Large Scale Federated Analytics and Differential Privacy Budget Preservation

by

Matheus Ulhoa Avelar Stolet

BA. Computer Science, The University of British Columbia, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia
(Vancouver)

August 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Large Scale Federated Analytics and Differential Privacy Budget Preservation

submitted by **Matheus Ulhoa Avelar Stolet** in partial fulfillment of the requirements for the degree of **Master of Science**
in **Computer Science**.

Examining Committee:

Ivan Beschastnikh, Computer Science, UBC
Co-Supervisor

Aline Talhouk, Medicine, UBC
Co-Supervisor

Abstract

This thesis presents two contributions. The first contribution deals with the problem of siloed data collection and prohibitive data acquisition costs. These costs limit the size and diversity of datasets used in health research. Access to larger and more diverse datasets improves the understanding of disease heterogeneity and facilitates inference of relationships between surgical and pathological findings with symptomatic indicators and outcomes. Unfortunately, freely enabling access to these datasets has the potential of leaking private information, such as medical records, even when these datasets have been stripped of personally identifiable information.

In the first part of this thesis, we present LEAP, a data analytics platform with support for federated learning. LEAP allows users to analyze data distributed across multiple institutions in a private and secure manner, without leaking sensitive patient information. LEAP achieves this through an infrastructure that maintains privacy by design and brings the computation to the data, instead of bringing the data to the computation. LEAP adds an overhead of up to 2.5X, training Resnet-18 with 15 participating sites, when compared to a centralized model. Despite this overhead, LEAP achieves convergence of the model's accuracy within 20% of the time taken for the centralized model to converge.

One of the techniques used by LEAP to preserve the privacy of sensitive queries is differential privacy. Successive DP queries to a dataset depletes the privacy budget. When the privacy budget is depleted, data curators must block access to the underlying dataset to prevent private information from leaking. In the second part of this thesis, we present a system called the SmartCache. The SmartCache optimizes the use of the privacy budget by interpolating old query results to help answer new queries using a synthetic dataset. Queries answered from the synthetic dataset have a smaller privacy cost, so more queries can be answered before the budget runs out. For statistical queries, the SmartCache saved 30%-50% of the budget for threshold values of 0.99 and 0.999, and for gradient queries it consumed 70% less of the privacy budget when training a fully connected model.

Lay Summary

Collecting a large enough dataset that covers all forms of disease, including rare variants is of utmost importance, but cannot be provided by a single health center. More and better data leads to better medical research, which leads to better medicine and more lives saved. Furthermore, medical data is often sensitive and existing techniques used to query sensitive data restrict the number of queries that can be made.

We present two systems, LEAP and SmartCache, that deal with the challenges above. LEAP allows researchers to query data distributed across multiple institutions, thus making more available to users, and the SmartCache solves the problem of querying sensitive data by allowing users to execute more queries on sensitive datasets without a privacy risk.

Preface

The work presented here was conducted in the Systopia Lab at the University of British Columbia. This thesis is an original, unpublished work by Matheus Stolet, written under the supervision of Ivan Beschastnikh and Aline Talhouk, with the guidance and collaboration of Mathias Lécuyer.

In addition to my work, several individuals helped in the implementation of various features. Chris Yoon helped me with the initial LEAP prototype, Aditya Chinchure implemented the REDCap connector, and Kalli Leung assisted in the data generation for the SmartCache and in searching for systems similar to LEAP.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I want to thank Dr. Ivan Beschastnikh and Dr. Aline Talhouk for the helpful guidance in the past years. The feedback and support I have received from them is invaluable and has helped me become a better researcher. I also want to thank Dr. Mathias Lécuyer for going above and beyond his duty and guiding me with the SmartCache work. The support of these three was paramount for the completion of this thesis.

I am grateful for all the NSS/Systopia members and friends which I met during my six years at UBC. I appreciate the technical help, knowledge nuggets, and overall good times we had together.

In addition to the names mentioned above, I also want to thank my parents and family for their uncoditional love and support throughtout this adventure.

This work was supported by NSERC USRA and Discovery grants and by the Cascadia Data Alliance.

Chapter 1

Introduction

Artificial intelligence (AI) and Machine Learning (ML) as well as improvement in networking speeds, computational ability, and cloud computing present unprecedented opportunities to glean knowledge from health data, to improve patient care, clinical outcomes, and use health care resources more efficiently. However, to date, many of the benefits of AI remain unrealized in the health domain. This is because AI and ML require massive amounts of data to be effective. Even as vast amounts of health data are increasingly generated from traditional health care encounters, medical and consumer devices, wearables, and patient-reported outcomes, these data remain siloed due to privacy concerns.

A large enough dataset that covers all forms of the disease, including rare variants is of utmost importance, but cannot be provided by a single center. Traditionally, de-identified data from multiple institutions was centralized for the purpose of analysis. With patient records becoming richer and high-dimensional, moving that data to a central location causes both logistical and privacy concerns. Moreover, data transfer agreements between health institutions take a long time, and despite several efforts, access and transfer remains a barrier [39].

Federated Learning (FL) is a technical solution that does not require data to be centralized for the development of AI/ML models. This can help both with scalability and privacy. FL can run a variety of ML tasks over a network; the data stays where it was originally collected and never transferred. One of the benefits of federation is that it offers continuous control over the data access.

Federated data analysis does not guarantee privacy. For example, FL is susceptible to a variety of attacks such as model inversion [40] [21], membership inference [31] [36], and record re-identification [19] [30]. These attacks can be carried out by analyzing results from queries or predictions of a model.

Differential privacy [13] [15] (DP) is a framework that provides privacy. DP provides formal privact guarantees by ensuring that the removal or addition of a database record affects the outcome of a result by only a small amount. Thus, it prevents the identification of individuals and gives mathematical guarantees on the risk of being identified in the database. DP is achieved by adding random noise from a distribution to the result of a query, so that the real value is obfuscated but the returned result still retains statistical significance. DP enables quantification of privacy loss of a system through a parameter ε. For example, developers can enforce that a system will not have a privacy loss worse than ε by measuring the privacy loss of individual queries and blocking further queries from being answered when a limit is reached. Consequently, there has been much research into systems that use DP as efficiently as possible because of the constraints imposed by the accumulated privacy loss [8] [24] [41] [34] [35].

DP queries are composable, meaning that Successive queries to a dataset increases the privacy loss. Composition [18] [22] [12] [17] is one of the fundamental properties of differential privacy. With each query, privacy degrades, but it degrades in a controlled fashion, allowing the total privacy loss of a system to be measured. The privacy budget determines the max privacy loss acceptable by a system, so a data curator or system administrator can restrict queries to the system when all the privacy budget has been spent. In other words, the budget sets the upper bound on the privacy loss. For instance, a data curator can specify that the privacy budget of a dataset is $\varepsilon = 3$. If a user issues three queries, where each query has a privacy loss of $\varepsilon = 1$, the budget will be depleted and proceeding to query the dataset further will result in a privacy risk for individuals in the dataset. This imposes a query limitation on systems using differential privacy. These restrictions make differential privacy unpalatable to those who want to extract the most out of their data or to those who are operating in scenarios with large query volumes. Tackling this issue is important to increase the adoption of privacy systems using differential privacy.

Many techniques and relaxations of DP have been developed to reduce the privacy loss of individual queries [27] [29] [24]. However, there is not much work on preserving the budget by re-using information and insights gathered from past queries. A few theorethical results exist, but they are not practical enough to be deployed [20] [10].

My contributions in this thesis are two-fold. One is the development of a general purpose data analytics system that can be used to query federated health data across sites. Privacy is a concern for healthcare systems because they deal with sensitive data. A solution to these privacy concerns is to issue queries with differential privacy guarantees. The problem is that differential privacy restricts the number of queries a user can perform. Therefore, my second contribution is the development of a technique that maximizes the amount of queries answered by a system before running out of the privacy budget.

In the first part of this thesis I present my first contribution, a Large scale federated and privacy preserving Evaluation and Analysis Platform (LEAP) that allows users to analyze data distributed across multiple institutions in a private and secure manner, without leaking sensitive patient information. LEAP achieves this through an infrastructure that maintains privacy by design and follows a federated philosophy of bringing the computation to the data, instead of bringing the data to the computation.

The LEAP infrastructure is divided into two groups, the cloud infrastructure and the site infrastructure. Queries are distributed to different sites by the cloud infrastructure and a LEAP module running on each site sends back the result after running local computation on the local data. Optional noise can be added to results to provide differential privacy (DP) guarantees, when requested. These results are returned from multiple sites to the LEAP infrastructure in the cloud, where they are aggregated and returned to the end-user. The infrastructure also keeps track of the privacy loss from each query, safeguarding against information leakage from repeated queries. Our evaluations have shown that using LEAP incurs an overhead when compared to training a model or performing queries on a centralized dataset. Nonetheless, there are benefits to LEAP such as control over data (e.g. revoking access), simplified data agreements, and time saved in network IO from centralizing the data. For example, our evaluations have demonstrated that as more sites join LEAP (and thus more data is available) the convergence time for the validation accuracy decreases.

In the second part of this thesis I present a system called SmartCache. SmartCache saves DP budget across queries by learning from each query and interpolating old results to answer new queries without consuming much of the DP budget. SmartCache generates a DP synthetic dataset based on the real dataset, and answers incoming queries using both real and synthetic dataset. If the difference between results is smaller than an acceptable threshold, results are returned from the synthetic dataset at a small privacy cost (cache hit). If the difference between results is above the threshold, results from the real dataset are returned at a larger privacy cost (cache miss). Our cache initializes a vector of weights w, where each element $w_i \in w$ is associated to a row in the synthetic dataset. With each miss, the cache is improved by updating weights of the synthetic dataset to minimize differences between results from the real and synthetic datasets. Our evaluations have shown that the SmartCache is able to maintain the quality of queries answered, and save the DP budget when answering gradient queries. For statistical queries the results were not as positive and we were not able to use less of the privacy budget than the real dataset with DP.

Given that federated learning and differential privacy are important for understanding LEAP and the SmartCache, the next chapter will cover background material on both. For the federated learning section, the background chapter covers how a machine learning model is trained using federated learning. In the following section, I will review differential privacy and its basic properties.

Chapter 2

Background

This section covers some background material necessary to understand LEAP and the SmartCache. The following sections define and illustrate federated learning and differential privacy. LEAP performs computation in a federated setting where data never leaves the site holding the data. Below I go over how a machine learning algorithm can be trained in these situations. Differential privacy is important for both LEAP and the SmartCache. It gives a formal definition of privacy that lets us quantify how much privacy is lost with each query. In Section 2.2 I formally define differential privacy and some popular relaxations of the original definition.

2.1 Federated Learning

Federated learning [25] is a method for training machine learning models from decentralized data. In a traditional scenario, data is centralized on the machine where the learning takes places. Federated learning breaks this paradigm and enables different devices to contribute to training without ever exchanging the actual data. As shown in Figure 2.1, a ML model can be trained using federated learning by having a central coordinator push an initial model to each participating device. Each device then computes an SGD update using the data available locally to them. Each device then sends the gradients from the SGD update back to the central coordinator. The coordinator aggregates the gradients from each device by summing them together and then averaging the result. The averaged gradients are then applied to update the model weights, thus completing one SGD step. The updated model is then pushed to each device and the steps are repeated until the model reaches convergence.

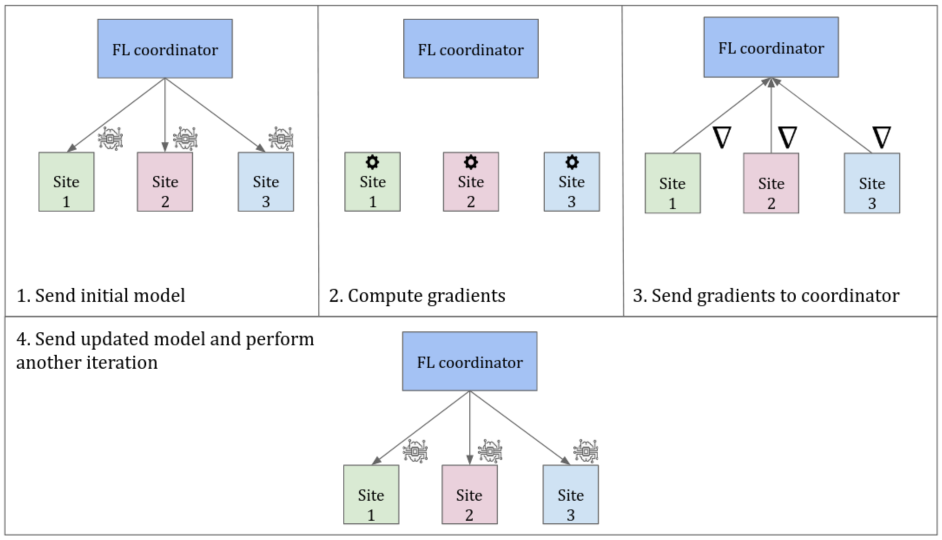
FL coordinator
Site 1
Site 2
Site 3
1. Send initial model
2. Compute gradients
3. Send gradients to coordinator
∇
4. Send updated model and perform another iteration

Figure 2.1: 1. Centralized coordinator sends model to each site. 2. Sites compute an update on the local data. 3. Sites send gradients back to centralized coordinator. 4. Updated model is sent back to each site and the process is repeated for *n* iterations.

