would likely require an assessment of the application base: To integrate the OpenDP framework and prepare the code base for additional connectors, WebDP would need to re-designed "with generalisation in mind" [15]. For this purpose, building a modular architecture, which separates user handling and datasets from the connector itself, has many benefits. The modular design is described in Section 3.1.1. The new framework is written in Go, a compiled, light and typed language.

The biggest design challenges in integrating additional frameworks alongside that of Tumult is managing the choice of which engine to run a given query on (further discussed in Section 3.3.1) and adapting the WebDP API for use with multiple frameworks (see Section 3.5).

3.1 Architectural considerations

As highlighted in Section 2.3, the implementation of WebDP is missing some flexibility with regards to incorporating additional DP libraries. This limitation arose from the tight coupling between user, dataset, and budget administration with the querying components of the application. In the subsequent subsections, the considerations undertaken in the design and implementation of the application to achieve greater modularity and flexibility are presented. These efforts aim to decouple the administrative aspects of the application from the querying logic.

3.1.1 Modularity and separation of concern

Although there was a possibility to extend DPella's proof-of-concept WebDP server and Tumult connector with more connectors, the proposed WebDP server is built from scratch with a more modular design. This decision was made for several reasons:

- The WebDP server and Tumult connector were tightly coupled: The estimated time it would take to refactor the current code base would take as much time as it would building a new server from scratch.
- Keeping the WebDP server and connectors in the same package would limit DP library choices to those providing APIs in the same programming language as the server.
- The WebDP server was not initially built with persistence as a priority. Therefore, revisiting its design would enable factoring in persistence-oriented considerations.

To solve the issues with the current WebDP architecture, the proposed solution is more modular architecture as seen in Figure 3.1 where the user handling, data handling and the DP engines are developed separately. This gives the code base low coupling which enables and makes it easier to, for example, add connections to the companies' own user handling systems, or add new DP engines in the future.

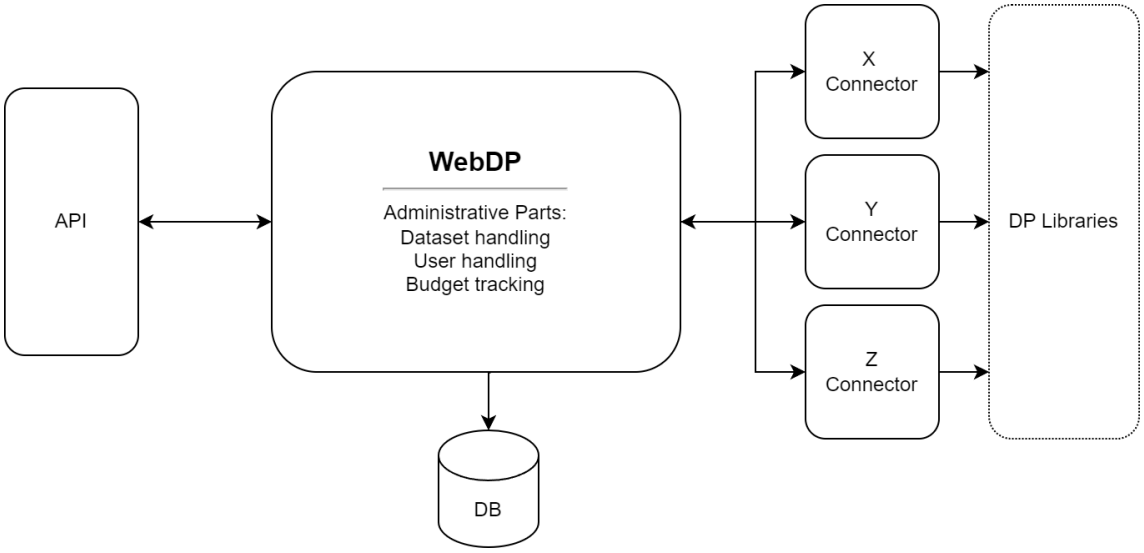

Figure 3.1: Proposed architecture for the WebDP application. Dataset and user handling are separated from the engine connector modules.

Simplifying the addition of new engines necessitates separating the administrative tasks from the querying logic. The proposed application comprises two major modules that operate in conjunction: The administrative module, which manages users, roles, budgets and datasets, and the querying module, which manages the communication with the engine services. Queries on datasets are performed by users with an allocated budget on the queried dataset, utilising specific engines for execution. This enables users to extract insights and analyse data within defined parameters, the details of which are defined in Section 2.1.

3.1.2 Engine connectors

In the proposed microservice architecture – where a central unit handles administration, authentication and authorisation of users, datasets and budgets – the task of integrating a connector to a DP library is the task of translating the sequence of query steps received from the central unit (WebDP) to coherent code that utilises the library in question. Figure 3.2 demonstrates the internals of an engine connector, where the query builder translates the query steps to achieve the goal of the query.

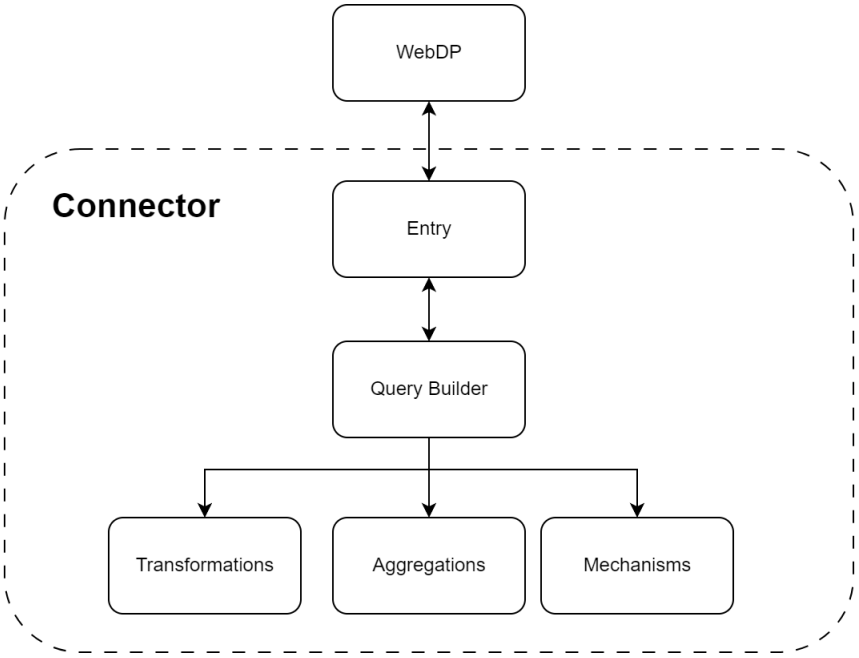

Figure 3.2: Architecture of the engine connectors. The query builder assembles the query.

The Sections 3.2 and 3.3 will highlight the implementation details of the OpenDP, Tumult, and GoogleDP connectors, placing particular emphasis on the OpenDP connector and its alignment with the function set of the OpenDP library within the WebDP specification. The integration of Tumult and GoogleDP connectors will further serve as tangible demonstrations validating the architectural decisions behind the proposed WebDP API.

3.1.3 Containers

One of the bigger obstacles to the adoption of a service such as WebDP is that clients might not want their data handled by a third party. Instead, what would happen is that services such as WebDP are sold as a software bundle, which is integrated into the clients' own hosted servers. To make this product marketable it needs to have a low cost of integration and the bundle should seamlessly integrate with the clients' own systems.

With these requirements as a backdrop, containerisation of the modules (using a container service like Docker [37]) has some significant advantages: First and foremost, only five internal endpoints (see Section 3.5.5) are needed to communicate with the WebDP network, making the interface easy to implement and in turn making the threshold to add new engines lower. Second, a load balancer can spin up new instances of any engine that is more popular than the others. Third, it makes the application mobile in the sense that the target system would not require any additional installation of dependencies. The Docker image containing the API router is only about 30 MB.

3.1.4 Authorisation scheme

When building the authorisation scheme, there was the option of either attaching the user role to the bearer token used at login, or fetching the role from the database each time the role is needed for authorisation – the latter allowing for dynamic role updates. Since the majority of operations need user role authorisation, setting roles dynamically has a big downside of querying the database layer each time a user performs an operation in the application. Assuming that role changes will be a rare occurrence and that logging in and out will not bother the user too much when it does happen, an authorisation scheme using token claims was found to be the more reasonable choice. This is reinforced by the fact that the user might even expect to be logged out when changing other authentication or authorisation credentials (like username and password).

Optimally, the application would make use of a more extensive token service with session and refresh tokens (collaborating with the front-end). For this use case however, the simpler solution using bearer tokens only is good enough in relation to the project scope, and further improving it in the future is possible.

3.2 The OpenDP connector

The OpenDP library has out-of-the-box support for a small subset of the WebDP query steps: *SumMeasurement*, *CountMeasurement* and *BinTransformation*. Some query steps could be supported by making small extensions to the OpenDP library: *RenameTransformation*, *FilterTransformation* and *SelectTransformation*. There is support for making mean measurements in OpenDP, but its interface is not compatible with the current state of the WebDP API. No other query step had support

in OpenDP. The OpenDP library supports making pre-evaluation accuracy calculations, which is its main complementary function to the Tumult framework and GoogleDP library.

3.2.1 Transformations: Filter, Rename and Select

The OpenDP library supports the filtering of rows based on boolean-valued columns. This means that, in order to filter rows according to some predicate P, the results of $P(x)$ for all rows x must be saved to a column in the dataset before uploading it. This is contrary to the dynamic filtering described in the WebDP API, where the filter is formulated at the discretion of the analyst. However, instead of using this limited way of filtering rows, or outright forbidding the use of filters, the module uses Pandas [38] to pre-process the data before making the differentially private measurements. As was discussed in Section 2.1, filtering does not change the amount of noise needed to preserve differential privacy. Renaming columns and selecting columns do not change the amount of noise needed either, so the decision was made to leverage the functionality of the Pandas library to enable these steps as well.

3.2.2 Measurements: Sum, Count and Bin

OpenDP has functionality for summing float and integer typed columns, counting rows of any column type and counting by category if the category has type boolean, integer, enum or string. Even though the WebDP API states that the column field of any measurement is optional, its use is enforced in the OpenDP connector. The reason is that the OpenDP library expects a valid column name whenever it makes a measurement.

The OpenDP function `count_by_categories(·)` is a combination of *BinTransformation* and *CountMeasurement*. The *BinTransformation* contains instructions of how to create the bins (categories) for the column of interest, and *CountMeasurement* contains instructions of how to count the bins (categories). More specifically, the *BinTransformation* does not have fields that determine how much budget to spend and from which distribution to sample the noise, whereas *CountMeasurement* does.

The `count_by_categories(·)` function counts occurrences of a category member by equivalence, rather than comparison or some other partitioning predicate. This means that partitioning integer-valued data into integer bins, such as *10-19, 20-29*, cannot be achieved by determining whether a value lies within one of the two intervals. Instead, the counts of all values within each interval must be calculated separately before the results can be grouped into the bins. However, consider the following example as to why grouping the results could be a bad idea: An analyst makes a histogram query on some integer-valued column with the bins *1-3, 4-6, 7-9*, spending ε_1 of their budget. The analyst then makes another histogram with the bins *1-4,5-9*, spending an additional ε_1 for a total expense of $2\varepsilon_1$. The only difference between the queries is the grouping of the results: The values 1 to 9 are counted in both instances. This means that the analyst paid twice for essentially the same

query, which is easily avoidable by simply returning the results without grouping them. For this reason, whenever the bins are integer-valued, the OpenDP connector will return the counts of each separate value between the min and max values of the bins. When the bin values are of type string or boolean, the OpenDP connector will return the counts of these values without any interpolation. Histograms of double-typed columns are not supported.

3.2.3 Measurements: Mean

In order to make a mean measurement with OpenDP, the dataset has to be resized to a given number of rows r. If r is smaller than the actual number of rows, then the resulting dataset will be a random sample of the original dataset. If r is larger, then a user-set default value, v, is used to fill in the missing rows. The OpenDP documentation suggests that a DP release of the number of rows should be used for r, whenever the number of rows of a dataset is private information [39]. However, it is not possible for a user to send either r or v to the connector in the current state of WebDP. The option left is to calculate the mean with a DP sum and count measurement, in which case support of the *MeanMeasurement* step is not justified. The reason is that calculating the mean in the back-end by post-processing a DP sum and count offers no extra utility to the user. The user will in that case lose control over how much of the query's budget to spend on the sum and count, respectively.

3.2.4 The utility of the mean function

The OpenDP mean function supports pre-evaluation accuracy estimations, which is its main advantage over the method of post-processing a DP count and DP sum (which does not support it). However, due to the necessary resize operation before the mean can be calculated, the pre-evaluation accuracy can give a catastrophically wrong impression when the analyst overestimates the number of rows (r) and fails to give an accurate enough estimate (v) of the true mean.

Figure 3.3 shows simulations of a series of mean calculations using the OpenDP mean function. The dataset contained 1000 rows of normally distributed random data, sampled around the mean $\mu = 0$ with the standard deviation $\sigma = 1$. Each box is the result of 100 mean calculations, where the x-axis shows by which percentage the r parameter deviated from the true row count (1000). The y-axis shows the signed error of the estimation, where below 0 means an underestimation. The parameter v was set to one standard deviation below the true mean as a way to simulate a bad, but not catastrophic, guess. The figure shows that when the analyst overestimates the number of rows by more than 5%, then the measurement itself gets more precise (narrower whiskers and boxes), but the true value is highly unlikely to lie within its range. This is because 5-20% more rows have been injected into the data (in the resize operation), all containing the value $\mu - \sigma$. Of course, if $v = \mu$, then no bias will be injected, but it is probably much to ask of an analyst to accurately estimate the very thing that they wish to measure – especially since the estimation may bias the measurement, and thus ruins any possibility of asserting the correctness of the guess. The wider whiskers of the boxes where the number of rows is underestimated

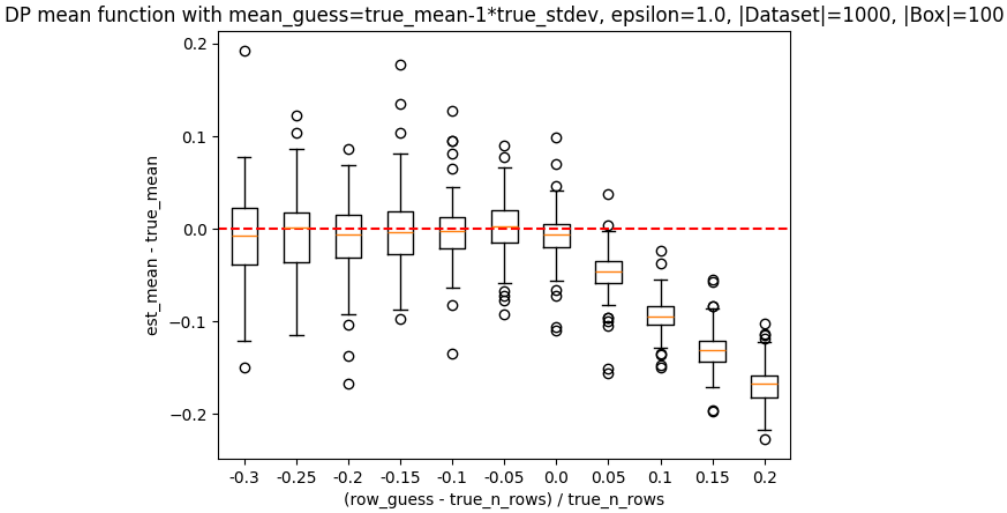

Figure 3.3: Box width: $IQR = |Q_1 - Q_3|$, whiskers: $Q_1 - 1.5IQR$, $Q_3 + 1.5IQR$; filled lines represent the box median, and the dotted line represents 0.0. Circles indicate outliers. The plot illustrates how the utility of an OpenDP mean measurement degrades when an analyst overestimates the number of rows—even for a non-catastrophic *a priori* estimation of the mean.

are due to sample error. The results of Figure 3.3 are consistent with the figures shown in the OpenDP documentation [40].

A possible strategy for an analyst that wishes to avoid the problems shown in Figure 3.3 could then be to make a DP count for some portion of the budget and subtract from the result the pre-evaluation accuracy estimation for some high level of confidence. For example, if a DP count of the dataset used in Figure 3.3 evaluates to 941, and the absolute error is estimated at 100 at a confidence of at least 0.99, then $941 - 100 = 841$ will most certainly not be an overestimation, and would therefore likely not bias the measurement.

Figure 3.4 shows the results of putting the aforementioned strategy to the test. The dataset is the same as in the experiment that produced Figure 3.3. The x-axis shows the percentage, p, of the total budget ($\varepsilon = 1$) that was spent on the DP count (per mean measurement). The remaining percentage $(100 - p)\%$ was spent on the OpenDP mean function. For reference, the same test was made for the post-process method (Figure 3.5), where the portion of the budget not spent on counting was spent on summing (per mean measurement). Thus, each measurement made, in both experiments, had a privacy loss of no more than $\varepsilon = 1$.

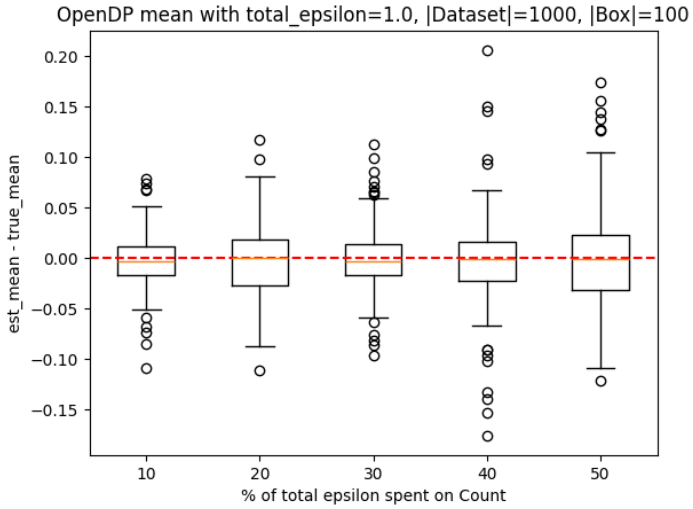

Figure 3.4: Box width: $IQR = |Q_1 - Q_3|$, whiskers: $Q_1 - 1.5IQR$, $Q_3 + 1.5IQR$; filled lines represent the box median, and the dotted line represents 0.0. Circles indicate outliers. The plot illustrates the signed errors of OpenDP mean measurements with $r = \tilde{C} - a_C$, where $\tilde{C}$ is a DP count and a_C is the pre-evaluation accuracy of $\tilde{C}$ at a confidence level of $\beta = 0.999$.

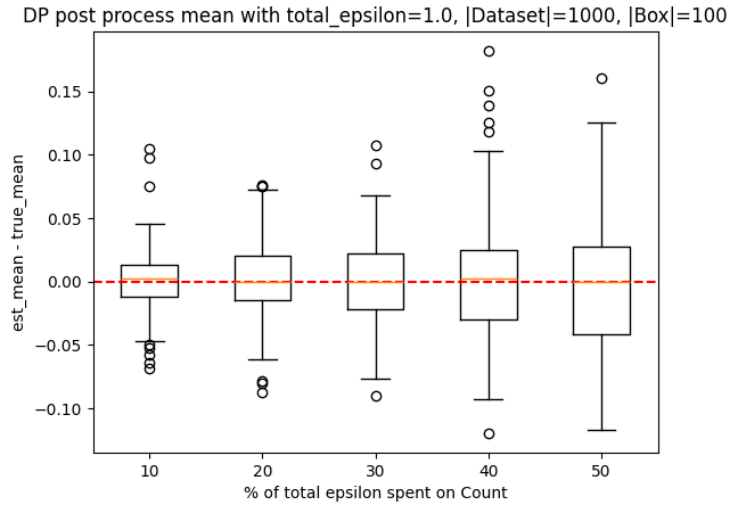

Figure 3.5: Box width: $IQR = |Q_1 - Q_3|$, whiskers: $Q_1 - 1.5IQR$, $Q_3 + 1.5IQR$; filled lines represent the box median, and the dotted line represents 0.0. Circles indicate outliers. The plot illustrates the signed errors of post-processed DP sums and counts to produce a mean.

A comparison of Figures 3.4 and 3.5 reveals two observations:

1. The strategy of avoiding over-estimations of the number of rows (r) eliminates bias.

2. Narrower boxes are observed when using the OpenDP mean. This suggests that errors tend to cluster more closely around 0 with the OpenDP method.

Again, this is consistent with the figures in the OpenDP documentation [40], although their comparison includes only the case when the post-process method splits the budget 50/50 (count/sum) and the OpenDP mean splits the budget 20/80 (count/mean).