2.2 Differential Privacy

Differential privacy [13] [15] is a framework for privacy that provides strong formal guarantees. Differential privacy enables the use of datasets in any analysis without adversely affecting data of individuals included in the dataset. For example, when differential privacy is upheld, with a certain probability the same conclusions of a study will be derived, regardless of whether an individual is part of a dataset or not: the probability of any sequence of responses to queries on the dataset is essentially the same, independent of the presence or absence of an individual. Presented formally, a randomized mechanism M satisfies ε-differential privacy (also known as pure differential privacy), for any adjacent datasets [1] A, B and any set of possible outputs $U \subseteq Range(M)$ if

$$\Pr[M(A) \in U] \leq e^{\varepsilon} \Pr[M(B) \in U].$$

As this definition shows, the probability of the output of a mechanism M when an individual is added or removed from a dataset is bounded by e^{ε}. This translates to the notion that a smaller value of ε provides stronger privacy guarantees and a larger value has weaker guarantees. A bigger ε increases the bound on how much the probability of the output of a mechanism may change when an individual is added or removed from a dataset, potentially making it easier to draw conclusions about an individual, since one person has a larger influence on the output.

There are different mechanisms that produce DP results, but the most common mechanism for pure DP is the laplace mechanism, which is defined as

$$M_L(f(x)) = f(x) + \text{Lap}(0, \frac{\nabla_1 f}{\varepsilon})$$

where f is a non private query and $\nabla_1 f$ is the max $\ell 1$-sensitivity of query f taken over adjacent datasets A and B

$$\nabla_1 f = \max_{A,B} ||f(A) - f(B)||_1.$$

Differential privacy has two properties that are of interest to our work: robustness to post-processing and composability. The post-processing theorem of differential privacy states that if the output of a mechanism M satisfies differential privacy, so does $g(M(A))$, where g is an mapping from the set of possible outputs of M to an arbitrary set. This allows a differentially private result, or a dataset in our case, to be released and modified without losing DP guarantees. Furthermore, even if a query is DP, privacy degrades with each query. One of the nice properties of differential privacy is that it degrades in a controlled fashion and the composability theorem allows us to quantify the loss after n queries. For example, suppose we have a mechanism that is ε-DP. If we query this mechanism n times, the result is $n\varepsilon$-DP.

[1] Adjacent datasets are datasets that differ by at most one row

2.2.1 (ε, δ)-DP

In this thesis we deal with a relaxation of ε-DP (pure differential privacy) called (ε, δ)-DP [14]. A mechanism M is (ε, δ)-DP if for any adjacent datasets A and B and any set of possible outputs $U \subseteq Range(M)$

$$\Pr[M(A) \in U] \leq e^{\varepsilon} \Pr[M(B) \in U] + \delta.$$

This relaxation means that the mechanism is ε-DP with probability $1 - \delta$. One of the reasons that we opt for this relaxation is because it allows us to use the Gaussian mechanism instead of the Laplace mechanism, since the Gaussian mechanism does not satisfy ε-DP but it does satisfy (ε, δ)-DP guarantees. The Gaussian mechanism is defined as:

$$M_G(f(x)) = f(x) + \text{Normal}(0, \sigma^2)$$

where σ is equal to $\frac{\nabla_2 f}{\varepsilon}\sqrt{2\ln(\frac{1.25}{\delta})}$ and the $\ell 2$ sensitivity of function f is

$$\nabla_2 f = \max_{A,B} ||f(A) - f(B)||_2.$$

The Gaussian mechanism is preferrable in our case because the tails of the Gaussian distribution decay faster than the tails of the Laplace distribution. This means that when adding random noise there is a smaller chance of the noise coming from a large number on the tails of the distribution. The Gaussian distribution also has convenient properties and is familiar to most data analysts and scientists, which makes it convenient when releasing DP results. Furthermore, it allows us to use advanced composition theorems [22] so that for small values of ε and a large number of queries, we have a tighter bound on the privacy loss. This means that more queries can be issued with a smaller privacy cost.

Differential privacy is used by both LEAP and the SmartCache to give formal privacy guarantees for sensitive queries, while federated learning is the basis for the LEAP architecture. Given the background material on both these approaches, the next chapter will cover the design of LEAP, how the design in LEAP allows for federated queries, and how differential privacy is used for sensitive queries.

Chapter 3

LEAP Design

LEAP is a general purpose federated data analytics platform that enables users to query data distributed among multiple sites. Participating sites register with the LEAP system and users send query requests that are distributed to each site. LEAP aggregates results from each site and sends the aggregated result back to the user. Three main design goals influenced the development of LEAP: flexibility, security and cloud deployment. With respect to flexibility, LEAP is a data agnostic platform that supports a large set of statistical queries used for exploratory data analysis in addition to supporting training of ML models. LEAP provides a flexible interface that allows the addition of new query types to suit user needs. A user can either use one of the predefined queries available in the LEAP API, or they can create their own custom queries by composing a set of primitive query building blocks.

In terms of security, the LEAP system has multiple components to provide a layered defence. The LEAP infrastructure at each site has different modules that are each responsible for either accessing the data, executing queries, or communicating with the infrastructure in the cloud. Furthermore, the federated nature of the system gives continuous control over data access to the data owners. The infrastructure in the cloud also follows a similar design where different LEAP functionalities are divided into modules. For example, user access control, site coordination and query execution reside in different services. Some participant sites may contain sensitive data, and even though federated queries might provide an initial layer of defense against attackers, it does not provide formal privacy guarantees. With this in mind, LEAP was designed to allow for random noise to be added to queries in order to provide differential privacy guarantees.

The third design goal was to support cloud deployment for part of the infrastructure. Namely, the aggregation and coordination are hosted in the cloud, which facilitates IT management, scalability, and gives us flexibility with different server types. This goal also goes hand in hand with security and flexibility. A cloud deployment allows us to use robust logging and monitoring tools from a cloud provider to improve system security and use the scalability of the cloud to support more computationally intensive queries.

In the rest of this chapter I will cover the threat model delineating the capabilities of different components and agents. I will also cover in more detail the LEAP architecture and how data and requests flow through the system.

3.1 Threat Model

Sites: We model sites as honest but curious. We assume they will properly follow the protocols in the LEAP system. We trust sites to participate correctly in the machine learning training process and use their local data to return a correct result. Nonetheless, we do not trust a site with another site's data. For example, if a site gets hold of another site's sensitive data, we do not trust them to not inspect the data.

Coordinator and Cloud Algo: The coordinator and cloud algo are hosted in the cloud. We do not trust the coordinator and cloud algo with the raw data from the sites. However, we do trust these cloud-hosted services to honestly participate in the LEAP system and trust them with the query results from each site.

User: Users of LEAP are not trusted with sensitive data from each site, but they are allowed to have access to aggregated results for executed queries. Different types of users have different levels of trust attached to them, as defined by the capabilities associated with each role.

3.2 Architecture

The LEAP infrastructure (Figure 3.1) is divided into two groups, the cloud infrastructure and the site infrastructure. The cloud infrastructure forms the first point of contact between a user and LEAP. Users send requests to the **coordinator**, which is responsible for managing connections to different sites and authenticating user credentials. Credentials are kept in a **user database**. With each request LEAP checks whether the user is a valid LEAP user and has the capabilities to issue the query. The other job of the coordinator is to orchestrate and send the request to the correct sites.

The point of contact between the cloud and a site is the **site connector**. The site connector is responsible for registering with the coordinator, so that the cloud knows which sites are available, and it takes incoming requests from the coordinator and passes these down to the appropriate site algo. The site connector is also responsible for detecting whether the site algo is down and returning the appropriate error to the coordinator. At the moment, LEAP runs only one site algo instance at each site, but in the future the plan is to be able to scale the computational modules up and down on demand at each site.

The **site algo** is the main computational module at each site. With each request the site algo uses one of the data connectors such as the **REDCap connector** [6] or the **OpenSpecimen connector** [7] to fetch the appropriate data. This data is consumed by the site algo, which performs the user requested computation and returns a result back to the site connector which pipes it back to the cloud through the coordinator.

The coordinator gets results from all the sites and pushes these results to the **cloud algo**, which is the computational module in the cloud. The cloud algo acts as an aggregator for all the results and can return the aggregated result back to the user, and controls the logic of when to stop and return the result to the user if the query is iterative, such as when training a model using federated learning. DP noise can also be added at this step for global differential privacy or at the site algo level for local differential privacy.

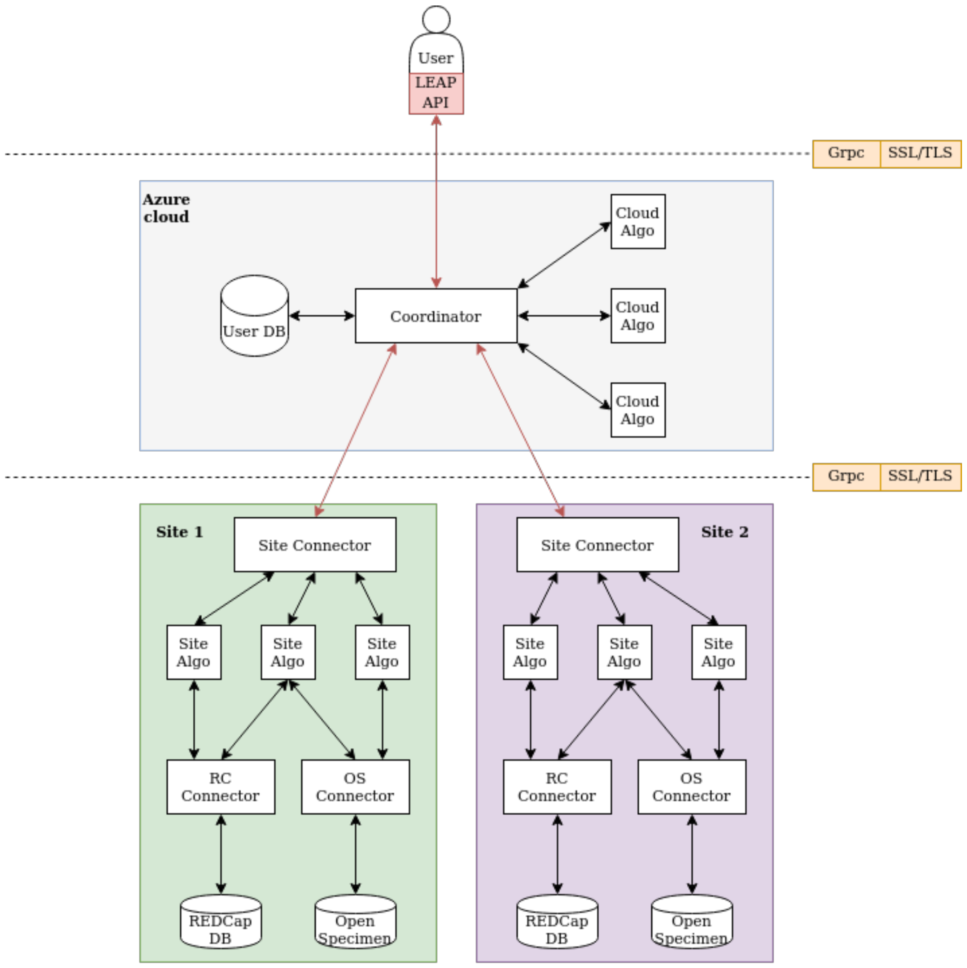

Figure 3.1: The coordinator connects to the site connector module on each site. At each iteration the site algos compute a result on the local data which gets sent to the coordinator by the site connector through a secure connection. Results are then aggregated in the cloud algo modules hosted in the cloud.

3.3 Data Flow

LEAP is designed so that raw data from a site does not leave the site. Namely, data is exchanged between different components in a site, such as the data connectors and site algo, but this data is never passed to the coordinator residing in the cloud. LEAP only pipes the result of running a certain query on the local data to the cloud for aggregation. Below is a description of the relevant remote procedure calls issued by the different LEAP components when computing the result for a query. A time-sequence diagram in Figure 3.2 visualizes the process.

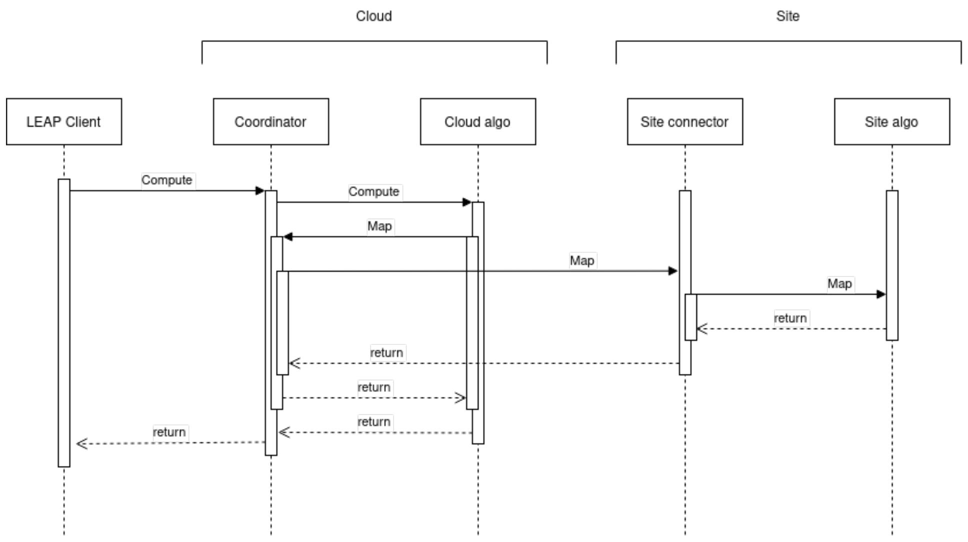

Figure 3.2: Data flow for executing a query in LEAP.