3.2.5 The relative error of a DP count measurement

The negative consequences of overestimating the number of rows, as shown in Figure 3.3, can be put in perspective. For a total budget of $\varepsilon = 1$ to spend on a mean, where 30, 40 or 50 percent is spent on counting, the relative errors of the measurements are no more than 1.6% (see Listing 3.1) at a confidence level of 0.99, for a dataset size of 1000.

```
def dp_count(eps: float, data:str):
    t = lambda scale: (
        sel >> # loading data and selecting a column
        then_count() >>
        then_base_discrete_laplace(scale)
    )
    scale = binary_search_param(
        make_chain=t, d_in=1, d_out=eps
    )
    return scale, t(scale)(data)

c_03 = dp_count(0.3, DATA)
c_04 = dp_count(0.4, DATA)
c_05 = dp_count(0.5, DATA)

def print_acc(x, conf):
    acc = discrete_laplacian_scale_to_accuracy(x[0], 1 - conf)
    print(acc)

print_acc(c_03, 0.99) # = 15.81
print_acc(c_04, 0.99) # = 11.96
print_acc(c_05, 0.99) # = 9.64

# 15.81 / 1000 = 0.01581 < 1.6%
```

Listing 3.1: A Python program that calculates pre-evaluation accuracies of DP counts at different budgets ($\varepsilon \in \{0.3, 0.4, 0.5\}$) using the OpenDP library. The confidence level is 0.99.

If the dataset contained 10,000 rows instead, an absolute error of 15.81 would result in a relative error of 0.15%. Therefore, when calculating the mean with the OpenDP mean function, using a DP count as an estimate for the number of rows significantly reduces the likelihood of introducing large sample- or bias-related errors as the dataset grows in size – regardless of whether the strategy described is used.

3.2.6 Reducing noise of DP sums

To perform a DP sum on a column using the OpenDP library, the column must be "clamped" to a closed interval, meaning that the values of the column are forced to lie within the bounds of the interval. For example, if we have the interval $[0, 2]$ and wish to sum the column $[-1, 1, 3]$, then -1 will be clamped to 0 and 3 will be clamped to 2, resulting in the column sum $0 + 1 + 2 = 3$. When the column type is integer, the sensitivity of the summing function

$$\max(|L|, U)$$

for a lower bound L and an upper bound U [41]. When the column contains values of type float, then the sensitivity is

$$\max(|L|, U, U - L)$$

As can be seen, the summation functions for integers use the unbounded DP definition, whereas bounded DP is used for float values. The reason is that there is a discrepancy between real numbers and floating-point approximations of them, which breaks differential privacy guarantees [41]. The solution has been to limit the number of float-valued rows one can sum [41], which means that only bounded DP can be guaranteed.

In WebDP, the dataset curator specifies the domain for each column during dataset creation, which includes defining lower and upper bounds for numeric types. These are the bounds that are applied to clamp the column values in all queries made on that particular column. However, the approach of only allowing curator-set bounds restricts analysts from making sums with tighter bounds, potentially limiting the accuracy of their measurements in certain scenarios. For example, consider a car-selling company that sells two brands of cars: Reds and Blues. The company maintains a dataset of all their sales, where each row contains the price ($Price$) of a sale and the brand ($Brand$) of the car being sold. The 2024 prices of new Reds and Blues range from 350,000 SEK to 1,000,000 SEK, which also is the integer interval on which the $Price$ column is defined. Now consider an analyst wanting to know the sum-total price of all new Blues sold by the company. The analyst would send the query in Listing 3.2 to WebDP.

```
        ...
        "budget" : {
            "epsilon" : 1.0
        },
        "query" : [
            "filter" : ["Brand == \"Blue\""],
            "sum" : {
                "column" : "Price",
                "mech"   : "Laplace"
            }
        ]
        ...
```

Listing 3.2: A WebDP query to the OpenDP connector: The query sums the selling-price of Blues.

The pre-evaluation accuracy of the query in Listing 3.2 is approximately 4,600,000 SEK at a confidence level of 0.99. However, upon examining a (public) Blue price list, it is evident that the most expensive new Blue sells for approximately 650,000 SEK. Thus, an analyst could be confident that summing the column with the clamp bounds $(350,000,\ 650,000)$ would not significantly alter the true accuracy of the result, if at all, since the column values fall comfortably within the specified clamp bounds. The accuracy of the query in Listing 3.2 with the new bounds would be approximately 2,700,000 SEK at a confidence level of 0.99 – a clear improvement.

3.2.7 The syntax of a query

At the heart of the OpenDP connector is the *DPQueryBuilder*, whose classmethods are listed in Listing 3.3.

```
class DPQueryBuilder:
    def __init__(self, privacy_notion, column_schema, data)
    def apply_select(self, columns: List[str])
    def apply_rename(self, rename_mapping: Dict[str, str])
    def apply_filter(self, filters: List[str])
    def add_bin(self, column: str, bins: List[any])
    def make_sum(self, column: str, mechanism: Mech)
    def make_count(self, column: str, mechanism: Mech)
    def add_noise(self, budget: Budget, discrete: bool)
    def evaluate(self)
    def accuracy(self, confidence: float)
    def validate(self)
```

Listing 3.3: Outline of the DPQueryBuilder methods.

This query builder operates as a finite automaton, incrementally assembling a DP query using the OpenDP and Pandas libraries. An advantage of this design choice is that the syntax is easy to understand and demonstrate (see Figure 3.6 and Listing 3.4). The builder offers a Java Stream-like interface, letting the user chain operations.

```
# initialisation of column schema and pandas dataframe above
builder = DPQueryBuilder(
    privacy_notion=PrivacyNotion.PureDP,
    column_schema=a_column_schema,
    data=pandas_dataframe
)

result = builder.apply_select(["column1", "column2"])\
                .apply_rename({"column1" : "new_name"})\
                .apply_filter(["new_name > 100", "column2 < 100"])\
                .make_count(
                    column="new_name", mechanism=Mech.LAPLACE)\
                .add_noise(
                    budget=Budget(epsilon=0.1), discrete=True)\
                .evaluate()
```

Listing 3.4: A differentially private count query build with DPQueryBuilder.

As illustrated in Figure 3.6, a DPQueryBuilder is designed to perform exactly one differentially private measurement. While this does not inherently prohibit a WebDP query from containing multiple measurements, this restriction is enforced in the OpenDP connector. This decision also implies that the *BinTransformation* is constrained to include only a single column-bin mapping. Thus, the valid sequences of query steps sent from the WebDP sever are on the same form as in Figure 3.6.

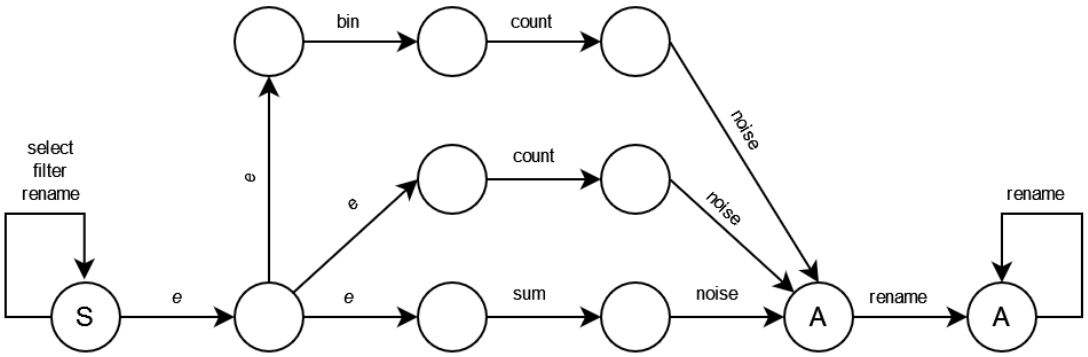

Figure 3.6: A NFA of the DPQueryBuilder's allowed sequences of operations. S is the starting state, A the accepting states and e is the empty transition.

3.3 Additional connectors

To validate the architectural decision to design the WebDP API as a microservice-based system, and as a proof of concept, two additional connectors – for Tumult Analytics and GoogleDP – were implemented and integrated alongside the OpenDP connector.

The *Tumult connector*, written in Python, was the second connector integrated into the proposed WebDP API. It closely replicates the Tumult implementation in the current WebDP API. Administrative features such as user, budget, and dataset management were omitted, while the query functionalities largely remain the same. The function set of the Tumult connector are the same as in the current WebDP implementation.

The third connector integrated into the proposed WebDP API is the *GoogleDP connector*, leveraging the GoogleDP library [30]. Since both the OpenDP and Tumult connectors were implemented using Python, the GoogleDP connector was developed in Go to demonstrate the language agnosticism inherent in the proposed WebDP architecture. The functions that the GoogleDP library offers are, at the time of writing, only measurements such as count, sum, mean, variance and quantiles, in addition to the noise generator. This means that if transformation functions are desired, it is up to the developer of the connector to implement them. In addition to the offerings in the standard library the connector also offers filtering and binning functionality. Figure 3.7 describes the sequence of query steps that is accepted by the GoogleDP connector. The following sequences are accepted:

- Measurement
- Bin → Measurement
- Filter → Measurement
- Filter → Bin → Measurement

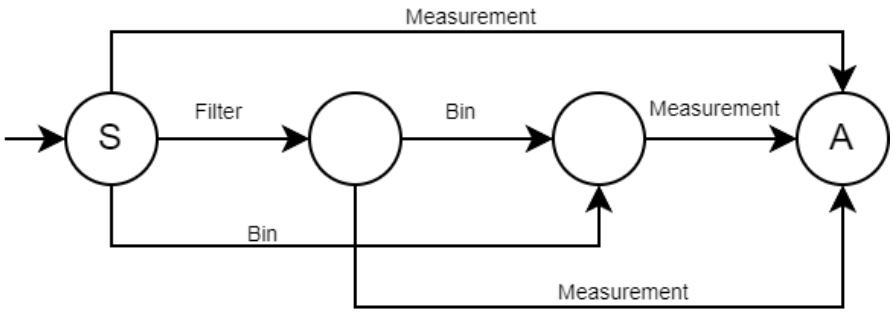

Figure 3.7: A DFA of the GoogleDP QuerySteps allowed sequences of operations.

3.3.1 Handling engine choices and default engines

There are multiple ways to handle sending queries to engines, if there are more than one engine: If a query operation is supported by multiple engines, either the user or the back-end needs to choose to which engine to route the request. On the user side, you could either set a default engine as part of the user entity (by the administrator) or dataset entity (by the dataset curator) and override it when appropriate, or include a preferred engine as a parameter of every request. However, these options would all require significant changes to the API.

The decision landed on setting a default engine across the application, while providing analysts with the capability to modify their engine selection while sending query requests by entering a query parameter in the URL (see Section 3.5 for details on the query parameters). The rationale behind this decision stemmed from a reluctance to impose choices upon users through the application. Moreover, granting curators the authority to restrict engine options for their datasets was deemed to introduce unnecessary complexity. The notion of limiting engine choices was primarily considered in cases such as when a particular engine had known vulnerabilities. However, expecting all curators to possess thorough knowledge of each engine to make such decisions seemed impractical.

When a query is not supported by an engine, the request will simply be denied. To aid the user in choosing a compatible engine for their query, some new helper endpoints for validating queries against different engines, getting lists of functions and reading the documentation have been implemented (see Section 3.5). This lets the user validate a query before evaluating it, which could potentially save some budget during evaluation.

3.4 Persistence and databases

One of the major features missing in the current implementation of the WebDP server is persistence of users, datasets and budgets (allocated and consumed) across sessions, meaning that you would have to complete tasks in one go or start over, should a session be lost. To achieve persistence, a Postgres database was implemented as part of the WebDP module. This allows users to log in and out without losing their progress. The relational setup is presented in Appendix B.

3.5 Improvements to the API

The transition from DPella's initial WebDP proof-of-concept (version 1) to the proposed implementation (version 2) underscores a focus on seamless engine integration. This evolution prompts adjustments to the API specification. In this section, the new endpoints in version 2 are introduced, along with explanations of the process of engine selection within the updated API and additional minor API changes. For the full list of changes to the API endpoints, see Table 3.1.

3.5.1 New endpoints

Due to the increase in the scope of the API – so that multiple frameworks can be supported – there is a need to introduce new endpoints to the API to handle the new functionality in a clear and accessible manner. Therefore, endpoints for getting a list of supported engines, getting lists of functions of all or of specific frameworks, and getting each framework's documentation are introduced to the proposed API specification. In addition, an endpoint letting users validate queries to engines before evaluating them is proposed.

The *engines* endpoint, `/queries/engines`, returns a list of enabled engines to the user. The use case for this endpoint is to inform the user of which engines they can use in their queries:

```
["engine1", "engine2", "engine3"]
```

Listing 3.5: Sample response: Engines endpoint.

The *documentation* endpoint, `/queries/docs?engine={engine}`, returns a markdown file where the details of the engine implementation are specified. The goal of this endpoint is to give a better understanding of the framework, and the function set the framework provides:

```
# Title
{Description}

## {Engine} supported functions
{Table of supported functions}

### Information
{Information}

## Endpoints
{Table of endpoints}
```

Listing 3.6: Sample response: Documentation endpoint.

The *functions* endpoint, `/queries/functions`, returns a JSON object which includes the transformations and measurements that an engine supports. The goal of this endpoint is to assist the user in building queries. There are two uses: When calling `/queries/functions` the API will return all supported query steps for all engines. When calling the endpoint with a query parameter specifying the engine name, `/queries/functions?engine={engine}`, the supported set of query steps of that engine is returned:

```
// all engines
{
    "engine1": {
        "function1": {
            ...
        },
        "function2": {
            ...
        }
    },
    "engine2": {
        "function1": {
            ...
        },
        "function2": {
            ...
        },
        "function3": {
            ...
        }
    },
    "engine3": {
        "function1": {
            ...
        }
    },
    ...
}
```

```
// specific engine
{
    "function1": {
        "enabled": true,
        "required_fields": {
            "req1": "description",
            "req2": "description"
        }
    },
    "function2": {
        "enabled": true,
        "optional_fields": {
            "opt1": "description"
        }
        "required_fields": {
            "req1": "description"
        }
    },
    "function3": {
        "enabled": true,
        "required_fields": {
            "req1": "description",
            "req2": "description"
        }
    },
    ...
}
```

Listing 3.7: Sample response: Functions endpoint, all engines and specific engine. Functions may have required fields and optional fields for formatting the query.

The *validate* endpoint, `/queries/validate`, is implemented to give the user a way to validate their query without spending any budget, meaning that a user can opt to validate their query before evaluating it to ensure that the evaluation will behave as intended before investing budget into it. The way it is implemented gives the user two choices. The first is to validate a query on all engines; the result is a JSON object, which indicates whether an engine would be able to run the query or not. The second option is to send a request to the endpoint using a query parameter, where the user can specify an engine and only validate the query on the specified engine: `/queries/validate?engine={engine}`.

The functionality is implemented as an extension to that of the *evaluate* endpoint. Using the fact that the DP frameworks do not keep track of the budget, the query is "evaluated" and the result is discarded before the response – whether the evaluation was successful or not – is returned. This is, however, as computationally expensive as an *evaluate* call. See Section 4.3.2 for further discussion on this topic.

```
// all engines
{
    "engine1": {
        "valid": True,
        "status": "description"
    },
    "engine2": {
        "valid": True,
        "status": "description"
    },
    ...
}
```

```
// specific engine, successful validation
{
    "valid": True,
    "status": "description"
}
```

```
// specific engine, failing validation
{
    "valid": False,
    "status": "description"
}
```

Listing 3.8: Sample responses: Validate endpoint.

3.5.2 Example calls

The below section will showcase how calls to the new endpoints can be made and what the results look like. The first example that is showcased can be seen in Listing 3.9; this is the result of calling the `/engines` endpoint. As the extended WebDP hosts multiple DP engines, this endpoint will give the user insight into which engines are available for use.

```
["tumult", "opendp", "googledp"]
```

Listing 3.9: WebDP enabled engines, `/engines` call.

As an example, a call to validate a *count* query will be made. See Listing 3.10 to see the structure of the body.

```
{
    "budget": { "epsilon": 0.2 },
    "dataset": 1,
    "query": [
        { "count": { "column": "age", "mech": "Laplace" } }
    ]
}
```

Listing 3.10: WebDP validation of a `count` query, `/validate` call.

The result from the query in Listing 3.10 will give a list of engines able to run said query; see Listing 3.11 for the result. As can be seen, all available engines are able to run the *count* query.

```
{
    "googledp": {
        "status": "query is valid in GoogleDP",
        "valid": True
    },
    "opendp": {
        "status": "query is valid in OpenDP",
        "valid": True
    },
    "tumult": {
        "status": "query is valid in tumult",
        "valid": True
    }
}
```

Listing 3.11: WebDP validation result of a `count` query, `/validate` call.

The request body for validation and evaluation is identical; see Listing 3.10 for the structure of the request body. Choosing a specific engine can be done using the query parameter; using the same body as in the previous example (Listing 3.10) and sending the request to the URL `/evaluate?engine=googledp` will give the result shown in Listing 3.12.

```
{
    "rows": [{ "age_count": 993 }]
}
```

Listing 3.12: GoogleDP `count` evaluation, `/evaluate?engine=googledp` call.

The query validated in Listing 3.10 will be validated in all available engines. However, not all queries will be able to run on all engines: See Listing 3.13 for an example, and Listing 3.14 for the result of the validation.

```
{
    "budget": { "epsilon": 0.2 },
    "dataset": 1,
    "query": [{ "min": { "column": "age" } }]
}
```

Listing 3.13: WebDP validation of a `min` query, `/validate` call.

```
{
    "googledp": {
        "status": "unknown query step type: min",
        "valid": False
    },
    "opendp": {
        "status": "the min measurement is not supported",
        "valid": False
    },
    "tumult": {
        "status": "query is valid in tumult",
        "valid": True
    }
}
```

Listing 3.14: WebDP validation result of a `min` query, `/validate` call.

When trying to evaluate a query that has failed to validate, for example if evaluating the query in Listing 3.13 using GoogleDP, the result showed in Listing 3.15 is returned.

```
{
    "detail": "something bad happened: unknown query step type: min",
    "status": 500,
    "title": "Unexpected error",
    "type": "Unexpected error"
}
```

Listing 3.15: GoogleDP failed evaluation of a `min` query result, `/evaluate?engine=googledp` call.

3.5.3 Other changes

The API paths for budgets, datasets, users and queries have been updated to common path naming conventions, using plurals for entities. For accessibility, the OpenAPI specification has been added to the *spec* endpoint, `/spec`. This endpoint returns a Swagger UI web page, running locally, for viewing the specification – mirroring the OpenAPI view endpoint of the current WebDP server implementation. The specification endpoint does not require authentication to access. With the new option to run queries on different engines, and with the *validate*, *docs* and *function* endpoints to aid the user in constructing queries for each engine, the *custom* endpoint provides no additional functionality, hence the custom endpoint was removed from the updated specification. See Section 4.3.2 for the reasoning as to why the custom endpoint was removed.

Category	Current: /v1/	Proposed: /v2/
Budgets	/budget/allocation/ /{user}/{dataset} /budget/dataset/{dataset} /budget/user/{user}	/budgets/allocations/ /{user}/{dataset} /budgets/datasets/{dataset} /budgets/users/{user}
Datasets	/datasets /dataset/{dataset} /dataset/{dataset}/upload	/datasets /datasets/{dataset} /datasets/{dataset}/upload
Authentication	/login /logout	/login /logout
Users	/users /user/{user}	/users /users/{user}
Queries	/query/accuracy /query/custom /query/evaluate - - - -	/queries/accuracy - /queries/evaluate /queries/docs /queries/engines /queries/functions /queries/validate
Specification	-	/spec

Table 3.1: The updated APIs. The paths use plural forms and the /queries/ endpoint have been expanded.

3.5.4 The internal network

The proposed WebDP server design allows implementation of multiple connectors to various libraries and frameworks, irrespective of their supported languages. However, this introduces some data access and data consistency challenges. To that end, an internal network was introduced to facilitate communication between WebDP and the connectors to enable data transfers. Two additional endpoints were introduced to allow this, one endpoint in WebDP and one endpoint for every connector; see figure 3.2. The GET method /datasets/{datasetId} is used by the connectors to fetch the dataset from WebDP, and the DELETE method /cache/{datasetId} is used by WebDP to delete the dataset from the connector cache only if the connector has implemented a dataset cache.

Part	Method	Path
WebDP	GET	/datasets/{datasetId}
Connector	DELETE	/cache/{datasetId}

Table 3.2: Internal data handling methods.

3.5.5 Connector API specification

The connectors need to adhere to a specific API specification to enable communication between WebDP and the connectors. Additionally, future connector implementations must also comply with this specification to facilitate seamless integration of the connector. Refer to Table 3.3 for details on the API specification.

Method	Internal path	WebDP path
`POST`	`/evaluate`	`/queries/evaluate`
`POST`	`/validate`	`/queries/validate`
`POST`	`/accuracy`	`/queries/accuracy`
`DELETE`	`/cache/{datasetId}`	`/datasets/{datasetId}`
`GET`	`/functions`	`/queries/functions`
`GET`	`/documentation`	`/queries/docs`

Table 3.3: Connector API specification. Internal endpoints for the engine connectors, with the corresponding WebDP endpoints.