3.3.1 Message Types

ComputeRequest

- AlgoCode: Specifies what algorithm to run
- Sites: Specifies what sites to run the algorithm
- Functions: Custom building block functions (custom algorithm)

ComputeResponse

- Response: Aggregated result from running algorithm

MapRequest

- AlgoCode: Specifies what algorithm to run
- Sites: Specifies what sites to run the algorithm
- Functions: Custom building block functions (custom algorithm)

MapResponse

- Response: Individual map results from each site

3.3.2 Coordinator

ComputeResponse ← Compute(ComputeRequest)

- Caller: Client
- Callee: Coordinator
- Description: Client calls coordinator which starts the execution of the specified algorithm.
- Arg: ComputeRequest
- Return: ComputeResponse

MapResponses ← Map(MapRequest)

- Caller: Cloud algo
- Callee: Coordinator
- Description: Receives state from the cloud algo (can include model parameters) and collects results of map computation from each site.
- Arg: MapRequest
- Return: MapResponse

3.3.3 Cloud Algo

ComputeResponse ← Compute(ComputeRequest)

- Caller: Coordinator
- Callee: Cloud Algo
- Description: Coordinator makes RPC call to cloud algo to start the execution of the specified algorithm
- Arg: ComputeRequest
- Return: ComputeResponse

3.3.4 Site Connector

MapResponse ← Map(MapRequest)

- Caller: Coordinator
- Callee: Site connector
- Description: Sends a request to the local site algo service requesting for map function to be computed on local data. Returns the result to the coordinator.
- Args: MapRequest
- Return: MapResponse

3.3.5 Site Algo

MapResponse ← Map(MapRequest)

- Caller: Site connector
- Callee: Site algo

- Description: Runs map function on local data and returns result to the site connector.
- Args: MapRequest
- Return: MapResponse

This chapter covered the different components and their roles in the LEAP system. It also covered the LEAP architecture and how a query from a client flows through the system. The next chapter will go into implementation details, such as the message protocols used, the programming languages used for the different components, how LEAP ensures security through authentication and encryption, and how users are represented in LEAP.

Chapter 4

LEAP Implementation

LEAP was implemented as a series of modules that can be divided into two main groups. The infrastructure modules implemented in Go, such as the coordinator and the site connector, and the computational modules implemented in Python, such as the cloud algo and the site algo. This chapter gives an overview of the implementation of the different components in LEAP. Furthermore, this chapter also covers implementation details of security protocols such as encryption, authentication, and message protocols. Finally, the end of the chapter covers details on how users interact with the system and are authenticated and created.

4.1 Components

Cloud Algo: The cloud algo is written in Python and is responsible for aggregating the data from the multiple sites and post-processing them before returning an aggregate to a user. To get a response from pre-selected sites, the cloud algo uses GRPC to send a protobuf message to the coordinator, which handles all communication between the cloud and a local site.

Coordinator: The coordinator, written in the Go programming language, is responsible for managing all the sites available in the LEAP system. It is responsible for propagating the proper error messages to the cloud algo when a site is down, and for passing the computation requests issued by the cloud algo to the proper sites. When a coordinator contacts a site, it serializes the request using protobuf and uses grpc to issue a remote procedure call to each site. The coordinator has a multithreaded design in which one goroutine (Go thread) listens for cloud algo connections and another goroutine listens for site connector connections.

Site Connector: The site connector runs a grpc server written in Go that listens to requests from the coordinator. Before showing up as available, the site-connector sends a registration request to the coordinator. After this is done, the site connector can receive computation requests from the coordinator and delegates them to a node running the site algo. Communication between the site connector and the site algo also uses grpc and protobufs for serialization.

Site Algo: The site algo is a Python program responsible for retrieving the appropriate data from a database and running some sort of computation on the data. If differential privacy (DP) is turned on, the site algo is responsible for taking the appropriate measures to make the result differentially private, such as adding Laplacian noise to the result. The site algo uses the REDCap API to retrieve the data from a REDCap database or a REDCap external module that is optimized for cases where all the necessary computation can be done in a SQL query.

User Database: The user database keeps a record of all the registered users in the LEAP system. It keeps track of their username, password and the sites the user has access. When a new request arrives at the coordinator from a user, the coordinator checks whether the user has the permissions to perform the requested computation on the selected sites.

4.2 Message Protocols

Each service in the LEAP infrastructure runs a server that listens to requests from the appropriate services at a port specified in a configuration file. These services communicate using remote procedure calls and utilize TCP as the underlying transport protocol for reliable message transfer. We use Google's Grpc [3] library as the remote procedure call implementation and protocol buffers as the serialization method. Protocol buffers [5] provide us with faster serialization than JSON, although for some messages we still pass JSON blobs for convenience. These JSON blobs are used to pass Python functions that are executed in the LEAP infrastructure. Each JSON blob is turned into a string that is then passed as a field in the protobuf structure. The protobuf structure is deserialized at an endpoint, the JSON string is extracted, and the functions are executed accordingly.

4.3 Node Encryption and Authentication

LEAP uses mutual TLS/SSL [33] to encrypt connections between nodes and authenticate the identity of nodes in the system. An internal certificate authority is used to authenticate connections between nodes. This internal certificate authority generates a 2048 bit private RSA key that is used to generate a SHA256 self-signed root certificate that is valid for a specified number of days. This root certificate is distributed to the trusted nodes in the system (coordinator, cloud algo, site connector, site algo). Each one of these nodes generates a private 2048 bit RSA key, and a SHA256 certificate signing request, which can be adjusted based on the certificate authority being used. This certificate signing request is signed with the private key of the node that generated it and is sent to the local LEAP certificate authority using the administrative procedures in place. The signing request is signed with the private key of the certificate authority and is returned to the requesting node.

Every connection established between nodes is authenticated by having the nodes on both sides of the connection send their certificates to each other. This certificate is checked against the root certificate to ensure that it is coming from one of the nodes trusted by the LEAP administrators. After authenticity is verified, the two nodes communicate using an encrypted connection as established by the SSL/TLS protocol.

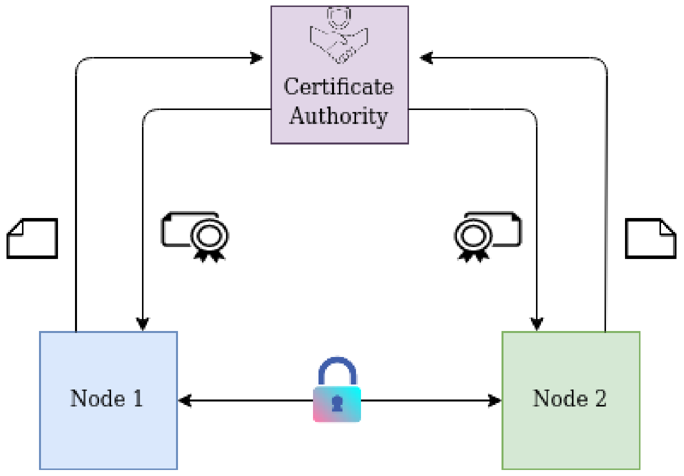

Figure 4.1: Nodes send their certificates to a certificate authority. The certificate authority signs and returns the certificate to the LEAP node. Nodes establish mutual trust and open an encrypted connection by verifying the signed certificates against their copy of the root certificate.

4.4 Users

Users can issue queries to different locations based on their permissions. Each user is associated with a list of sites that they have access to based on their credentials, and they are only allowed to issue queries to those sites. This state is stored in the User DB shown in Figure 3.1.

4.4.1 User Interface

Users interact with LEAP through a programming interface in Python. A Python library exports the necessary functions to register and authenticate a user, which returns a token to be used in subsequent requests. The library also exports the functions available in LEAP and users can select the sites to send their queries.

4.4.2 User Creation

When new users are created, they pick a username that uniquely identifies them in the LEAP system and a password that is used to login. Only the hash of the password is stored to prevent an attacker from acquiring the password used by the user. On top of that, user passwords are salted, so that a unique hash is created for every input.

In the salting process, a unique salt is created for each user and this salt is appended to the user password before hashing the password and storing it in the database. For example, suppose a user chooses catdog as the password. During the sign-up process we will generate a salt, such as b3ob, for that user. The salt is then appended to the password before being hashed and stored. This means that when the user signs up we store the hash of catdogb3ob. Salting is meant to prevent rainbow table attacks, where an adversary uses a precomputed table of hashes to recover the original cleartext password of a user.

4.4.3 User Authentication

After a user is created, that user requires a token to send computation requests to the LEAP system. This token is obtained by calling the LEAP API and passing the username and password for that user. We append the salt stored for that user to the password inputted and hash the appended input. If the input hash matches the hash stored in the database, LEAP returns a JWT token [4] to the user that expires after a certain amount of time. The user then has to attach this token to any request sent to LEAP. This token is then validated and parsed at the coordinator and the claims are checked to extract the user associated with the token. If the user exists, LEAP checks whether this user has permissions to query the selected sites.

Capabilities are associated with users out-of-band, where a prospective user contacts the person responsible for the LEAP system. The administrator will then associate the user with the appropriate query capabilities and access to sites.

This chapter covered the implementation details of the LEAP system. Namely, the libraries and programming languages used to build the different components and ensure communication is properly encrypted and authenticated. We also covered how users are represented in LEAP and how they interact with the system. The following chapter evaluates the design and implementation details provided in this chapter. Namely, it evaluates the overhead of using LEAP with different types of queries.

Chapter 5

LEAP Evaluation

In this section, we aim to answer four research questions:

1. What is the overhead of using LEAP?
2. Where is most of the time spent during execution of a query?
3. How does adding more sites affect execution time (distributed overhead)?
4. How does having more data available affect the convergence time of a machine learning model in the LEAP system?

To answer these questions we designed a set of experiments where we compare LEAP to a centralized baseline that has all the data stored locally. We run different types of queries and train different machine learning models using LEAP to measure the overhead incurred by the system. Our experiments reveal that there is an overhead associated with using the LEAP system. This overhead is more pronounced when training more complex models such as Resnet-18 over running simple statistical queries such as count. Nonetheless, when enough sites join LEAP the convergence time for a model becomes similar to the centralized baseline.

5.1 Experimental Setup

Experiments were conducted in a cluster of 17 nodes. Each node is an Azure Standard D4s v3 Ubuntu 18 virtual machine with 4 vcpus and 16 GB of memory.

15 out of the 17 nodes were used as LEAP sites and had the site algo and the site connector modules installed. Furthermore, a REDCap instance was installed on each of the sites and populated with tabular data from the HAM10000 skin lesion dataset [37]. Skin lesion images were saved to the hard disk of the site virtual machines. One of the 17 nodes was used for the cloud infrastructure and had the cloud algo and the coordinator modules deployed. One other node was used as a client that issued query requests to the coordinator and waited for the result of a query.

The nodes in the cluster were deployed in different Azure regions. The client and the cloud algo, along with three sites were deployed in the West US. The other three sites were deployed in East Australia, East Asia, East US, and West Europe. The sites were deployed in different regions to mimic a federated network of hospitals and health centres distributed around the globe. Each experiment was run four times and the end to end training time was averaged.

5.2 Performance Overhead

The results in this section help answer three out of the four proposed research questions, while the question on model convergence is answered in Section 5.3. Namely, this section discusses the overhead of using LEAP, the stages that cause the overhead, and how adding more sites affects the overhead. We first discuss the associated overhead to train machine learning models such as Resnet-18 and a simple logistic regression, and then we discuss the overhead associated to running a statistical query, such as count.

To evaluate the scalability of LEAP when training deep learning models the images from the HAM10000 dataset were divided between each of the sites. This resulted in 8000 training images being equally distributed to the sites, so that each site contained a total of 533 images. Resnet was then trained to classify the type of skin lesion from an image using the federated learning algorithm implemented in LEAP. Resnet was trained for a total of 1000 iterations, where 100 of those were global iterations where the cloud algo aggregated the model updates for each site and for each global iteration 10 local iterations were computed at each site. The baseline is a Resnet-18 model trained on 8000 images stored on disk. REDCap does not support images well, so for the Resnet experiment, both on the baseline and site VMs, the training dataset was saved and retrieved from disk while training. It is important to note that when training these machine learning models, the amount of data available on each site is not important for measuring the overhead of our system. At each federated learning iteration LEAP performs the same number of local iterations. Therefore, even if a site has more or less data, the amount of computation done at each iteration is the same.

As the bar chart in Figure 5.1 demonstrates, there is an overhead to training Resnet using LEAP. Most of the time is spent in the site compute stage, which is where gradients are computed on the local data. Interestingly, more time is spent during the site compute stage than in the total baseline time. In an ideal scenario, the time spent on both of these should be very similar, but LEAP ends up spending extra time serializing and deserializing gradient and model parameters for sending and receiving data from the coordinator. Despite being a smaller portion of the total running time, LEAP also spends time in the cloud compute stage. The cloud compute stage is where gradient aggregation happens, which explains why the time spent on this stage increases as the number of participating sites incrases. The more participating sites, the more gradients have to be aggregated. As the number of participating sites continues to increase, the time spent in the site compute stage may overshadow the site compute stage.