The endpoints specified in the connector API, as depicted in Table 3.3, closely align with those in the WebDP API. When a user invokes the `/queries/evaluate` endpoint in WebDP the request undergoes transformation (see Listing 3.16 for an example) and is then dispatched to the connector endpoint `/evaluate`. Since the *validate* request mirrors the structure of *evaluate*, the same procedure applies for `/queries/validate`. Upon a user's deletion of a dataset, WebDP triggers calls to `/cache/{dataset}` on all connectors to purge the cached dataset from each connector. When `/queries/functions` is invoked in WebDP without a query parameter, WebDP, in turn, calls the endpoint `/functions` on all connectors, compiling a JSON object from their responses. Conversely, if a query parameter is provided, WebDP calls only the designated connector and returns the outcome to the user. Furthermore, when `/queries/docs` is invoked by the user in WebDP, WebDP initiates a call to the `/documentation` endpoints, returning a markup page containing the documentation for each corresponding connector.

```
{
    "budget": { "epsilon": 0.2 },
    "dataset": 1,
    "query": [
        {
            "count": {
                "column": "age",
                "mech": "Laplace"
            }
        }
    ]
}
```

```
{
    "budget": { "epsilon": 0.2 },
    "dataset": 1,
    "query": [
        {
            "count": {
                "column": "age",
                "mech": "Laplace"
            }
        }
    ],
    "schema": [
        {
            "name": "age",
            "type": {
                "name": "Int",
                "low": 0,
                "high": 100
            }
        }
    ],
    "privacy_notion": "PureDP",
    "url": "webdp/datasets/1"
}
```

Listing 3.16: Left: Example query sent by user. Right: The same query transformed by WebDP to be sent to the connector.

3.5.6 Specification management in WebDP

In Python servers generated with OpenAPI Generator, the routing to the controllers is done via an OpenAPI Specification (OAS) using Python's *connexion* [42] library. The library lets developers register an API directly from a specification file, allowing for fast and easy specification-driven development using OpenAPI/Swagger. This does have some benefits: The routing paths are only hard-coded in the specification and not in the source code, meaning that when you update your API specification, you can load it into the code base to easily implement the necessary changes.

There have been efforts to build OpenAPI toolkits in Go, the most prominent being *go-openapi* [43]. In a similar way to how you would register an API using *connexion*, *go-openapi* lets you load an OAS file to implement its endpoints and validate them against the specification [44]. However, the library only supports OAS 2.0 and although an update to version 3.x has been requested, the development has been discontinued [45] and libraries supporting OAS 3.x are still at an early stage [46] – meaning that, at the time of writing, any new specifications would have to be

downgraded to OAS 2.0 to be usable with the library. The alternative is updating the specification separately from the code base. Not only does this slow down the workflow, but it also increases the surface area for errors.

3.6 Testing

As the final product intends to obscure individuals' private information, it is crucial to make sure that there are no data leaks in the application. In order to ensure the guarantees of each DP model hold, it is a priority to conduct several forms of testing to ensure valid and reliable results. Therefore the test suites include unit tests, integration tests and regression tests to ensure continuity with respect to the current WebDP implementation.

4

Discussion

In this chapter, the results of the project and its implications are discussed. These cover considerations regarding data privacy practices for both the community and developers, the important design choices made during development of WebDP, and research in the ever-changing data privacy domain.

4.1 Public interests vs company interests

Data privacy has, and always will be, an important part of society's endeavour toward trust in digital interactions and companies. This trust will also help in upholding the individual's fundamental rights in how the data should be used, modified or removed [47]. Presuming the adoption of differential privacy services improves data privacy practices, not only would it support society and the rights of the individual, but it would also have a positive impact on how data is utilised more broadly. By ensuring that an individual's sensitive information remains protected, actors would prevent corporations, organisations and governments to use personal data in unethical manners.

This ethical question about the actors' interests can be directly related to this thesis. DPella AB is a company founded by researchers with an interest in providing a more secure and trusting landscape in the digital world. Seeing how Swedish industries are lagging behind in adopting modern privacy tools, their goal is to develop an accessible DP service to lower the adoption barrier [15]. The product will be useful for fellow researchers as well as companies interested in adhering to privacy laws and regulations like EU's GDPR and Title 13 in the United States; due to definitions in mathematical and legal fields being disjoint, these laws can be tricky to follow when developing any kind of product concerned with data privacy [48]. However, differential privacy implies security against *predicate singling out attacks* in the sense of complying with both the legal and mathematical aspects [48], making it a better choice for data privacy compliance than its predecessors.

However, we should keep in consideration the companies' motivation and aspirations as they are ultimately the ones who are responsible for how data is stored and used. This means that DPella will have responsibility in the effectiveness of their product – not only for the sake of their clients, but for the privacy community as a whole

– and take the lead in aiding and educating business owners, data analysts and developers in privacy "done right".

4.2 Developers' responsibilities

In the context of IoT devices for medical use, Rasool *et al.* argue that "it is important to teach the system designers ethical approaches for data acquisition, storage, and usage" [49, p. 10], further mentioning differential privacy as a technique that ensures ethical data usage.

It is clear that developers' design choices affect what personal data can be leaked, and to what extent. If developers do not implement privacy mechanisms in their systems' data handling modules, introduce bugs, or in other ways allow personal data to be exposed, they open up for errors (and attacks) to be made. In 2022, 59% of all personal data breach incident reports in Sweden were due to human errors [50]; this number would likely be lower if the data was handled privately and ethically *before* being exposed to error-making humans. Hence, it is of great importance not only make sure the privacy guarantees that DP provides are persistent throughout various implementations, and that the data coming in and out of the queries is protected, but also to make sure the system as a whole is reliable and rigid, and that the raw data cannot be reached (and by extension leaked) by unauthorised persons. However, finding all possible errors and vulnerabilities in the code is difficult even for experienced programmers: For example, in Myers' classical triangle test, highly qualified professional programmers could only find 56% of the possible test cases, on average [51]. Generally, it is difficult to prove that a program works and will continue to work the way you intended it to. Even though effort was put into testing, the possibility of having missed crucial test cases is rather high.

4.3 Design choices

For the resulting product to meet the expectations and goals set in this project, some design choices were made during the development. These choices, and their advantages and limitations, are discussed in this section.

4.3.1 Modular engines

Since the engine connectors are separated and spun up as containers as per the microservices architecture, users can disable any engine connector by removing its corresponding entry in the list of available engines – a plain-text configuration file in the WebDP project directory. This proves very convenient if a DP library is found to have security vulnerabilities or ethical issues, if it poses a conflict of interest, or has other flaws. Actors running WebDP instances can disable any undesired DP engine by removing the configuration entry, or alternatively, by closing down the container in which it runs.

One of the most important questions is that of choosing a default engine for the application. On the one hand, you want to help users by providing them with the best functionality, even if that means using a proprietary engine. On the other hand, you could use the most open, transparent engine because it is the most ethical choice, even if that means sacrificing some of the functionality. To make things even more complicated, there are some other scenarios to consider: If a company pays you to set their engine as default, even though it might not be in your user's best interests, would you do it? If your preferred engine starts taking a fee for each query you run, would that affect your choice? Whatever the choice, transparency towards the users is key; the reasons for choosing a particular engine or framework as default should be clearly stated in the documentation and user manuals. Granted, a client running their own instance of WebDP can specify their preferred default engine in its configuration files.

4.3.2 API implementation

There is no guarantee that a third-party engine will be able to execute a given query and return a result, even if the structure of it looks correct and is allowed by WebDP. To make the interface more user-friendly, and to minimise the number of malformed or incorrect queries, each connector implements the previously mentioned *validate* endpoint (see Section 3.5.1) that lets a user submit a query and get a "Yes" or "No" answer for whether it would successfully execute. In theory, this would reduce bandwidth usage between the engines and the database since it only needs to retrieve the schema of the chosen dataset. All the needed information (the column names and their types) is given by the schema, meaning that one could ensure the existence of the desired columns specified in the query, and ensure that the type constraints of a function used on a column coincides with that of the correlating schema, from that very schema.

However, as seen with OpenDP's *validate* endpoint, it instead executes the query like an *evaluate* call and simply omits the result in the response. This approach is as computationally expensive as a normal evaluation, making it an inefficient way to validate a query. Evaluation involves several sub-tasks: Confirming the user's permission to query the dataset, ensuring the user has sufficient budget, and verifying that the query steps can be executed by the connector. The WebDP server handles the first two, while the connector addresses the third. To avoid the inefficiency of evaluating queries for validation, a state machine similar to those presented in Sections 3.2 and 3.3 with specific checks on the query steps can be employed. This would also protect the application from timing attacks – i.e. attacks analysing the time spent evaluating the data – which the current implementation is vulnerable to, especially considering the validation does not consume any budget.

The current WebDP API includes the endpoint `/query/custom`, which is meant to evaluate a query that is written in a custom language. That is, it has the same functionality as `/query/evaluate`, with the difference that the query is formulated in a text string that is then parsed on the server, instead of using the specified query steps. The proposed WebDP API does not support the evaluation of a query

written in a custom language. The reasoning behind this is that implementing a *custom* endpoint would entail making contra-productive adaptations to the code. The *custom* endpoint could be implemented in two ways:

1. The query string is transpiled to query steps and is thereafter interpreted in the same way as a query to the *evaluate* endpoint.

2. There is a parser on the server which translates the query string into something that is inexpressible with the available query steps.

The first solution is arguably a front-end feature: The functionality could be supported by making several calls to the *evaluate* endpoint. The endpoint only accepts well-formed queries. In computer security terms, this would be the *default deny* paradigm. The second solution undermines the API by introducing inconsistencies: There is simply no point in expressing inexpressible queries. Additionally, parsing the queries to see whether they are well-formed or not would be quite a challenge: Rather than add to the functionality, a custom query language might introduce security vulnerabilities similar to that of the infamous SQL injection.

4.3.3 Using code generators

Using code generators, as in the case for the proof-of-concept implementation of WebDP, is a quick and easy way to get a server up and running. However, there are obvious limitations to what a developer can achieve in terms of flexibility and suitability for the project's purposes and ambitions – code generators shine when setting up simple, standardised servers, which one could argue WebDP is not. Simply put: One size does not fit all. As discussed in Sections 3.1.3 and 4.3.1, not only does a custom-made, containerised architecture prove to be convenient for adding additional engine connectors in a straight-forward manner, but it is also able to run the engine containers separately to let users set up and take down engine modules as they please. The initial time spent on development will pay off in the maintenance and the longevity of the product.

4.4 Design suggestions

During development of the product and researching of DP APIs, several aspects of the implementations of the APIs were disputed for their theoretical and practical implications. This section takes a closer look on these design choices and discuss their significance and alternative solutions.