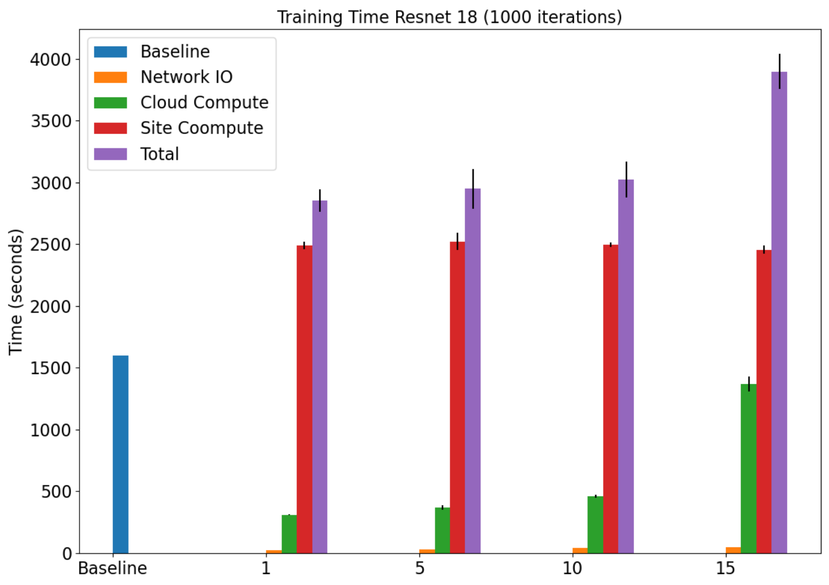

Figure 5.1: Time to train Resnet 18 using HAM10000 dataset on 1, 5, 10 and 15 sites.

Similar to the Resnet-18 experiment, the logistic regression experiment trained a logistic regression model on the HAM10000 dataset with the objective of classifying the gender of the skin lesion from tabular data on the dataset. Since this experiment deals only with tabular data, the dataset was stored in REDCap and retrieved during training using API calls sent to the REDCap database. The logistic regression was also trained for a total of 1000 iterations, where 100 of those were global iterations where the cloud algo aggregated the model updates for each site and for each global iteration 10 local iterations were computed at each site.

As seen in Figure 5.2 the logistic regression experiment shows that LEAP has a similar total running time to the centralized baseline when only one site is added, but this overhead becomes significantly higher when more sites are added. The time spent in the cloud compute stage is significantly larger than the time spent in the site compute stage. This can be partly explained by the less compute intensive nature of the logistic regression when computing gradients. Nonetheless, the overhead still seems higher than expected and it is not clear what is causing this.

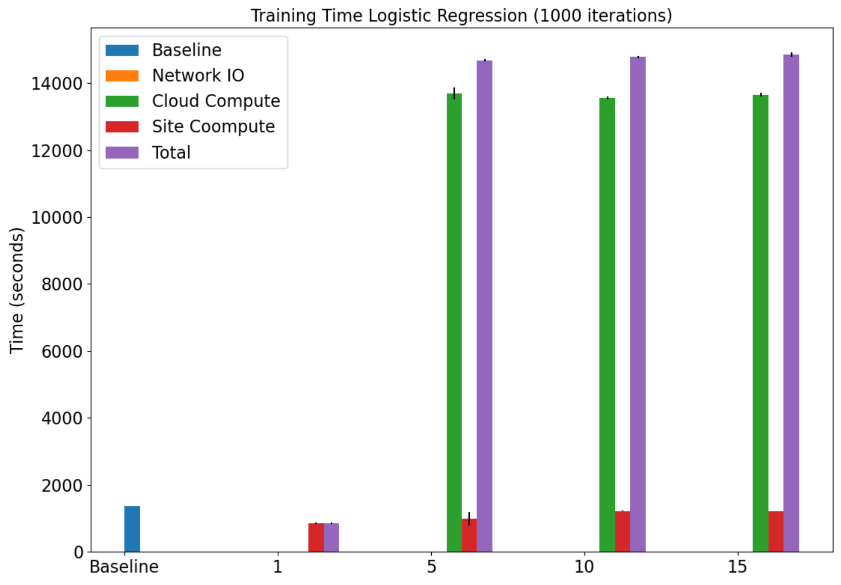

Figure 5.2: Time to train a Logistic Regression modelusing HAM10000 dataset on 1, 5, 10 and 15 sites.

On top of training machine learning models, we were also interested in understanding the performance of LEAP when executing simple statistical queries. One of the big differences between these queries and the model training queries is that they are what we call single-shot queries, results get aggregated only once so there is no iterative back and forth, and they are less computationally intensive.

With the statistical queries we were concerned with understanding the performance of a query as more sites join LEAP and increase the size of the total data available. The HAM10000 dataset was divided to each site, so that every site held 10000 records. A count query that retrieved all of the data and then counted the number of records was then executed with different numbers of sites. The time for the query to be returned to the client was averaged among four runs and is displayed in Figure 5.3.

In this experiment we can notice that the time to compute a count query increases when there are more participating sites in LEAP. The overhead seems to plateau when five sites are added and only slightly increases from 10 participating sites to 15. This demonstrates that multi-site LEAP has an overhead when scaling the size of the dataset. The increase in execution time is caused by more time being spent aggregating the data from the different sites.

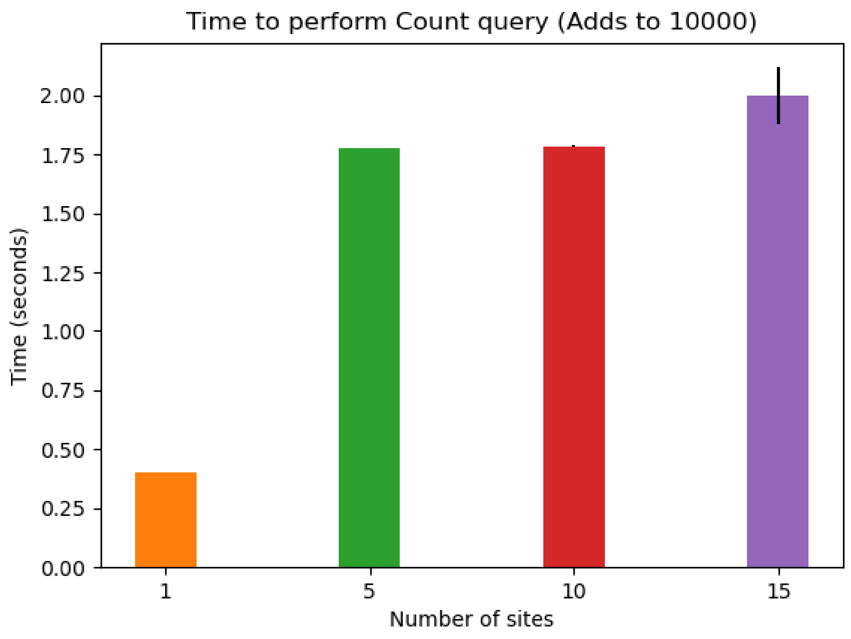

Figure 5.3: Time to perform a count query on different number of sites. Each site had a total of 10000 records

5.3 Convergence Time

As the experiments above have shown, there is an overhead to using LEAP. One of the advantages of using LEAP is that by connecting different sites and research centres we have more data available for researchers. This brings us to our fourth research question: how does having more data available affect the convergence time of a machine learning model in the LEAP system? How can the accuracy of a model improve as more data becomes available? To measure this we trained Resnet-18 using LEAP and increased the number of participating sites at different runs. The 8000 training images from the dataset were divided to each site so that each site had a total of 533 images. For example, in a run with one participating site the one site would have 533 images for a total of 533 images in the LEAP network. With five participating sites, each site would have 533 images for a total of 2665 images in the LEAP network.

As seen in Figure 5.4, as more sites join the LEAP network, the convergence time to train the model to convergence decreases. With only 533 images available, one participating site wasn't even able to increase the validation accuracy of the model. As more sites were added to the system, the convergence time improved. With 15 sites, and a total of 8000 images available in the network, LEAP was able to make the convergence time similar to the centralized model with 8000 images. Essentially, despite the overhead incurred by using LEAP, the larger datasets available in a deployment setting give benefits in the quality of the results.

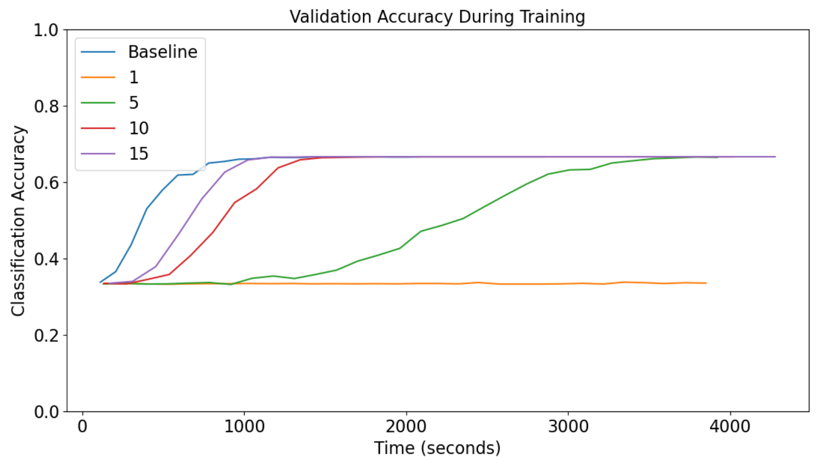

Figure 5.4: Convergence time and accuracy to train Resnet 18 measured in number of total seconds.

In this section we covered different experiments designed to measure the overhead of using LEAP, find where the overhead comes from, inspect how adding more sites affects execution time, and answer how having more data available affects the quality of results, such as when training a machine learning model. These experiments showed that there is an overhead to using LEAP, coming from time spent aggregating results from different sites and serializing and deserializing these results. Furthermore, as the number of participating sites increases, the overhead also increases due to more time being spent in the aggregation. Nonetheless, LEAP shows gains in the quality of trained models, with better accuracy and faster convergence when more sites are added. Given these results, the next chapter will compare LEAP to similar systems and will explore the similarities and differences between them and LEAP.

Chapter 6

LEAP Related Work

Coordinating health research with data distributed across several institutions poses several important challenges including ensuring privacy without revealing confidential patient information. Furthermore, different health organizations may collect data with vastly differing representations requiring reconciliation before further analysis can be carried out. Moreover, sending data to other sites for analysis, especially high-dimensional imaging and genomic data, can incur significant communication and computation costs.

Federated and distributed research networks are useful in facilitating cross-site research where privacy and security policies may prevent data from leaving the governance of the data owner or health organization. These systems can also provide gains in speed by foregoing data centralization.

6.1 CanDIG

CanDIG (Canadian Distributed Infrastructure for Genomics) [1] is a Canadian p2p-distributed, secure and private platform aiming to enable healthcare research for genomics data. CanDIG has similarities to LEAP. Namely, both are distributed and support differential privacy. Furthermore, local sites control access to the data. CanDIG differs from LEAP in its architecture, which is peer to peer. Furthermore, CanDIG offers limited support for machine learning tools. At the moment CanDIG only supports a Decision Tree Classifier.

6.2 DataShield

DataShield [2] is a free to download R library that allow for federated data analysis using non-disclosive summary statistics from local sites. Queries are issued through an Analysis Computer (AC) through secure web services to multiple Data Computers (DCs) and analysis at each local site is done simultaneously. DataShield is popular in the healthcare community, but different from LEAP in that it does not provide machine learning methods. Furthermore, it does not support differential privacy like LEAP and CanDIG.

6.3 COINSTAC

The Collaborative Informatics and Neuroimaging Suite Toolkit for Anonymous Computation (COINSTAC) [32] is a Cloud-based distributed platform for large scale analyses of brain imaging data with the intention to allow federation of neuroimaging data motivated by privacy and data sharing concerns, and the size and dimensionality of neuroimaging data. COINSTAC's data is kept private through differential privacy algorithms. In addition, users are able to build pipelines for additional steps such as feature generation, matrix factorization models, and preprocessing to their workflows (i.e. may want to convert file formats or apply batch transformations to imaging data) – this can be accomplished relatively simply through config files and scripts. This system uses summary statistics of local nodes, which are then aggregated at a remote node and analyzed using global inference rules. Despite some similarities such as support for differential privacy and deployment in the healthcare setting, COINSTAC is not a general purpose analysis platform such as LEAP. For example, the COINSTAC toolkit seems mostly geared towards brain imaging data.

This chapter covered the similarities and differences between LEAP and other popular data sharing systems, specially in healthcare. This wraps up the LEAP section of this thesis. The following chapters will now delve into the SmartCache and how it can be used to optimize the use of the privacy budget when answering DP queries.

Chapter 7

SmartCache

In the differential privacy framework, each query issued spends a part of the privacy budget. The budget presents an upper limit on how many queries can be issued before information leaks. This provides a practical problem to data analytic systems because it restricts the number of queries that can be issued by users. Successive DP queries compose and incur a compounded privacy loss. For example, if a user issues five DP queries to the LEAP system, each with a privacy loss of ε, the total privacy loss for the five queries is 5ε. This means that the privacy budget may be depleted if too many queries are issued, thus blocking users from using the LEAP system if only DP queries are allowed. A key challenge in LEAP will be to maximize the utility of analysis queries (i.e., the accuracy of analysis results and models) over time, while maintaining DP guarantees.

An analysis in LEAP is assigned a privacy budget, and each new query that runs on the data spends a portion of that budget. To maintain DP guarantees, this budget cannot be overspent. To maximize the value of this privacy budget we developed a system called SmartCache, which works by generating a DP synthetic dataset and answering queries from this dataset with a smaller privacy loss. If the answer from the synthetic dataset is not accurate, the SmartCache falls back to the real dataset. The following chapters cover synthetic data generation, improving the accuracy of results from the synthetic dataset, making better use of the privacy budget, and concludes with an evaluation of the accuracy and privacy costs of the SmartCache.

Chapter 8

SmartCache Threat Model

Data Curator: We assume there is a data curator that controls access to a dataset *D* with sensitive information. This data curator may have access to privacy leaking data and they provide data access to the SmartCache. We assume the data curator will not leak information to other participants.

SmartCache: We assume the SmartCache is honest and runs in a trusted environment with permission from the data curator and access to the real dataset. Untrusted parties or programs do not have access to the internals of the SmartCache and cannot influence it in any way. The SmartCache provide an interface that allows queries to be issued to the system and only returns differentially private results. We assume this interface cannot be modified by a malicious agent so that more than the differentially private result of a query is released.

Users: We assume users are malicious and will take advantage of information disclosure of sensitive information. Nonetheless, we also assume they only have access to *results* of queries issued to the SmartCache. Users can collaborate with each other to join query results to form a coordinated attack, but they cannot query the dataset when the collective privacy budget has been depleted.

Chapter 9

SmartCache Design

The goal of the SmartCache is to mitigate the privacy loss from repeated queries while maintaining good accuracy. We achieve this by generating a DP synthetic dataset from the real data. Queries answered from the synthetic dataset use only a small portion of the privacy budget, so the objective of the SmartCache is to maximize the amount of times we can accurately answer a query using the synthetic dataset, and falling back to the real dataset when the query cannot be answered with precision from the synthetic dataset. This is done by using a three step process comprised of a decision mechanism, a query extrapolation procedure and a quality check. If a query can be answered using the synthetic dataset we consider this to be a *hit*, and if it is answered from the real dataset we consider it a *miss*. When a query arrives, we use the decision mechanism to determine whether to use the cache. This is done by comparing results from queries on subsets of the synthetic dataset. If the SmartCache is not confident in a hit, it returns the query result on the real data with noise and consumes the query budget. On the other hand, if the SmartCache is confident in a hit then it runs the query extrapolation procedure. In this step, SmartCache queries a DP dataset generated from the real data. If the result of the query in the synthetic dataset has low error SmartCache returns the synthetic result. If the error is high SmartCache has a miss and return the result from the real dataset with DP.

The insight behind the SmartCache is that we use misses as an opportunity to make our cache ”smarter”. Weight parameters are used to reweight the contribution of each example in the synthetic dataset, are updated with each miss with the objective of minimizing a cost function that measures the difference between results. The update procedure leads to more cache hits and allows us to consume less of our privacy budget because the synthetic dataset is differentially private.

Results from the SmartCache can be used to answer a variety of statistical queries or to train a machine learning model. In the case of statistical queries, we have added support for different types of mean, histogram and count queries (the API can be extended to add more). The user issues a query to the SmartCache and the result is returned using the heuristics described above. Statistical queries that result in a hit are computed using the synthetic dataset. Each query computed from the synthetic dataset uses the cache's parameters to reweight the contribution of each row in the synthetic dataset.

Gradient queries are used to train machine learning models. For instance, a user can train a model by sending a gradient query to the SmartCache at each iteration of the training procedure. The query takes the model parameters and optimizer as input and the SmartCache returns a gradient update computed from either the synthetic or real data. This gradient update can then be used to train a local model.

The following sections will go deeper into how the SmartCache works. Namely, we detail how data is generated, how the decision mechanism decides whether SmartCache should use the cache or not, how weights are updated in the synthetic dataset, and how we ensure that the SmartCache returns accurate results in a DP manner.

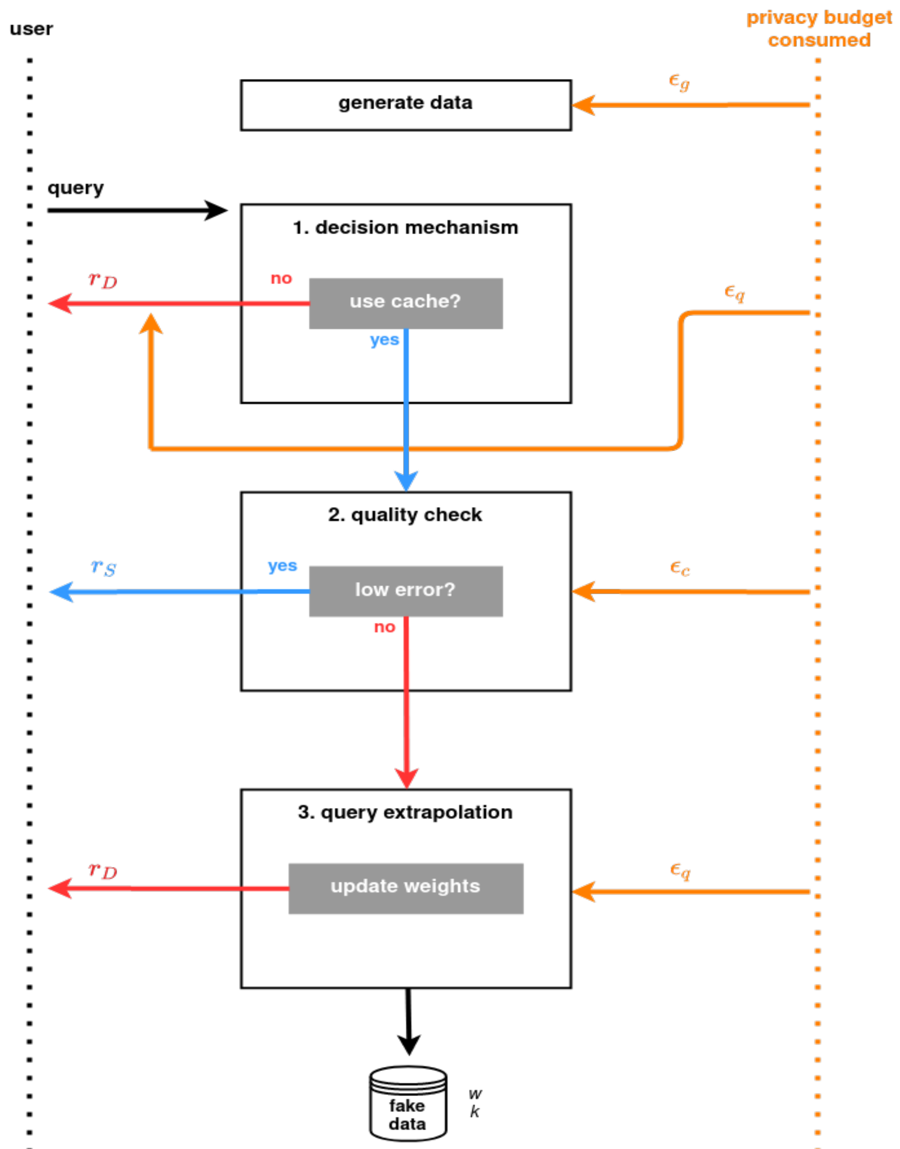

Figure 9.1: When the SmartCache is initialized, synthetic data is generated at a cost of ε_g. A new query goes through three components: decision mechanism, quality check, and query extrapolation. These components decide whether results from the synthetic dataset r_S or the real dataset r_D are released. On top of ε_g budget spent for data generation, the SmartCache consumes ε_q when a query from the real dataset is released and ε_c when the quality check is run.

9.1 Data Generation

The synthetic dataset is generated using an independent feature generator. For each numerical feature in the dataset the mean and variance of the feature is calculated. DP noise is added to the mean and variance to guarantee differential privacy. Values are then randomly sampled from a normal distribution with the calculated mean and variance. The synthetic dataset is populated with these random samples for each feature column in the dataset. For the categorical features the unique values in each column are identified and a histogram is created with the frequency of these values. DP noise is then sampled and added to each frequency bin to guarantee that the generation process is DP. The frequency of each value is then used to randomly generate samples with the probability given by the frequency bins. After the synthetic data is generated, each row in the dataset is assigned a weight, as seen in Figure 9.2. Weights are uniformly initialized, so that each row has the same weight.

The independent feature generator will not pick up correlations between columns. Nonetheless, if enough samples are generated, the weight tuning process gives more importance to rows that are able to more closely resemble the original dataset. The privacy cost of generating data using an independent feature generator is also smaller than using other methods such as VAEs and GANs.

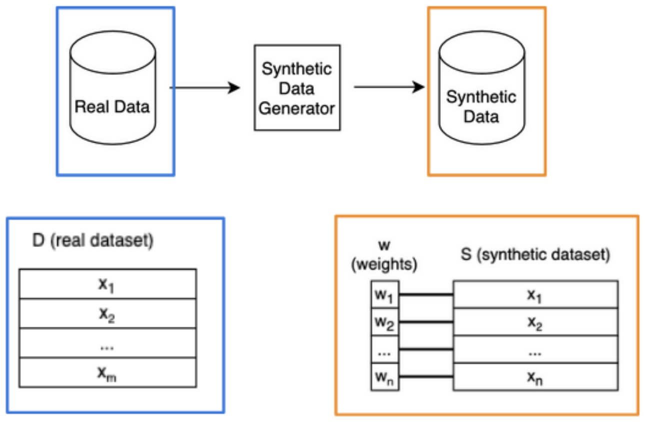

Figure 9.2: The real data is used to generate a synthetic dataset where each row is associated to a weight.

9.2 Decision Mechanism

After data is generated the SmartCache is ready to receive queries. Nonetheless, a good decision mechanism is important so that we only use the SmartCache when there is a high probability of a hit. To do that, we leverage the following insight: if a cache can learn the right answer from past queries, multiple versions/initializations should agree; if it is extrapolating arbitrarily, they should disagree. Therefore, we divide the synthetic dataset into multiple smaller datasets, each with an independent set of parameters k and w that are used to reweight the contribution of each example in the dataset. When a query arrives, it is computed on each of the datasets. The variance of query results is also computed to determine whether there is a chance that the SmartCache will hit because it is unlikely that each dataset will converge to the same poor result. On the other hand, if the variance is high, there is a high chance that the SmartCache will return a poor answer, so we return a result from the real dataset with added DP noise. When a query is answered, the weights are updated on a random subset of the original dataset. The reason behind this is so that there is more variability in the weights of the datasets, otherwise they will converge to similar weights and there will be little difference between results from the different datasets.

Formally, we have a set of n datasets $S_1 \ldots S_n$, each with an independent set of weights $w_1 \ldots w_n$ and $k_1 \ldots k_n$. When a query q arrives it is answered on all datasets individually. The appropriate set of weights w are used to reweight the contribution of a row in the dataset to the query result and the k parameter is used to rescale the result to the appropriate magnitude [1] For example, a weighted count is computed by $k\sum_{i=0}^{n} w_i x_i$. The result from each dataset is then averaged and returned as the final synthetic result. For example, if the SmartCache is initialized with n datasets, queries $q(S_1, w_1, k_1) \ldots q(S_n, w_n, k_n)$ will return results $r_1 \ldots r_n$. To get the final synthetic result we average the results by computing $(r_1 + \ldots + r_n)/n$. We also compute the variance of the results $r_1 \ldots r_n$ from each dataset and use this variance to make our decision. If the variance is above a certain threshold we deem the chance of a hit to be low, so we simply return the answer from the real dataset.

[1] The synthetic and real datasets are often not of the same size. For some queries, such as counts, the result will depend on the size of the dataset. Parameter k is used to rescale the result so that the magnitude of the answer from the synthetic and real datasets match.

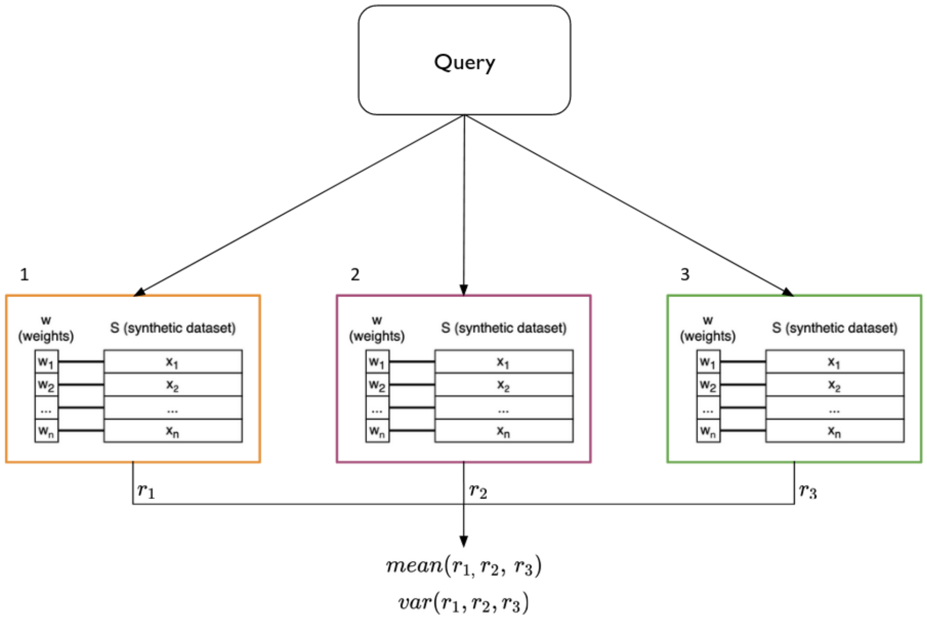

Figure 9.3: The variance of the results between each sub dataset is used to predict whether we will have a hit or not.

9.3 Query Extrapolation

There are two optimization procedures used to update the parameters in the SmartCache and extrapolate results from previous queries. One of them deals with updating parameters when general statistical queries such as mean, count, and histograms cause a miss, while the other deals with improving the weights to answer gradient queries used to train a model. For both procedures we employ an online learning setup that utilizes results from a recently executed query to update the cache's parameters. The update reweights rows in the synthetic dataset. Rows that represent good approximations of the real dataset are given more weight, while rows that poorly represent the dataset are given less weight.

For statistical queries we use the results of the queries executed on the synthetic and real dataset to compute the L1 loss. We minimize the loss by updating the weights w on each row of the dataset. In popular machine learning parlance, the target value is the result from the real data and the predicted value is the result from the synthetic dataset.

The results of queries such as counts and histograms are altered by the size of the dataset, so for some queries we also optimize a parameter k that is used to scale queries on the synthetic dataset. For instance, a synthetic dataset with double the size of the real dataset will produce a count query that is roughly double the same count query on the real dataset. Since the size of the dataset can be privacy leaking, we use a small amount of the budget to initialize k with some random noise before starting the optimization procedure. For each miss we update w and k for multiple epochs E using the same result from the real dataset, but recomputing the query from the synthetic dataset with the new parameters at each iteration. This process is described in Algorithm 1.

Another technique used to alleviate the outsized effect of some query types is the normalization of query results to values between 0 and 1. For example, for mean queries we extracted a normalized result by getting the min and the max of the column that was used to compute the mean, and getting the following normalized result $(r - min(col))/(max(col) - min(col))$. For count queries we simply divide by the parameter k so that we get r/k. Other queries will have their own intuitive computations for normalizing the values between 0 and 1.

During the development process we noticed that without normalization we overfitted to queries with large results, which hindered the accuracy of other queries. Another potential solution is to standardize the dataset itself, but we chose not to follow this route because our synthetic dataset would not be useful for analytic tasks that do not use standardized or normalized datasets and we would still have a problem with large count queries.

Algorithm 1: Parameter Update (Query) E is the number of epochs, $q(S,w,k)$ is a query on the synthetic dataset, $q(D)$ is a query on the real dataset.

```
UpdateCacheParams(w, k, D, S):
    for $t \leq E$ do
        $r_D = Q(D)$
        $r_S = Q(S,w,k)$
        $f = |r_D - r_S|$
        $\nabla f_w = f\frac{df}{dw}$
        $\nabla f_k = f\frac{df}{dk}$
        $w_{t+1} = w_t - \eta \nabla f_w$
        $k_{t+1} = k_t - \eta \nabla f_k$
    end
```

For gradient queries we update the weights on the synthetic dataset each time the SmartCache misses on a gradient query. The procedure for the update, described in Algorithm 2, takes the parameters of a machine learning model θ and the gradient updates computed for this model when using the synthetic dataset and the real dataset. The gradient updates are used to compute L_w, the normalized dot product of two gradient tensors.

$$L_w = \frac{\nabla f_\theta^S \cdot \nabla f_\theta^D}{\|\nabla f_\theta^S\|_2 \cdot \|\nabla f_\theta^D\|_2} \tag{9.1}$$

Furthermore, we take the gradient of L_w with respect to w and minimize the normalized dot product between gradients from the two datasets. For each miss, we update w multiple times as denoted by the number of epochs E. We have found that this helps the cache warm-up faster and find better weights earlier.

Algorithm 2: Parameter Update (Gradient) E is the number of epochs for each update, S is the synthetic dataset, D is the real dataset, θ is the parameters of the regression model trained using loss L_θ, and w is the weights of the synthetic dataset. Gradients w.r.t. to θ are returned from the synthetic dataset in $Q(\theta, w, S)$ and the real dataset in $Q(\theta, D)$.

```
UpdateCacheParams(w, θ, D, S):
    for $t \leq E$ do
        $\nabla f_\theta^S = Q(\theta, w, S)$
        $\nabla f_\theta^D = Q(\theta, D)$
        $g = L_w(\nabla f_\theta^S, \nabla f_\theta^D)$
        $\nabla g_w = g \frac{dg}{dw}$
        $w_{t+1} = w_t - \eta \nabla g_w$
    end
```

9.4 Quality check

The quality check procedure is used to prevent the SmartCache from releasing low quality results. The error between results from the synthetic and real datasets are compared to a user-defined threshold with noise added to it, so that the process is DP. If the error is below the threshold, the synthetic result is released. If the error is above the threshold, the real result with DP is released. For statistical queries we base the error threshold on the distribution of DP noise. A user can set a threshold so that the difference between the real and synthetic result falls within $x\%$ of the values from the sampling distribution of the noise. A smaller value of x leads to more precise queries, while a larger value may be less precise, but will lead to more hits. For example, a query is executed on the real dataset and noise is added to it by sampling from $\mathrm{N}(0, \sigma)$. The user wants a hit if the difference between the real and synthetic value is within 95% of the samples from the Gaussian distribution. The threshold is defined by setting $T = \phi^{-1}(0.95)$, where ϕ is the inverse of the cumulative distribution function.

For gradient queries we use the normalized dot product as the hit threshold. This is a good threshold because it returns a hit when the gradient computed on the synthetic dataset is pointing in a similar direction as the gradient from the real dataset. The user can decrease the threshold if they are comfortable with a greater variation in the direction of the gradients, or they can increase the threshold, so that they only have hits when the direction of the two gradients is close together.

To bound the sensitivity of the threshold for the gradient queries we slightly modify the normalized dot product formula, so that the threshold is

$$T = \frac{1}{ln}\sum_{i=0}^{n}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C}$$

where g_S^{ij} is the gradient for the i-th example and l-th layer on the synthetic dataset and $\tilde{g}_D^{ij}$ is the clipped gradient for the i-th example and l-th layer on the real dataset. This formula has a sensitivity of $\frac{1}{n}$

Proof: Assume the $\ell 1$-sensitivity of function f taken over adjacent datasets D and D' is given by $\nabla_1 f = \max_{D,D'} ||f(D) - f(D')||_1$.

$$\begin{aligned}
f(D) &= \frac{1}{ln}\sum_{i=0}^{n}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C} && (9.2)\\
&= \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C} + \sum_{j=0}^{l}\frac{g_S^{nj}\cdot \tilde{g}_D^{nj}}{||g_S^{nj}||_2 C})\\
&\leq \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C} + \sum_{j=0}^{l}\frac{||g_S^{nj}||_2||\tilde{g}_D^{nj}||_2}{||g_S^{nj}||_2 C})\\
&= \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C} + \sum_{j=0}^{l}\frac{||g_S^{nj}||_2 C}{||g_S^{nj}||_2 C})\\
&= \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C} + \sum_{j=0}^{l}1)\\
&= \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C}) + \frac{l}{ln}\\
&= \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot \tilde{g}_D^{ij}}{||g_S^{ij}||_2 C}) + \frac{1}{n}
\end{aligned}$$

Since adjacent datasets differ by at most one row assume without loss of generality that

$$f(D) = \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot\tilde{g}_D^{ij}}{||g_S^{ij}||_2 C}) + \frac{1}{n} \tag{9.3}$$

$$f(D') = \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot\tilde{g}_D^{ij}}{||g_S^{ij}||_2 C})$$

This means that

$$\max_{D,D'}||f(D)-f(D')||_1 = \max_{D,D'}||\frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot\tilde{g}_D^{ij}}{||g_S^{ij}||_2 C}) + \frac{1}{n} - \frac{1}{ln}(\sum_{i=0}^{n-1}\sum_{j=0}^{l}\frac{g_S^{ij}\cdot\tilde{g}_D^{ij}}{||g_S^{ij}||_2 C})||_1 \tag{9.4}$$

$$= \max_{D,D'}||\frac{1}{n}||_1$$

$$= \frac{1}{n}$$

QED

If the calculated value is within the stipulated threshold, the SmartCache will deem the value to be accurate and will return the value from the synthetic dataset, thus minimizing the use of the privacy budget. On the other hand, if the value is not within the threshold it will fail the quality check. In this case, the answer from the real dataset is returned and more of the privacy budget is consumed. We can control the quality check to use a larger value of ε and thus consume less of the budget. This allows us to find a balance where if there are enough hits we use less of the budget than querying just from the real dataset. For example, in a regular DP setup we will consume a part of the budget with every query. The value of ε cannot be too small or else the accuracy of the query will suffer. In the SmartCache, if a query is a cache miss we consume both ε_q, the privacy loss for the query, and ε_c, the privacy loss due to the added noise in the quality check mechanism. The trick is that we can use smaller values of ε for the threshold, since the accuracy of the query depends less on it. If there is a hit we consume only ε_c. This means that if there are enough hits we end up with privacy savings.

Another option is to use the sparse vector mechanism as a quality check mechanism. The sparse vector technique lets us report queries such that privacy degrades only with the number of queries above a threshold. In the basic formulation of the technique, noise is added to the result of a query and the answer is reported only when the noisy value exceeds the threshold. Therefore, the total privacy loss of a function is the number of queries above the threshold. In the SmartCache setting, if the error from a query exceeds the sparse vector technique threshold the real value is returned and the threshold is renoised. If the error is below the threshold then the value from the synthetic dataset is returned.

9.5 Budget Analysis

The privacy budget in the SmartCache can be divided into three groups: data generation, queries, and the quality check. The synthetic dataset used for the SmartCache has to be generated in a DP manner. This means that noise is added during the generation process and some of the privacy budget is consumed upfront. The more noise that is added to the generated dataset, the more private it becomes, but the less accurate it will be. We use ε_g to stand for the privacy loss stemming from the data generation process. The budget from the generated data is only consumed once during the initialization of the SmartCache. The synthetic dataset can then be used without any more budget penalties.

After the initial data is generated the SmartCache is ready to receive queries. Queries only consume the privacy budget from the SmartCache if an answer from the real dataset is returned. We let ε_q to stand for the privacy loss stemming from each query that misses and that has to return an answer from the real dataset. On top of ε_q each query also consumes some budget from the quality check procedure ε_c. The total budget consumption for the SmartCache if there are n queries and m misses is

$$\varepsilon_t = \varepsilon_g + m\varepsilon_q + n\varepsilon_c$$

If the sparse vector mechanism is used for the quality check the quality check budget will be consumed only when there is a miss. The total budget consumption when the sparse vector mechanism is used is given by

$$\varepsilon_t = \varepsilon_g + m\varepsilon_q + m\varepsilon_c$$

The budget accounting can look a bit different if Rényi differential privacy is used to account for the budget spent by the queries. In Rényi-DP, also known as $(\varepsilon, \alpha) - RDP$, the budget expenditure can be given in terms of the best value of alpha, which gives tighter bounds on the privacy loss of the queries.

Chapter 10

SmartCache Evaluation

The goal of the SmartCache is to reduce the budget spent by a set of queries, thus allowing users to perform more queries without compromising the privacy of individuals in the dataset. On top of that, we also want the SmartCache to provide accurate results. In this evalation we want to answer two research questions:

1. How much budget is spent by using the SmartCache when compared to answering queries directly from the real dataset with DP?

2. How does the SmartCache affect the accuracy of queries and how does it compare to answering queries from a synthetic dataset with uniform weights?

To evaluate the SmartCache we ran experiments on the bank-marketing [28] and the avocado prices [11] datasets. The bank-marketing dataset contains a combination of 16 categorical and numerical features and a total of 45211 rows. The avocado prices dataset also contains a combination of categorical and numerical features but with a total of 11 features and 18249 rows. These experiments measured the accuracy of different statistical queries and gradient queries and calculated the privacy cost of using the SmartCache against querying the real dataset with differential privacy. Experiments were run on an Ubuntu 18 machine with 32GB of RAM and an Intel Core i7-8700 CPU with six cores. A Nvidia GeForce GTX 1080 with 8GB of memory was used as the graphics processing unit.

10.1 Budget Spent

We evaluated the budget savings for statistical queries by executing 600 queries against the SmartCache and recording the privacy loss after each query was executed. We compared the privacy loss of the SmartCache to the privacy loss of directly querying the real dataset with noise added to the result of each query. The blue line shows the budget spent by the SmartCache and the green line shows the budget spent by our baseline (querying a real dataset with DP). The red line shows the amount of budget spent by using just a synthetic dataset with DP guarantees, without the decision mechanism, quality check, and decision mechanism used in the SmartCache. In short, the budget spent by the red line is equivalent to the budget spent in the data generation process. The SmartCache is considered succesful if the blue line is below the green line, or in the case of gradient queries, if the blue line is below the green line before convergence. This means that the SmartCache used less of the privacy budget than the baseline.

When answering statistical queries for the avocado dataset (Figure 10.1), the SmartCache was able to yield privacy savings when using a threshold error of 0.99 and 0.999, but it was not able to yield privacy savings when a more strict threshold of 0.9 was used. The reasons why the SmartCache is not able to yield privacy savings with a threshold of 0.9 are two-fold: there is an initial cost for generating data and misses are more expensive because of the budget spent answering the query and performing the quality check. These results can be improved by either improving the query extrapolation process, so that the weights are better tuned and we have more hits, or by improving the decision mechanism so that we have fewer misses when we decide to use the cache. As seen in Figure 10.2 the SmartCache was not able to yield privacy savings when answering statistical queries on the bank-marketing dataset. It was close to breaking even with thresholds of 0.99 and 0.999, but the initial cost for data generation added extra expenses to the privacy budget that made the total budget consumed by the SmartCache highter than the budget consumed by the DP baseline. There seems to be a dataset dependency in the results, as evidenced by the contrast in performance between the bank-marketing and avocado datasets. A possible reason for these differences is that the hyperparameters chosen for the avocado dataset were better tuned to the task than the hyperparameters chosen for the bank-marketing dataset.

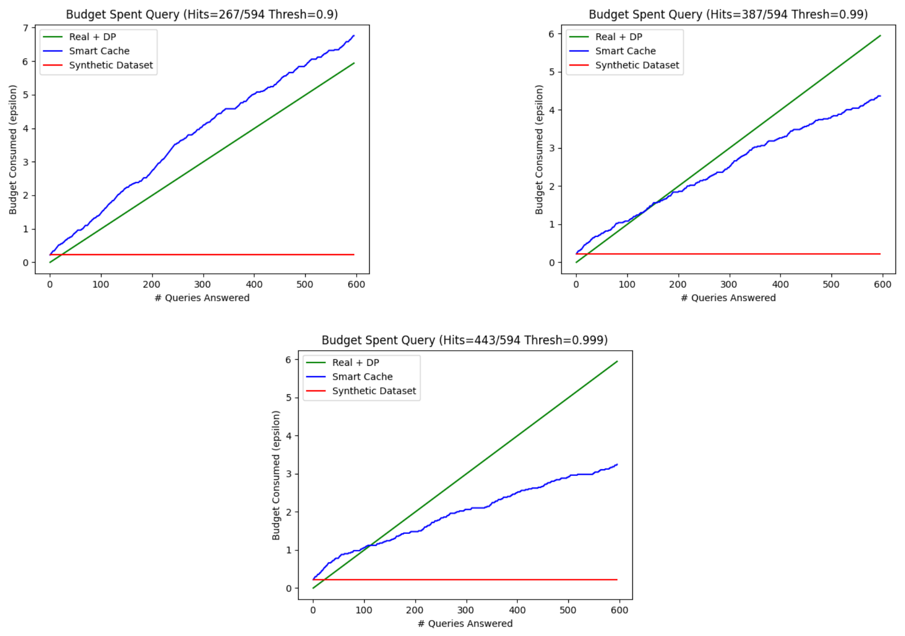

Figure 10.1: Budget spent after answering 500 statistical queries on the avocado dataset. The 500 queries were a mix of mean, histogram and count queries. Each graph shows the results of using the SmartCache with a threshold of 0.9 (top left) 0.99 (top right), and 0.999 (bottom).

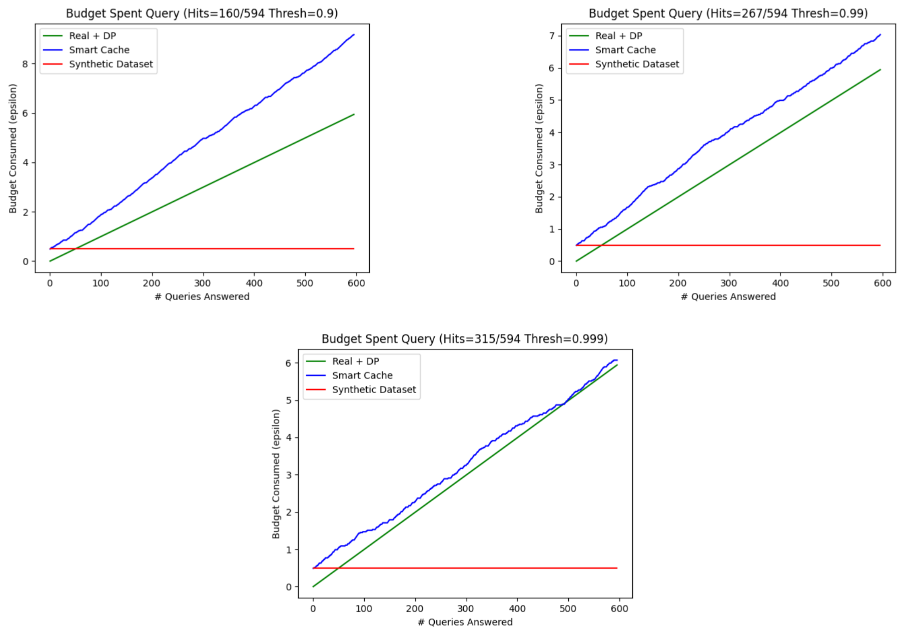

Figure 10.2: Budget spent after answering 500 statistical queries on the bank-marketing dataset. The 500 queries were a mix of mean, histogram and count queries. Each graph shows the results of using the SmartCache with a threshold of 0.9 (top left) 0.99 (top right), and 0.999 (bottom).

For gradient queries we trained both a fully connected model and a ridge regression to convergence by using the SmartCache and recorded the privacy loss after each iteration in the training of the model. We compared the privacy loss of the SmartCache to the privacy loss of directly training a DP model on the real data.

As seen in Figure 10.3 the SmartCache is able to consume a mininal amount of budget for the first 200 queries of the ridge regression. Afterwards, we see the budget consumed by the SmartCache rise rapidly and overtake the budget consumed by the model trained only on the real dataset with DP. This is because of the extra expense in the quality check procedure. This rapid rise in budget consumption can be mitigated by having the decision mechanism make more accurate predictions, but towards the middle of training it becomes innacurate. The budget spent by the SmartCache eventually overtakes the budget spent by the real dataset with DP, but this happens only after the model converges as denoted by the dotted yellow line, which leads to privacy savings. The results for the bank-marketing dataset (Figure 10.4) was not as favourable as the results for the avocado dataset. As denoted by the dotted convergence line, the budget consumed by the SmartCache surpasses the budget consumed by the DP baseline before the model converges. Nonetheless, the total budget consumed by both was similar and the SmartCache was able to stay under the green line for close to three fifths of the queries. Once again, if the decision mechanism is able to identify that we have more misses when closer to convergence, we would be able to have budget savings in Figure 10.4.

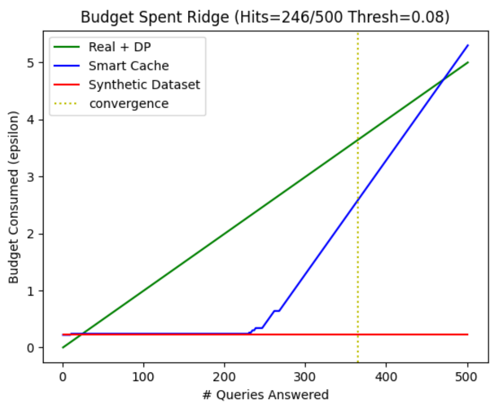

Figure 10.3: Budget spent after each query used to answer ridge regression on the avocado dataset.

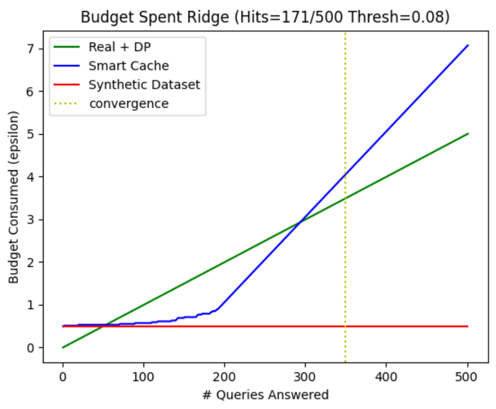

Figure 10.4: Budget spent after each query used to answer ridge regression on the bank-marketing dataset.

With the avocado dataset, the fully connected model performed significantly better than the ridge regression in terms of budget saved. Figure 10.5 shows that the fully connected model consumed a small amount of budget and most of the queries hit the cache. It is not clear why the fully connected model performed so much better than the ridge regression, but with better selection of thresholds and hyperparameters it is possible that the ridge regression can have similar performance. The results with the avocado dataset did not translate to the bank-marketing dataset (Figure 10.6). For the bank-marketing dataset the budget consumed always stayed above the budget consumed by the DP baseline. Careful selection of hyperparameters may help yield better results for other dataset types.

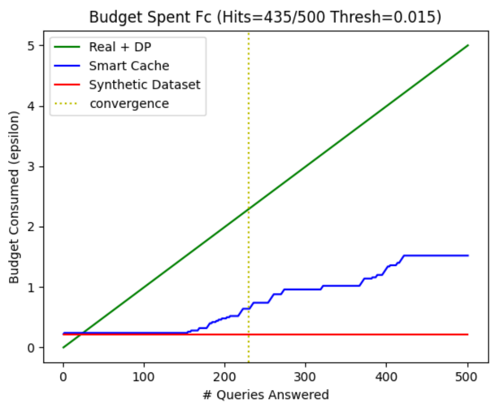

Figure 10.5: Budget spent after each query used to answer fully connected model on the avocado dataset.

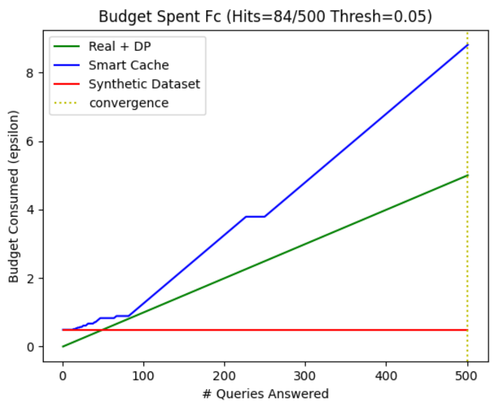

Figure 10.6: Budget spent after each query used to answer fully connected model on the bank-marketing dataset.

10.2 Accuracy

It is important for the SmartCache to return accurate results for it to be practical. We evaluate the accuracy of statistical queries executed on the SmartCache by measuring the mean absolute error of queries executed using the SmartCache to the DP values on the real dataset. We compare this error to the error of returning queries using only a synthetic dataset without the weight parameters used by the SmartCache. Before the absolute error was computed for statistical queries the result from the query on the synthetic dataset and the SmartCache were divided by the query threshold, which is based on the sensitivity of the query. This was done so that the results from different query types can be compared and analyzed together. For example, $a = |(p/t) - (r/t)|$, where a is the absolute error, p is either the predicted value from the SmartCache or the synthetic dataset, r is the real value with DP and t is the query threshold.

As seen in Table 10.1 the SmartCache significantly improves the accuracy when compared to just using a synthetic dataset to answer queries. This shows that it can maintain the quality of queries released to the user within the stipulated threshold. The SmartCache was run for different choices of threshold, so that released queries fall within 90% 99% and 99.9% of DP noise.

Type	Dataset	Thresh	MAE	MAE Scaled
SmartCache	Avocado	0.9	10749.53	0.39
SmartCache	Avocado	0.99	11951.83	0.35
SmartCache	Avocado	0.999	15354.082	0.10
Synth	Avocado	-	259040.38	4.50
SmartCache	Bank-Marketing	0.9	43.19	0.24
SmartCache	Bank-Marketing	0.99	116.33	0.27
SmartCache	Bank-Marketing	0.999	150.81	0.28
Synth	Bank-Marketing	-	1958.72	2.54

Table 10.1: Average absolute error and average variance of queries answered using the SmartCache and the synthetic dataset with uniform weights. MAE is the mean absolute error and thresh is the threshold used to determine hits for the SmartCache.

For gradient queries we trained a ridge regression and fully connected model to convergence using the SmartCache and compared the final loss of the model trained using the SmartCache to a model trained using only the real data with DP guarantees and a model using only DP generated synthetic data.

Figures 10.7 10.8 and Figures 10.9 10.10 show that both the fully connected model and the ridge regression were trained to similar validation loss using the SmartCache as to using the real dataset with DP. The red line in both graphs show the results of training the models with only the synthetic dataset and without any of the weight update procedures used in the SmartCache (weights are uniform for every row). These graphs show how the weight update procedure helps improve the accuracy of the model.

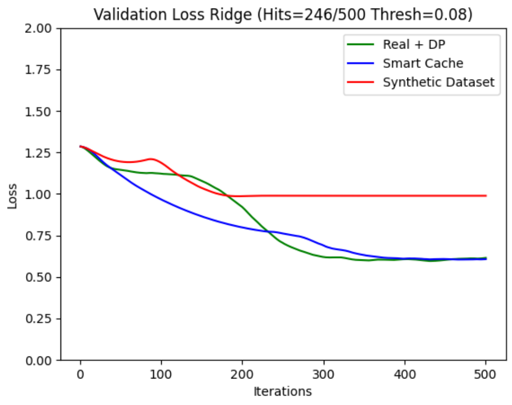

Figure 10.7: Validation loss after each query used to answer ridge regression on avocado dataset.

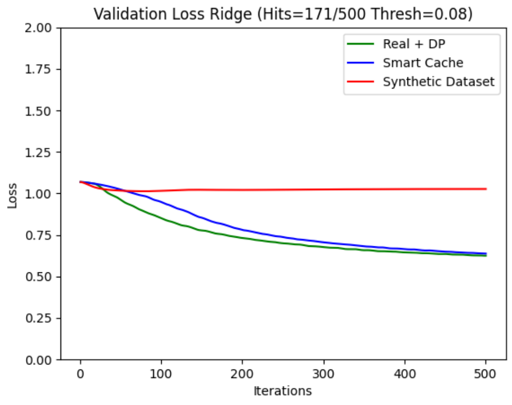

Figure 10.8: Validation loss after each query used to answer ridge regression on bank-marketing dataset.

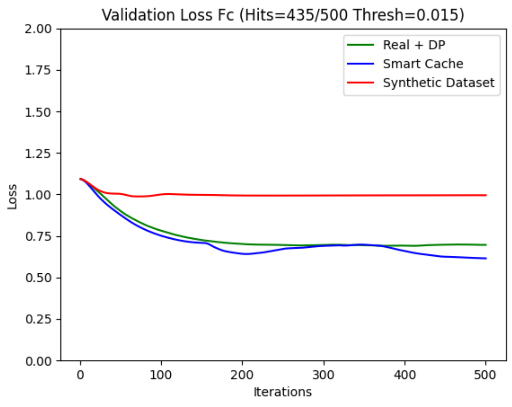

Figure 10.9: Validation loss after each query used to answer fully connected model on avocado dataset.

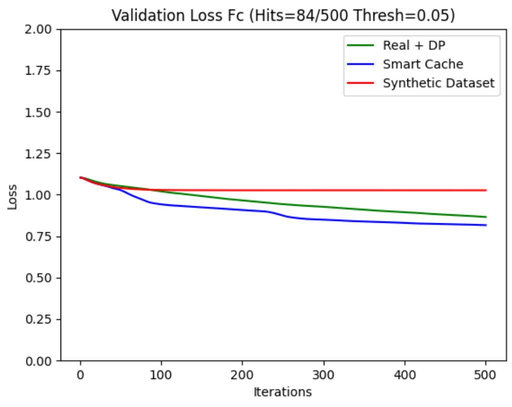

Figure 10.10: Validation loss after each query used to answer fully connected model on bank-marketing dataset.

Given the results in this section, we can see that the SmartCache was able to optimize the use of the privacy budget when answering gradient queries and statistical queries for some thresholds. Unfortunately, the results didn't translate as well for different dataset types, but with more careful hyperparameter tuning we believe it is possible to improve these results. The second goal of the SmartCache was to maintain good accuracy when answering queries. This goal was achieved when answering both gradient queries, as evidenced by the low validation error of the model trained using the SmartCache, and the low absolute error of the statistical queries answered using the SmartCache.

Chapter 11

SmartCache Related Work

Optimizing the use of the privacy budget is an active research area in differential privacy. Without good mechanisms to control the amount of budget spent by successive queries, users may be blocked from issuing more queries. Therefore, maximizing the use of the privacy budget is important to make a system more usable. Techniques such as the multiplicative weights mechanism and the sparse vector mechanism have provided interesting ways to improve the use of the privacy budget and influenced the design of the SmartCache. Nonetheless, in SmartCache we have taken some of these ideas that work well in theory and in some restricted scenarios and made them more practical.

11.1 Multiplicative Weights

The multiplicative weights mechanism [20] allows for a large number of interactive counting or linear queries to be answered. The multiplicative weights mechanism views databases as distributions over the data universe. Each item in the database is associated with a positive weight, where the initial weights are a uniform distribution where each weight is $1/N$. When the t-th query q_t arrives a noisy answer is computed by adding Laplace noise to the true answer. The answer from round t is compared to the answer from round $t-1$. If the answers between the rounds are close then the weights are kept the same as in the previous round. If the answers are far from each other the weights on the dataset are reweighted by using multiplicative reweighting, so that the new updated database is closer to an accurate answer. This method is similar to what we do in the SmartCache. In fact, it is the inspiration to the weighted dataset approach used by the SmartCache. Nonetheless, the multiplicative weights mechanism deals with only linear queries, while the SmartCache is also able to deal with non-linear queries such as returning update gradients for neural networks. Another difference is that the SmartCache is meant to support a varied number of queries, that more closely resembles the workload of a real data analyst. This led to the implementation of the decision mechanism and the quality check mechanism so that we can guarantee good accuracy and better utilize the privacy budget.

More recent work has shown that the multiplicative weights mechanism can also be used to answer linear convex minimization queries [38]. This work has shown that multiplicative weights can be used to answer a large number of queries used to train a machine learning model as long as the optimization problem is convex. This differs from the SmartCache approach where we are also able to answer queries to train non-linear models with non-convex minimization, such as with neural networks.

11.2 Sparse Vector Technique

The sparse vector technique [16] [23] satisfies differential privacy and allows a person to output some queries without an added privacy cost. The sparse vector technique works by setting a certain threshold T and outputing whether an answer from a query is above or below the threshold. The threshold has noise added to it and each query result is compared to this perturbed threshold. If a query result is below the threshold no budget is consumed. The privacy budget is only deducted when the outcome of a query is above the threshold. This means that as long as queries are not above the threshold, queries can still be answered with no privacy risk. This mechanism is useful in settings where results above the threshold are sparse. For example, if users only care about results that are above the threshold and most queries issued by them are below the threshold, the sparse vector technique becomes a good alternative to save the privacy budget.

When compared to the SmartCache, the sparse vector technique is another method to help privacy budget preservation. In fact, it can be slightly modified and used in the quality check to decide whether to release a query from the real or synthetic dataset. Nonetheless, the sparse vector technique would require a hit-rate of above 50% for any budget gains, since there is a cost to renoise the threshold when a query is above it, thus doubling the amount of privacy budget necessary for the system. In other words, if query results that are above the threshold are not truly sparse, the sparse vector mechanism is not a good choice. This issue is mitigated in the SmartCache by using the decision mechanism to reduce the amount of times we use the SmartCache and have a miss.

11.3 Other Budget Optimizing Systems

The above techniques have been adopted by various systems with the purpose of optimizing the use of the privacy budget. Honeycrisp [34] and Orchard [35] use the sparse vector technique to increase the number of times a query can be answered before a budget runs out. Nonetheless, they do not focus on strict budget optimization and thus lack features in the SmartCache such as the weighted synthetic dataset and decision mechanism to help decide when it is viable to use the sparse vector technique.

Other data analytics systems such as Airavat [9] for a MapReduce interface and PINQ [26] for SQL like queries exist. Despite the strong and mature querying capabilities of these systems, they do not have the privacy budget optimization focus of the SmartCache. Nonetheless, many of the techniques used in the SmartCache can be integrated into other data analytics systems.

Chapter 12

Conclusion

This thesis presented two systems: LEAP and the SmartCache. LEAP solves the problem of performing data analytics on multiple sites. There is substantial overhead to using LEAP (up to 2.5X slower when training a model such as Resnet-18), but it provide benefits when data sharing. Some of these benefits are control over data (e.g revoking access), facilitating data agreements, and time saved not centralizing data. These benefits are in large part achieved by the federated nature of the system: raw data does not need to be centralized to answer queries.

The SmartCache, on the other hand, solves the issue of optimizing the privacy budget. It allows users to perform more queries before a dataset is at risk of leaking private information. The SmartCache was able to train models to the same validation loss as a DP baseline on the real dataset and consumed close to 70% less of the privacy budget when training a fully connected model on the avocado dataset. For statistical queries, results varied depending on the thresholds used. For thresholds of 0.99 and 0.999 the SmartCache consumed approximately 30% and 50% less of the budget for the same amount of queries when compared to querying the real dataset with DP.

Despite both systems standing on their own, they can also interact. LEAP, for example, can issue DP queries when dealing with sensitive datasets. In this instance, the SmartCache can be used on each site to answer queries from a weight tuned synthetic dataset, and thus allowing each site to answer more queries without putting individuals present in the datasets at risk.

Bibliography

[1] URL https://www.distributedgenomics.ca/. → page 30

[2] URL https://www.datashield.org/. → page 31

[3] grpc. URL https://grpc.io/. → page 18

[4] Jwt.io. URL https://jwt.io/. → page 21

[5] Protocol buffers. URL https://developers.google.com/protocol-buffers. → page 18

[6] Redcap. URL https://www.project-redcap.org/. → page 11

[7] Aug 2021. URL https://www.openspecimen.org/?__cf_chl_jschl_tk__=pmd_024dcf3e278a0ae7fbabeca869d673b0eb0a7fb2-1629177294-0-gqNtZGzNAc2jcnBszQhi. → page 11

[8] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi:10.1145/2976749.2978318. URL https://doi.org/10.1145/2976749.2978318. → page 2

[9] I. R. S. T. S. Ann and K. V. S. E. Witchel. Airavat: Security and privacy for mapreduce. In *Proc. Usenix Org*, pages 297–312, 2011. → page 60

[10] S. Aydore, W. Brown, M. Kearns, K. Kenthapadi, L. Melis, A. Roth, and A. Siva. Differentially private query release through adaptive projection, 2021. → page 3

[11] H. A. Board. Avocado prices, 06 2018. URL https://www.kaggle.com/neuromusic/avocado-prices. → page 47

[12] J. Dong, A. Roth, and W. J. Su. Gaussian differential privacy. *arXiv preprint arXiv:1905.02383*, 2019. → page 2

[13] C. Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation””TAMC*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer Verlag, April 2008. URL https://www.microsoft.com/en-us/research/publication/differential-privacy-a-survey-of-results/. → pages 2, 6

[14] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology (EUROCRYPT 2006)*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer Verlag, May 2006. URL https://www.microsoft.com/en-us/research/publication/our-data-ourselves-privacy-via-distributed-noise-generation/. → page 8

[15] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-32732-5. → pages 2, 6

[16] C. Dwork, M. Naor, O. Reingold, G. N. Rothblum, and S. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 381–390, 2009. → page 59

[17] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60, 2010. doi:10.1109/FOCS.2010.12. → page 2

[18] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4): 211–407, 2014. → page 2

[19] K. Emam, D. Buckeridge, R. Tamblyn, A. Neisa, E. Jonker, and A. Verma. The re-identification risk of canadians from longitudinal demographics. *BMC medical informatics and decision making*, 11:46, 06 2011. doi:10.1186/1472-6947-11-46. → page 2

[20] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, 2010. → pages 3, 58

[21] Z. He, T. Zhang, and R. B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 148–162, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450376280. doi:10.1145/3359789.3359824. URL https://doi.org/10.1145/3359789.3359824. → page 2

[22] P. Kairouz, S. Oh, and P. Viswanath. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6): 4037–4049, 2017. doi:10.1109/TIT.2017.2685505. → pages 2, 8

[23] M. Lyu, D. Su, and N. Li. Understanding the sparse vector technique for differential privacy, 2016. → page 59

[24] M. Lécuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu. Privacy accounting and quality control in the sage differentially private ml platform. *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, Oct 2019. doi:10.1145/3341301.3359639. URL http://dx.doi.org/10.1145/3341301.3359639. → pages 2, 3

[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017. → page 5

[26] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009. → page 60

[27] I. Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, 2017. → page 3

[28] S. Moro, P. Cortez, and P. Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014. ISSN 0167-9236. doi:https://doi.org/10.1016/j.dss.2014.03.001. URL https://www.sciencedirect.com/science/article/pii/S016792361400061X. → page 47

[29] T. Murakami and Y. Kawamoto. Utility-optimized local differential privacy mechanisms for distribution estimation. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1877–1894, Santa Clara, CA, Aug. 2019. USENIX Association. ISBN 978-1-939133-06-9. URL https:

//www.usenix.org/conference/usenixsecurity19/presentation/murakami. → page 3

[30] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125, 2008. → page 2

[31] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753, 2019. doi:10.1109/SP.2019.00065. → page 2

[32] S. M. Plis, A. D. Sarwate, D. Wood, C. Dieringer, D. Landis, C. Reed, S. R. Panta, J. A. Turner, J. M. Shoemaker, K. W. Carter, P. Thompson, K. Hutchison, and V. D. Calhoun. Coinstac: A privacy enabled model and prototype for leveraging and processing decentralized brain imaging data. *Frontiers in Neuroscience*, 10:365, 2016. ISSN 1662-453X. doi:10.3389/fnins.2016.00365. URL https://www.frontiersin.org/article/10.3389/fnins.2016.00365. → page 31

[33] E. Rescorla. tls, Aug 2018. URL https://datatracker.ietf.org/doc/html/rfc8446. → page 19

[34] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 196–210, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368735. doi:10.1145/3341301.3359660. URL https://doi.org/10.1145/3341301.3359660. → pages 2, 60

[35] E. Roth, H. Zhang, A. Haeberlen, and B. C. Pierce. Orchard: Differentially private analytics at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1065–1081. USENIX Association, Nov. 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/roth. → pages 2, 60

[36] R. Shokri, M. Stronati, and V. Shmatikov. Membership inference attacks against machine learning models. *CoRR*, abs/1610.05820, 2016. URL http://arxiv.org/abs/1610.05820. → page 2

[37] P. Tschandl, C. Rosendahl, and H. Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5(1):1–9, 2018. → page 23

[38] J. Ullman. Private multiplicative weights beyond linear queries, 2015. → page 59

[39] W. G. Van Panhuis, P. Paul, C. Emerson, J. Grefenstette, R. Wilder, A. J. Herbst, D. Heymann, and D. S. Burke. A systematic review of barriers to data sharing in public health. *BMC public health*, 14(1):1–9, 2014. → page 1

[40] Z. Yang, J. Zhang, E.-C. Chang, and Z. Liang. Neural network inversion in adversarial setting via background knowledge alignment. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 225–240, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367479. doi:10.1145/3319535.3354261. URL https://doi.org/10.1145/3319535.3354261. → page 2

[41] D. Zhang, R. McKenna, I. Kotsogiannis, G. Bissias, M. Hay, A. Machanavajjhala, and G. Miklau. ektelo: A framework for defining differentially private computations. *ACM Trans. Database Syst.*, 45(1), Feb. 2020. ISSN 0362-5915. doi:10.1145/3362032. URL https://doi.org/10.1145/3362032. → page 2