4.4.1 Extending the MeanMeasurement step for OpenDP

Section 3.2 presented a comparison between the OpenDP mean function and post-processing a mean value from a DP sum and count. The results revealed that overestimating the number of rows in a dataset can significantly impact the accuracy of the result. Specifically, at a confidence level of β:

1. As the analyst's estimation (r) of the number of rows (n) increases, the range of the OpenDP mean function narrows.

2. As the difference $r-n$ grows, the result of the OpenDP mean function becomes increasingly biased by the analyst's estimation of the mean (v).

This implies that a pre-evaluation accuracy assessment of an OpenDP mean measurement, particularly when r is large, would yield a narrow confidence interval. However, if $r \gg n$, an analyst may be misled by the narrow confidence interval, erroneously assuming that the result will closely align with the true mean. In reality, the result will closely resemble the *a priori* estimation, v, of the mean.

A strategy to remedy this was proposed, which was shown to be as accurate, if not more, than calculating the mean by the post-process method on a dataset with 1000 rows of normally distributed data and a total budget of $\varepsilon = 1$. In addition to being no worse than the post-process method, it has the advantage of letting the analyst make pre-evaluation accuracy estimations that do not overestimate the accuracy. However, any conclusions about the differences in performance for total budgets $\varepsilon \leq 1$ on small datasets (1000 rows or lower) cannot be drawn from observing the results in Section 3.2. There is a possibility that when applying the proposed strategy under these conditions, the mean is calculated using up to a 30% smaller subset of the data, compared to the post-process method that estimates the number of rows at $\pm 15\%$ around the true count (assuming a confidence level of 0.99 and a budget of $\varepsilon = 0.03$). The results also show that the strategy is less important for large datasets, as the relative error of a DP count gets very small.

As the dataset size increases, both methods of calculating the mean become more accurate. This is demonstrated in Theorem 3 for the post-processing method: If $\tilde{x}$ and $\tilde{y}$ represent a DP count and sum, respectively, of the same column, then both exhibit an expected rate of change that is linear in the dataset size. However, the denominator in the expression grows quadratically with the dataset size, causing the expected absolute error to tend towards 0 as the data grows. Figure 3.3 illustrates that the OpenDP mean also produces mean measurements within a narrowing range as the number of rows increases. Given that the mean function in OpenDP, in addition to being at least as accurate as the post-process method for datasets larger than 1000 rows, allows for pre-evaluation accuracy estimations, there is a compelling argument for introducing a new query step, e.g *MeanMeasurement2*, or extending the existing *MeanMeasurement* with fields for the analyst's row and mean estimations.

4.4.2 Enabling analyst-set numeric bounds in queries

Clamping numeric column values to an interval is a common technique used to introduce bounded noise in differentially private measurements. For example, the sum and mean measurements in Tumult Analytics, GoogleDP, and OpenDP all restrict the measurements' domains or images by clamping. The WebDP API acknowledges this trend in the definitions of datasets, where the curator defines numeric-typed columns on closed intervals. However, as discussed in Section 3.2, there are circum-

stances where an analyst may tighten the bounds to achieve more accurate results. It may therefore be advisable to include an optional field "bounds" in the query steps *MeanMeasurement* and *SumMeasurement*.

4.4.3 The OpenDP BinTransformation

Section 3.2 described how an analyst may create histograms by querying the OpenDP connector. When the column type is `text`, `enum` or `bool`, the behaviour is as one would expect: The count is returned for each category. When the column type is `int`, then the behaviour is maybe not as one would expect: The count of *all* values between the min and max value of the bins is returned. Given the restrictions of OpenDP's `count_by_categories(·)` function, this is a sound choice. The argument is that post-processing results in the back-end hides information from the analyst, who also cannot assess the accuracy of each bin as a consequence of the post-processing. However, when the categories are many in relation to the number of rows in the data, `count_by_categories(·)` can produce a far too volatile result to be of utility. By the theorem of parallel composition (Theorem 1), a query such as the one in Listing 4.1 would make $75,000$ ε-differentially private count measurements for $\varepsilon = 1$.

```
...
"budget" : {"epsilon" : 1.0},
"query" : [
    "bin" : {
        "salary_SEK" : [0, 15000, 30000, 45000, 60000, 75000]
    },
    "count" : {
        "column" : "salary_SEK",
        "mech"   : "Laplace"
    }
]
```

Listing 4.1: A query to the OpenDP connector: A histogram of salaries.

Assuming that the purpose of the bins in Listing 4.1 is to generate a total count of people with salaries falling within the specified ranges $[0,\ 15,000)$, $[15,000,\ 30,000)$, ..., $[60,000,\ 75,000)$, each bin would require summing $15,000$ differentially private count measurements. Consequently, each bin comprises $15,000$ independent random variables $X \sim \text{Lap}(\mu = \text{true count}, 1)$, resulting in a variance of each bin equal to

$$\begin{aligned}\textbf{Var}\left[\sum_{i=1}^{15,000} X_i\right] &= \sum_{i=1}^{15,000} \textbf{Var}\left[X_i\right] \\ &= \sum_{i=1}^{15,000} \textbf{Var}\left[\text{Lap}(1)\right] \\ &= 30,000.\end{aligned}$$

A variance of $30,000$ could be considered substantial, even for a large dataset. For reference, the true count of all bins would have to be at least $54,000$ to produce a relative error that is less than 1% with a probability of approximately 98.5% (using Chernoff bounds – see Appendix D). Alternatively, a method that partitions the data with mutually exclusive filters would yield as many measurements as there are bins, with each bin having a variance of $\mathrm{Var}\left[\mathrm{Lap}(\mu = \text{true count}, 1)\right] = 2$. The true count of the bins would in this case have to be approximately 800 to produce a similar accuracy as in the previous case. A similar argument applies to the Gaussian mechanism. This issue underscores a significant limitation of the OpenDP connector that requires rectification.

4.5 Future development

Throughout the project, a few noticeable differences have been found between DP libraries. For instance, one library might implement "Mean" while another implements "Average", or the same function might require a different number of parameters. This raised questions of standardisation for these libraries. However, since differential privacy (and implementations thereof) is a relatively new and emerging field, its maturity still lies in the future and it is still uncertain whether a standardisation is needed or even feasible at this point. Seeing how some large actors are developing their own DP libraries, one could speculate that an industry standard will emerge, making differential privacy services more uniform and readily available to the public through built-in frameworks and packages.

To aid in the user experience and the ease of use of DP tools, additional tools such as those implemented in this project – engine listing, query validation, functions and documentation – should be supported in the front-end application. It is therefore recommended to extend the front-end application so that it connects to the proposed (version 2) WebDP API. As mentioned in Section 4.3.2, the front-end could also support custom queries by making several calls to the *evaluate* endpoint.

As for the server, there is some work to be done in regards to optimisation. Notably, the *validate* endpoint can be re-written to validate queries without lifting the heavy load of the *evaluate* endpoint, as discussed in Section 4.3.2. This would also solve the timing attack vulnerability. Furthermore, the Python images are larger and slower than their Go counterparts; developers might save both performance and memory by translating the Python modules into other (compiled) languages. Additionally, more tests are needed: The server has not been tested on particularly large datasets; nor has it been tested for parallel queries.

To allow for cases where WebDP is used by clients that already have user handling as part of their IT management services, another field of future development would be for WebDP to allow users to identify and authenticate themselves using their existing user handling system, as an alternative to the internal user handling and authentication.

5

Conclusion

Initially, the goal was to connect WebDP to OpenDP. After an evaluation of the existing implementation, a rework was deemed necessary. This re-implementation of the proof-of-concept WebDP server focused on a modular design to allow each engine freedom from WebDP, so that every new engine – or updates to existing ones – require minimal changes to WebDP itself. The re-implementation functions as a proof-of-concept for the new design as well, proving that it does in fact serve the desired level of extensibility. This is further proven by the fact that the connectors are written in different languages.

Using a microservice architecture pattern proved useful in allowing the WebDP API to extend to new engines. After the implementation of the first and second engine connectors, and the tweaks and modifications in the API, a template on how new connectors should be added was created: The internal API specification. Using this specification, the implementation of the third connector was quicker as it simply needed to connect to the internal endpoints to reach full support. Thus, WebDP server does not have to be modified if future connectors are added or updated.

Since this service could potentially generate a lot of traffic, it needs the ability to handle the amount of concurrent users and connections – and scale accordingly. Using containers to run the engines gives the benefits of letting it run on any machine, starting more containers to handle increased workload, and keeping the instances separated. A drawback is a noticeably larger file size for certain engines, which can impact how many instances can be run simultaneously.

There is a compelling case for refining the *MeanMeasurement* step to better integrate with the OpenDP library. The library's mean function offers greater accuracy compared to post-processing methods and includes the valuable feature of pre-evaluation accuracy – a feature that is absent in both Tumult and GoogleDP. Additionally, there is a strong rationale for incorporating an optional "bounds" field in steps where clamping is expected to be used as a noise-bounding technique. This would enable analysts to achieve lower noise levels under specific circumstances. Another important case is that of the implementation of histograms in the OpenDP connector. It shows that even the most basic subset of WebDP's query step does not translate well to OpenDP's core library. A framework around the core library could potentially solve that issue.

The lack of unity and standardisation between the different DP libraries required some preparatory work in regards to digesting each library's documentation and interpreting the differences in the implementations in order to achieve the desired, modular API. Each connector individually translates DP operations from a query call before the respective external libraries are called. This helps the user create a single query that works for several engines, instead of individually handcrafting them for each. With this unifying work, our hope is that in the future, differential privacy becomes more unified and accessible.

References

[1] D. J. Solove, "'I've Got Nothing to Hide' and Other Misunderstandings of Privacy," *San Diego Law Review*, vol. 44, p. 745, 2007, GWU Law School Public Law Research Paper No. 289. [Online]. Available: https://ssrn.com/abstract=998565.

[2] P. Ohm, "Broken promises of privacy: Responding to the surprising failure of anonymization," *UCLA Law Review*, vol. 57, no. 6, pp. 1701–1777, 2010.

[3] L. Sweeney, "Simple demographics often identify people uniquely," Carnegie Mellon University, Working Paper, 2000, Project website: https://dataprivacylab.org/projects/identifiability/.

[4] C. Dwork, "Differential privacy: A survey of results," in *Theory and Applications of Models of Computation*, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19, ISBN: 978-3-540-79228-4.

[5] A. Wood *et al.*, "Differential Privacy: A Primer for a Non-Technical Audience," *Vanderbilt Journal of Entertainment & Technology Law*, vol. 21, no. 17, pp. 209–276, 2018, Berkman Klein Center Research Publication No. 2019-2. DOI: 10.2139/ssrn.3338027.

[6] The OpenDP Team, "The OpenDP White Paper," 2020. [Online]. Available: https://privacytools.seas.harvard.edu/publications/opendp-white-paper.

[7] Population Reference Bureau and the U.S. Census Bureau's 2020 Census Data Products and Dissemination Team, "Why the census bureau chose differential privacy," U.S. Census Bureau, Tech. Rep., 2023. [Online]. Available: https://www.census.gov/library/publications/2023/decennial/c2020br-03.html.

[8] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 1054–1067, ISBN: 9781450329576. DOI: 10.1145/2660267.2660348.

[9] Differential Privacy Team, Apple, "Learning with privacy at scale," Tech. Rep., 2017. [Online]. Available: https://machinelearning.apple.com/research/learning-with-privacy-at-scale.

[10] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*,

ser. CCS'16, ACM, Oct. 2016. DOI: 10.1145/2976749.2978318. [Online]. Available: http://dx.doi.org/10.1145/2976749.2978318.

[11] A. Blanco-Justicia, D. Sánchez, J. Domingo-Ferrer, and K. Muralidhar, "A critical review on the use (and misuse) of differential privacy in machine learning," *ACM Computing Surveys*, vol. 55, no. 8, Dec. 2022, ISSN: 0360-0300. DOI: 10.1145/3547139.

[12] DPella. "WebDP." Accessed: 2024-02-02. (2023), [Online]. Available: https://github.com/dpella/WebDP.

[13] DPella. "DPella | Privacy Preserving Analytics | Home." Accessed: 2024-02-02. (2024), [Online]. Available: https://www.dpella.io.

[14] Tumult Analytics. "Tumult Analytics." Accessed: 2024-02-02. (2024), [Online]. Available: https://www.tmlt.dev/.

[15] A. Russo, Project supervision meetings, 2024.

[16] OpenDP. "Developing Open Source Tools for Differential Privacy." Accessed: 2024-02-02. (2024), [Online]. Available: https://opendp.org/.

[17] B. Jin, S. Sahni, and A. Shevat, *Designing Web APIs: Building APIs That Developers Love*. O'Reilly Media, 2018, ISBN: 9781492026877.

[18] D. Crockford and C. Morningstar, "404: The JSON data interchange syntax," *ECMA (European Association for Standardizing Information and Communication Systems)*, Dec. 2017. DOI: 10.13140/RG.2.2.28181.14560.

[19] W. Stallings and L. Brown, *Computer Security: Principles and practice, Global Edition*. Pearson Higher Education, 2017, ISBN: 9781292220611.

[20] OpenAPI Initiative. "What is OpenAPI?" Accessed: 2024-04-23. (Dec. 2023), [Online]. Available: https://www.openapis.org/what-is-openapi.

[21] C. Dwork, "Differential privacy," in *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ser. ICALP'06, Venice, Italy: Springer-Verlag, 2006, pp. 1–12, ISBN: 3540359079. DOI: 10.1007/11787006_1.

[22] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014. DOI: 10.1561/0400000042.

[23] A. Beimel, K. Nissim, and U. Stemmer, "Private learning and sanitization: Pure vs. approximate differential privacy," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, P. Raghavendra, S. Raskhodnikova, K. Jansen, and J. D. P. Rolim, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 363–378, ISBN: 978-3-642-40328-6.

[24] Ninghui Li, Ming Lyu, Dong Su, Weining Yang, *Differential Privacy: From Theory to Practice*. Springer, 2016. DOI: 10.1007/978-3-031-02350-7.

[25] 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*, ser. PODS '03, San Diego, California: Association for Computing Machinery, 2003, pp. 202–210, ISBN: 1581136706. DOI: 10.1145/773153.773173.

[26] OpenDP. "Accuracy: Pitfalls and Edge Cases." Accessed: 2024-05-03. (2024), [Online]. Available: https://docs.opendp.org/en/stable/user/utilities/accuracy/accuracy-pitfalls.html.

[27] E. Lobo-Vesga, A. Russo, and M. Gaboardi, "A Programming Language for Data Privacy with Accuracy Estimations," *ACM Transactions on Programming Languages and Systems*, vol. 43, no. 2, 2021. DOI: 10.1145/3452096.

[28] OpenDP. "OpenDP documentation." Version 0.9.2. Accessed: 2024-04-15. (2024), [Online]. Available: https://docs.opendp.org/en/v0.9.2/opendp-commons/.

[29] S. Berghel *et al.*, "Tumult Analytics: A robust, easy-to-use, scalable, and expressive framework for differential privacy," Dec. 2022. arXiv: 2212.04133 [cs.CR].

[30] Google. "Differential Privacy." Accessed: 2024-04-17. (2024), [Online]. Available: https://github.com/google/differential-privacy.

[31] S. Casacuberta, M. Shoemate, S. Vadhan, and C. Wagaman, "Widespread underestimation of sensitivity in differentially private libraries and how to fix it," 2022. arXiv: 2207.10635 [cs.CR].

[32] Meta Platforms, Inc. "Opacus." Accessed: 2024-04-23. (2024), [Online]. Available: https://opacus.ai.

[33] Secretflow. "Secretflow: A unified framework for privacy-preserving data analysis and machine learning." Accessed: 2024-04-23. (2024), [Online]. Available: https://github.com/secretflow/secretflow.

[34] IBM. "Diffprivlib: The IBM Differential Privacy Library." Accessed: 2024-04-23. (2024), [Online]. Available: https://github.com/IBM/differential-privacy-library.

[35] WebDP, Internal documents, 2024.

[36] OpenAPI Generator. "OpenAPI Generator." Accessed: 2024-04-07. (2024), [Online]. Available: https://openapi-generator.tech/.

[37] Docker Inc. "Docker: Accelerated Container Application Development." Accessed: 2024-05-29. (2024), [Online]. Available: https://docker.com/.

[38] NumFocus, Inc. "Pandas - Python Data Analysis Library." Accessed: 2024-04-23. (2024), [Online]. Available: https://pandas.pydata.org/.

[39] OpenDP. "Aggregation: Mean." Version 0.9.2. Accessed: 2024-04-17. (2024), [Online]. Available: https://docs.opendp.org/en/v0.9.2/user/transformations/aggregation-mean.html.

[40] "Working with Unknown Dataset Sizes." Accessed: 2024-05-02. (2024), [Online]. Available: https://docs.opendp.org/en/stable/examples/unknown-dataset-size.html.

[41] OpenDP. "Aggregation: Sum." Accessed: 2024-05-07. (2024), [Online]. Available: https://docs.opendp.org/en/stable/user/transformations/aggregation-sum.html.

[42] PyPI. "Connexion - API first applications with OpenAPI/Swagger." Accessed: 2024-04-01. (2024), [Online]. Available: https://pypi.org/project/connexion/.

[43] Go-OpenAPI. "OpenAPI Initiative golang toolkit." Accessed: 2024-04-01. (2024), [Online]. Available: https://github.com/go-openapi.

[44] Go-Swagger. "Dynamic API." Accessed: 2024-04-01. (Jan. 2024), [Online]. Available: https://goswagger.io/go-swagger/tutorial/dynamic/.

[45] Go-OpenAPI. "OpenAPI v2 object model: Readme." Accessed: 2024-04-05. (2024), [Online]. Available: https://github.com/go-openapi/spec.

[46] Go-OpenAPI. "OAI object model: Readme." Accessed: 2024-04-05. (2024), [Online]. Available: https://github.com/go-openapi/spec3.

[47] Integritetsskyddsmyndigheten (IMY). "De registrerades rättigheter." In Swedish. Accessed: 2024-02-07. (2022), [Online]. Available: https://www.imy.se/verksamhet/dataskydd/det-har-galler-enligt-gdpr/de-registrerades-rattigheter/.

[48] A. Cohen and K. Nissim, "Towards formalizing the GDPR's notion of singling out," *Proceedings of the National Academy of Sciences*, vol. 117, no. 15, pp. 8344–8352, 2020. DOI: 10.1073/pnas.1914598117.

[49] R. U. Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, and J. Qadir, "Security and privacy of internet of medical things: A contemporary review in the age of surveillance, botnets, and adversarial ML," *Journal of Network and Computer Applications*, vol. 201, p. 103 332, 2022, ISSN: 1084-8045. DOI: 10.1016/j.jnca.2022.103332.

[50] Integritetsskyddsmyndigheten (IMY). "Anmälda personuppgiftsincidenter 2022." In Swedish. Accessed: 2024-02-07. (2023), [Online]. Available: https://www.imy.se/publikationer/anmalda-personuppgiftsincidenter-2022/.

[51] G. J. Myers, T. Badgett, and C. Sandler, *The art of software testing*. Wiley, 2012, ISBN: 1118133137. DOI: 10.1002/9781119202486.

A

Appendix: Engine Comparison

Comparision of the implemented engines, compared to the WebDP specification

In WebDP specification			
Transformations	Tumult	OpenDP	GoogleDP
Select	✓	✓	
Filter	✓	✓	✓
Rename	✓		
Map	✓		
Bin	✓	✓	✓
Measurements	Tumult	OpenDP	GoogleDP
Count	✓	✓	✓
Min	✓		
Max	✓		
Mean	✓		✓
Sum	✓	✓	✓
GroupBy	✓		

Not in WebDP specification			
Measurements	Tumult	OpenDP	GoogleDP
Variance			✓
Stdev			✓
Quantiles			✓

B

Appendix: ER diagram

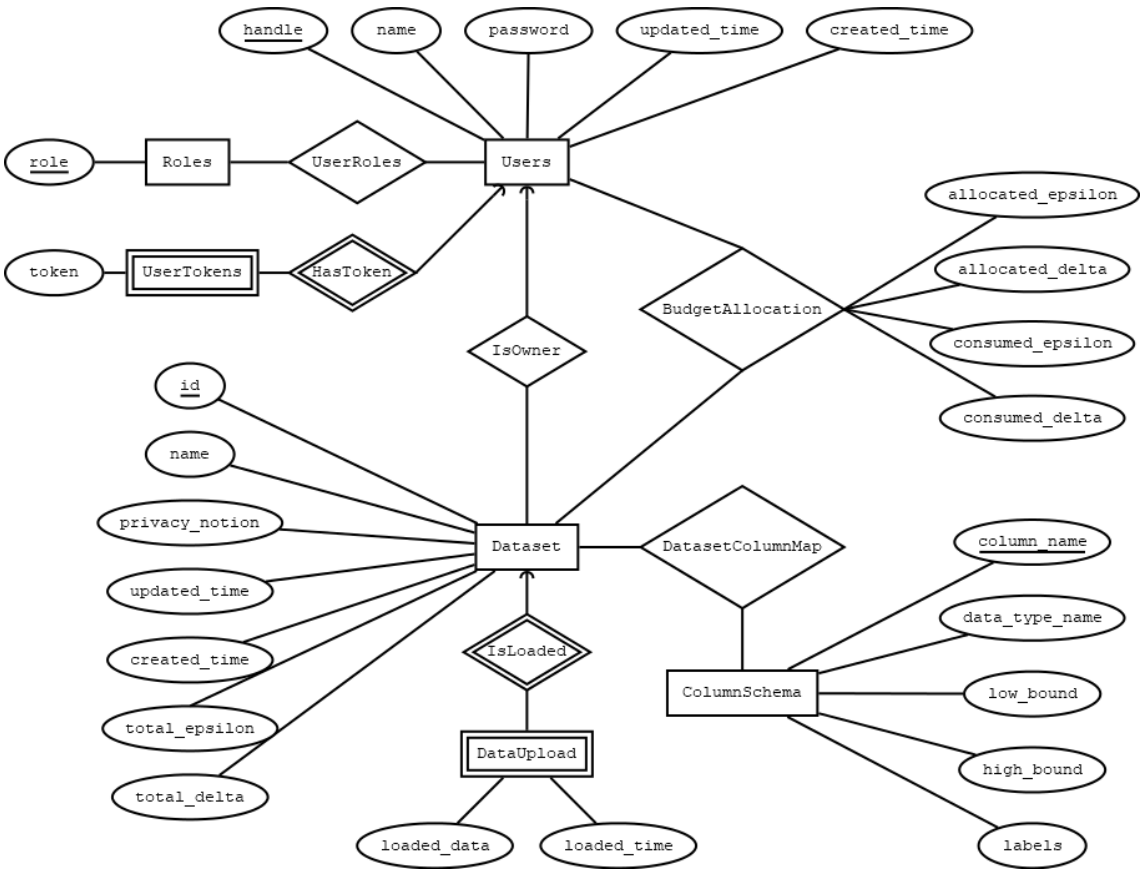

Figure B.1: The relational schema of the WebDP implementation.

C

Appendix: Proof of Theorem 3

To prove Theorem 3, we will make use of two lemmas.

Lemma 1. *Let x, y exist on the closed interval $[a, b] \subset \mathbb{R}$. Then we have that:*

$$|y - x| \leq \max\left(|x - a|, |x - b|\right) \tag{C.1}$$

Proof. Assume that $y \leq x$. Then we have that $y \in [a, x]$, and therefore that

$$|y - a| + |y - x| = |x - a|$$

This means that $|y - x| \leq |x - a|$, since all terms are non-negative. By the definition of $\max(\cdot)$, we have that:

$$\begin{aligned} |y - x| &\leq |x - a| \\ &\leq \max\left(|x - a|, |x - b|\right) \end{aligned}$$

The case when $y \geq x$ follows by the symmetry of distances between numbers, which finalises the proof. □

Lemma 2. *Let $\tilde{x}, \tilde{y} \in \mathbb{R}$ be the results of two differentially private queries, where $x, y \in \mathbb{R}$ denote their true values such that the pre-evauation accuracy estimates*

$$Pr\left[|\tilde{x} - x| \leq a_x\right] \geq \beta_x$$

$$Pr\left[|\tilde{y} - y| \leq a_y\right] \geq \beta_y$$

and it is given that $0 < \tilde{x} - a_x$. Then we have that

$$Pr\left[\frac{\tilde{y} - a_y}{\tilde{x} + a_x} \leq \frac{y}{x} \leq \frac{\tilde{y} + a_y}{\tilde{x} - a_x}\right] \geq \beta_x \cdot \beta_y$$

Proof. By the definition of pre-evaluation accuracy, we know that

$$Pr\left[\tilde{x} - a_x \leq x \leq \tilde{x} + a_x\right] \geq \beta_x \tag{C.2}$$

$$Pr\left[\tilde{y} - a_y \leq y \leq \tilde{y} + a_y\right] \geq \beta_y \tag{C.3}$$

Since they are probabilities of independent events, their joint probability is at least the product of their respective probabilities, $\beta_x \cdot \beta_y$. Assuming that $\tilde{x}$ and $\tilde{y}$ lie within their probabilistic error-bounds, then we may write:

$$\begin{aligned} \frac{\tilde{y} - a_y}{\tilde{x} + a_x} &\leq \frac{y}{\tilde{x} + a_x} && \textit{(By C.3)} \\ &\leq \frac{y}{x} && \textit{(By C.2)} \end{aligned}$$

and that

$$\begin{aligned} \frac{y}{x} &\leq \frac{y}{\tilde{x} - a_x} && \textit{(By C.2 and } 0 < \tilde{x} - a_x\textit{)} \\ &\leq \frac{\tilde{y} + a_y}{\tilde{x} - a_x} && \textit{(By C.3)} \end{aligned}$$

Since $\tilde{x}$ and $\tilde{y}$ lie within their probabilistic error-bounds at a joint probability of at least $\beta_x \cdot \beta_y$, the inequalities above are accurate at a probability of at least $\beta_x \cdot \beta_y$, which is what we wanted to prove. □

Proof of ***Theorem 3****.* Recall that Theorem 3 states that:

$$Pr\left[|m - \tilde{m}| \leq \frac{|\tilde{x}a_y + \tilde{y}a_x|}{\tilde{x}\left(\tilde{x} - a_x\right)}\right] \geq \beta_x \cdot \beta_y$$

where the preconditions are the same as in Lemma 2, and $m = \frac{y}{x}$, $\tilde{m} = \frac{\tilde{y}}{\tilde{x}}$

By Lemma 2, we know that at a probability of at least $\beta_x \cdot \beta_y$

$$m, \tilde{m} \in \left[\frac{\tilde{y} - a_y}{\tilde{x} + a_x}, \frac{\tilde{y} + a_y}{\tilde{x} - a_x}\right] \tag{C.4}$$

By (C.4) and Lemma 1 we have that

$$|m - \tilde{m}| \leq \max\left(\left|\tilde{m} - \frac{\tilde{y} - a_y}{\tilde{x} + a_x}\right|, \left|\tilde{m} - \frac{\tilde{y} + a_y}{\tilde{x} - a_x}\right|\right) \tag{C.5}$$

An expansion of the left operand in the RHS of (C.5) gives

$$\begin{aligned}\left|\tilde{m} - \frac{\tilde{y} - a_y}{\tilde{x} + a_x}\right| &= \left|\frac{\tilde{y}(\tilde{x} + a_x) - \tilde{x}(\tilde{y} - a_y)}{\tilde{x}(\tilde{x} + a_x)}\right| \\ &= \left|\frac{\tilde{x}a_y + \tilde{y}a_x}{\tilde{x}(\tilde{x} + a_x)}\right| \\ &= \frac{|\tilde{x}a_y + \tilde{y}a_x|}{\tilde{x}(\tilde{x} + a_x)} \qquad (By\ \tilde{x}(\tilde{x} + a_x) > 0)\end{aligned}$$

A similar expansion of the right operand gives

$$\left|\tilde{m} - \frac{\tilde{y} + a_y}{\tilde{x} - a_x}\right| = \frac{|\tilde{x}a_y + \tilde{y}a_x|}{\tilde{x}(\tilde{x} - a_x)}$$

Since $\tilde{x}(\tilde{x} + a_x) \geq \tilde{x}(\tilde{x} - a_x)$, it must be that case that the left operand is never greater than the right, which gives that

$$|m - \tilde{m}| \leq \frac{|\tilde{x}a_y + \tilde{y}a_x|}{\tilde{x}\,(\tilde{x} - a_x)}$$

is true at a confidence of at least $\beta_x \cdot \beta_y$, which is what we wanted to prove.

□

D

Appendix: Using Chernoff bounds for error-estimation

The use of Chernoff bounds is convenient to specify upper bounds on the probability that a random variable is larger or smaller than some specified quantity, a:

$$\begin{aligned} Pr\left[X \geq a\right] &\leq M_X(t)e^{-ta} && (t > 0) \\ Pr\left[X \leq a\right] &\leq M_X(t)e^{-ta} && (t < 0) \end{aligned}$$

Where M_X is the moment generating function (MGF) of the random variable X. Finding a t such that the RHS is minimized will produce the tightest Chernoff bound.

Error of a sum of Laplace random variables

If $X \sim Lap(0, b)$ then the MGF of X is:

$$M_X(t) = (1 - b^2t^2)^{-1}$$

If we let $Y = \sum_{i=1}^{n} X_i$ (X_i is independent) then the MGF of Y is:

$$\begin{aligned} M_Y &= \mathbf{E}\left[e^{t\sum_{i=1}^{n} X_i}\right] \\ &= \prod_{i=1}^{n} \mathbf{E}\left[e^{tX_i}\right] \\ &= \prod_{i=1}^{n} M_{X_i}(t) \\ &= (1 - b^2t^2)^{-n} \end{aligned}$$

Thus, by the Chernoff bound, we have that:

$$Pr\left[Y \geq a\right] \leq e^{-ta}(1-b^2t^2)^{-n}$$

The RHS has a minimum at:

$$t_1 = -\frac{n}{a} + \sqrt{\frac{n^2}{a^2} + b^{-2}} \tag{D.1}$$

We have that $t_1 > 0$ in (D.1) when $a > 0$ and $b > 0$. We know that $n \geq 1$. Note that when $a < 0$, we have a solution $t_2 = -t_1 < 0$ since:

$$e^{-a_2t_2}(1-b^2t_2^2)^{-n} = e^{-(-a_1)(-t_1)}(1-b^2(-t_1)^2)^{-n} \tag{D.2}$$

$$= e^{-a_1t_1}(1-b^2t_1^2)^{-n} \tag{D.3}$$

This also means that $Pr\left[Y \geq a\right] \leq P_1$ and $Pr\left[Y \leq -a\right] \leq P_1$, and thus we have that $Pr\left[|Y| \geq a\right] \leq 2P_1$. In the context of differential privacy, the scale parameter b in the Laplace mechanism is set to $\frac{\Delta_f}{\varepsilon}$. Thus, if we sum the result of a DP histogram where each bin has been counted and perturbed with the Laplace mechanism with a budget of ε, we get that:

$$Pr\left[|Y| \geq a\right] \leq 2e^{-a\left(-\frac{n}{a}+\sqrt{\frac{n^2}{a^2}+\left(\frac{\Delta_f}{\varepsilon}\right)^2}\right)}\left[1-\left(\frac{\Delta_f}{\varepsilon}\right)^2\left(\frac{-n}{a}+\sqrt{\frac{n^2}{a^2}+\left(\frac{\Delta_f}{\varepsilon}\right)^2}\right)\right]^{-n} \tag{D.4}$$

Where Y is the noise from the mechanism. Setting $a = 540$, $\varepsilon = \Delta_f = 1$ and $n = 15,000$ will yield results supporting the claims made in the discussion.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden
www.chalmers.se | www.gu.se

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY