Tiresias: A Peer-to-Peer Platform for Privacy Preserving Machine Learning

by

Kevin Zhang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2020

Author . . . .
Department of Electrical Engineering and Computer Science
January 17, 2020

Certified by . . . .
Kalyan Veeramachaneni
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Tiresias: A Peer-to-Peer Platform for Privacy Preserving Machine Learning

by

Kevin Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on January 17, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Big technology firms have a monopoly over user data. To remediate this, we propose a data science platform which allows users to collect their personal data and offer computations on them in a differentially private manner. This platform provides a mechanism for contributors to offer computations on their data in a privacy-preserving way and for requesters — i.e. anyone who can benefit from applying machine learning to the users' data — to request computations on user data they would otherwise not be able to collect. Through carefully designed differential privacy mechanisms, we can create a platform which gives people control over their data and enables new types of applications.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist

Acknowledgments

First, I would like to thank my supervisor, Kalyan, for his support, guidance, and mentorship throughout this project. His insightful feedback and ideas were invaluable for this thesis and his enthusiasm for tackling novel, ambitious, and open-ended problems made this project possible. I would also like to thank Carles, Santu, Felipe, Dongyu, Ivan, and Arash for their help with the design of the system, development of the open source libraries, and creation of the figures and diagrams in this work. Finally, I would like to thank my family for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The twenty-first century has seen massive growth in machine learning applications across various domains. However, machine learning is still only feasible for use in production systems by tech giants and large corporations that have access to huge amounts of proprietary data due to their large user bases [4]. Small entities like start-ups and research communities, as well as individuals, often find it difficult to develop and deploy novel machine learning models due to the scarcity of relevant training data. Despite the fact that machine learning provides immense potential for predictive analysis across various domains, the lack of of relevant data, as well as the absence of a platform that can facilitate systematic gathering of such data, serve as major barriers to entry. In this work, we propose a platform to address these issues.

1.1 Personal User Data

1.1.1 Why does personal user data matter?

Personal user data has immense economic value, as demonstrated by the $558 billion dollar online advertising industry which relies heavily on behavioral targeting powered by big data and machine learning [37]. Examples of personal user data range from an individual's physical location history and in-person purchases at stores to their web browsing history and online purchases.

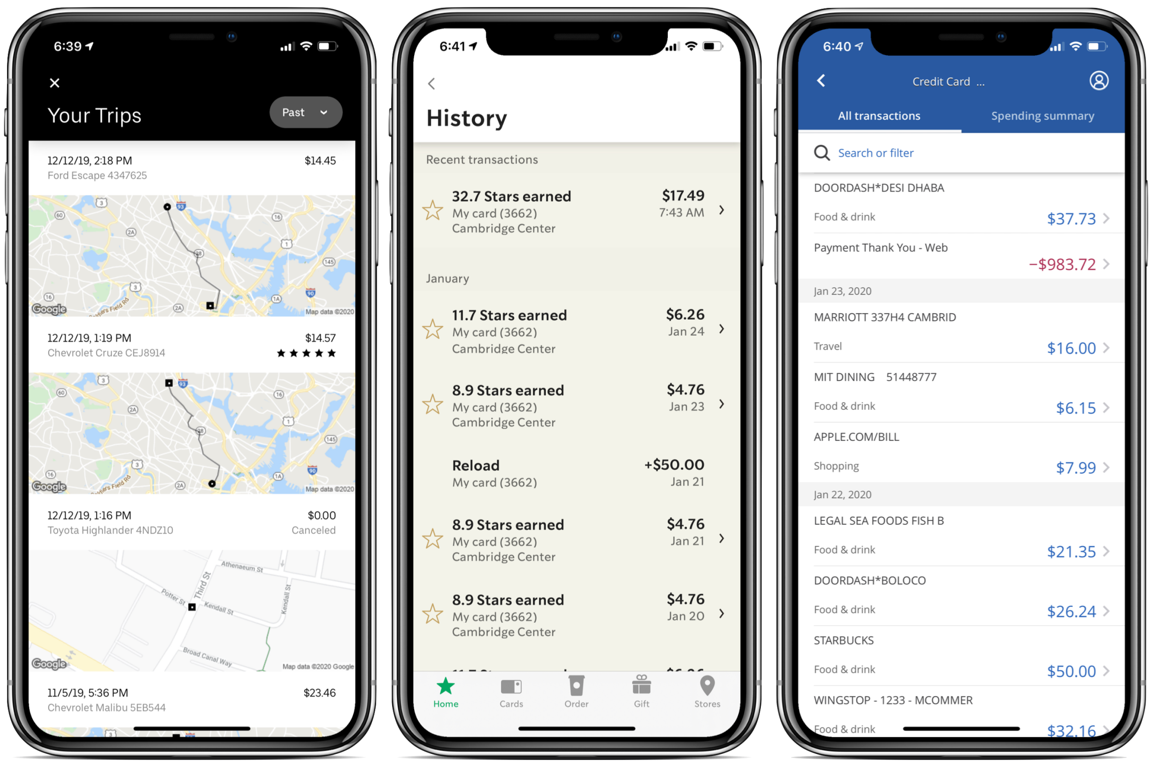
Large companies are able to use this data to deliver better products and services. For example, by aggregating user location data, companies can now offer services ranging from restaurant waiting time calculators, like what Google Maps provides, to navigational software apps with traffic prediction such as Waze. Other applications for personal user data include making recommendations for streaming services, as happens on Spotify or Netflix, detecting fraud in financial transactions by institutions such as Paypal and Bank of America, and even early detection of medical conditions by insurance companies.

We observe that in many of these cases, there are no technological barriers preventing individuals from doing the same thing in a peer-to-peer manner, merely a lack of access to sufficient training data. This is a common theme across a variety of scenarios involving individual personal data: although a group of individuals may collectively have access to all the personal data needed to solve a problem, they are unable to make use of it without having to rely on a third party due to the challenges involved in collecting, processing, and aggregating it.

1.1.2 What data is available to individuals?

A plethora of personal user data is available for individuals to capture and collect. The first type is data that has already been preprocessed by external applications such as Uber, Starbucks, and Chase, as shown in Figure 1-1. Individuals are allowed to export the data from these applications and use it for their personal purposes.

The second type of data that can be captured involves an active collection process. Rather than extracting features from an existing dataset, individuals who want to obtain this data must install collector applications that actively record information about their actions. Examples of data collection applications which have an active component range from a Chrome browser extension which tracks internet usage to a desktop application which monitors the amount of screen time spent on different software programs. These applications store their observations locally on the user's device; only the users themselves are able to view this data, and they choose what to do with it.


6:39

Your Trips

Past

12/12/19, 2:18 PM $14.45
Ford Escape 4347625

12/12/19, 1:19 PM $14.57
Chevrolet Cruze CEJ8914 ★★★★★

12/12/19, 1:16 PM $0.00
Toyota Highlander 4NDZ10 Canceled

11/5/19, 5:36 PM $23.46
Chevrolet Malibu 5EB544

6:41

History

Recent transactions

32.7 Stars earned $17.49
My card (3662) 7:43 AM
Cambridge Center

January

11.7 Stars earned $6.26
My card (3662) Jan 24
Cambridge Center

8.9 Stars earned $4.76
My card (3662) Jan 23
Cambridge Center

Reload +$50.00
My card (3662) Jan 21

8.9 Stars earned $4.76
My card (3662) Jan 21
Cambridge Center

8.9 Stars earned $4.76
My card (3662) Jan 20
Cambridge Center

Home Cards Order Gift Stores

6:40

Credit Card ...

All transactions Spending summary

Search or filter

DOORDASH*DESI DHABA
Food & drink $37.73

Payment Thank You - Web
−$983.72

Jan 23, 2020

MARRIOTT 337H4 CAMBRID
Travel $16.00

MIT DINING 51448777
Food & drink $6.15

APPLE.COM/BILL
Shopping $7.99

Jan 22, 2020

LEGAL SEA FOODS FISH B
Food & drink $21.35

DOORDASH*BOLOCO
Food & drink $26.24

STARBUCKS
Food & drink $50.00

WINGSTOP - 1233 - MCOMMER
Food & drink $32.16

Figure 1-1: Examples of personal user data from Uber, Starbucks, and Chase that are available for individuals to capture and collect. Uber shows your trip history including the time, price, and route. Starbucks shows the time, price, and location. Chase shows the time, price, and vendor.

We envision a system which supports both of these types of data collection and allows individuals to collect personal data across a variety of domains — ranging from their Starbucks order history to their web browsing traffic — and store it locally on their device as described in Section 3.1. At any point in time, a group of individuals would be able to opt into using a differentially private mechanism to contribute a subset of their data towards solving a problem of interest.

1.2 The Tiresias Platform

How can we make machine learning technology more accessible to smaller entities as well as to individuals? In other words, can we build a general platform that facilitates peer-to-peer machine learning? Because machine learning models are data-hungry,

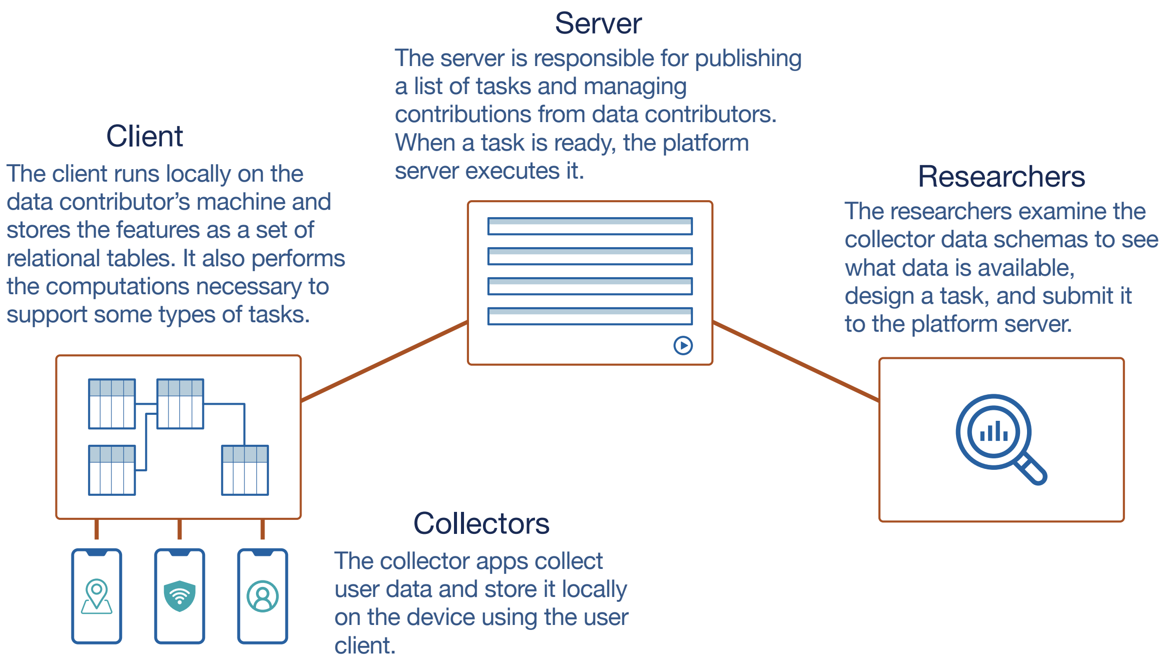

Server
The server is responsible for publishing a list of tasks and managing contributions from data contributors. When a task is ready, the platform server executes it.
Client
The client runs locally on the data contributor's machine and stores the features as a set of relational tables. It also performs the computations necessary to support some types of tasks.
Researchers
The researchers examine the collector data schemas to see what data is available, design a task, and submit it to the platform server.
Collectors
The collector apps collect user data and store it locally on the device using the user client.

Figure 1-2: Interactions between the collectors, client, server, and researchers on the Tiresias platform. The design of the collectors, client, and server is described in Section 3.3.

the platform must provide a way to securely share personal data among multiple untrusted users. More importantly, in this case, the platform itself is not trusted, because we want any end user to be able to initiate and host the machine learning process. How can we create a secure peer-to-peer machine learning platform that is hosted through an untrusted server, yet powerful enough to build useful data science applications?

In this thesis, we propose **Tiresias**, an open yet secure data science platform designed to enable peer-to-peer machine learning. Tiresias would allow **requestors** to write machine learning tasks that can be used to solve a wide variety of problems while enabling **data contributors** to collect and contribute data to those tasks. We argue that the design of such a platform is highly challenging for three reasons:

1. **Machine Learning:** The primary focus of the platform is to enable peer-to-peer machine learning. Thus, the system should provide a unified platform to gather training data that can support a wide variety of machine learning tasks.

This dimension includes various subtopics, from data collection and feature extraction to model selection and optimization.

2. **Privacy:** Privacy is a nebulous concept — it has been characterized as everything from the desire for personal space and freedom to the desire to avoid price discrimination or other, more insidious forms of discrimination [2]. There is significant evidence of shifting attitudes towards privacy and a growing backlash against privacy-violating firms, suggesting that stronger privacy protections may be a necessary component of future data-related endeavors [4]. Most importantly, privacy concerns often make end users skeptical about the repercussions of sharing their private data. For these reasons, the platform should provide a privacy-preserving way to share personal data.

3. **Systems:** Finally, the platform itself has a systems aspect which contains its own challenges, such as efficiency, robustness and resource management.

We understand that a comprehensive study of all three areas is beyond the scope of a single thesis, and primarily focus on two dimensions in this work: Privacy and Systems. For Privacy, we propose a decentralized data science platform that protects consumers through local differential privacy mechanisms. In the Systems area, we propose a new architecture for the platform and explain our current design choices in detail.[1]

The benefits of such a platform are manifold. First, the platform provides a unique locus for combining datasets across domains (e.g. we can merge data on web browsing from a Chrome extension with fitness/health data from a mobile app at an individual level). Second, this system allows users to retain ownership over their personal data. Users can also preserve their data for future use, even if they don't want to share it now. Finally, the platform democratizes access to the benefits of machine learning by allowing individuals to contribute to machine learning projects — potentially in exchange for economic incentives — which we view as a paradigm shift that will

[1]This is ongoing work. As this project develops, we expect our design choices to evolve. The reader can find the latest on the Tiresias platform at `https://github.com/DAI-Lab/Tiresias`.

characterize the next generation of machine learning.

1.3 Motivating Scenarios

In this section, we explore some common scenarios in which people could use the Tiresias platform. Each of these scenarios is presented from the point of view of the entity that wants to perform a particular data-driven task, and we provide a high-level description of what the platform does behind the scenes to execute it.

1.3.1 Peer-to-Peer Machine Learning

In the first scenario, a group of friends is attempting to build a machine learning model together and train it on their own personal data. They want to make sure they do this in a privacy-preserving manner. The Tiresias platform allows them to achieve this goal by (1) providing a client that allows them to collect and store their personal data and (2) providing a server library that, in conjunction with the client library, provides support for a variety of differentially private machine learning algorithms.

Specifically, each participant would simply need to install the client and start collecting their data by installing whatever collector application suits their particular use case (e.g. a geolocation collector which tracks their location history). Then, they could work together to write a task that trains a differentially private logistic regression model to predict a quantity of interest such as their commute time.

To the best of our knowledge, Tiresias is the first platform which enables individuals to perform this kind of differentially private collaborative machine learning, opening up possibilities for all kinds of new applications.

1.3.2 Supporting Research

In the next scenario, a researcher at an academic institution is attempting to gain insight into some societal trends. For example, consider a scenario where the researcher wishes to understand the relationship between geographic location and web browsing. The Tiresias platform enables this research by providing a personal data store for people to collect this data, and providing differentially private mechanisms for the researcher to securely interact with the data (e.g. estimating causal impacts using a difference-in-differences regression).

The key feature of the Tiresias platform is that it makes this process easier for both the users and the researcher. By providing a personal data store, the Tiresias platform allows users to passively collect data about themselves and store it locally; they then have the freedom to choose whether to provide private access to the data on a case-by-case basis (e.g. if another researcher wants to run a similar experiment but has a better reputation, the contributor can choose to deny the first request). On the researcher side, the platform allows them to work more efficiently, as they no longer have to collect the data themselves and can focus on modeling and analysis.

1.3.3 Transfer Learning

Another scenario where the Tiresias platform presents a compelling solution is model fine-tuning. There are many practical image and natural language processing tasks for which domain experts have released pre-trained models, from image classification to machine translation, but these models often need to be fine-tuned if they are to be applied to a specific problem. The Tiresias platform enables this by allowing researchers to start from a pre-trained model and fine-tune it using private user data, allowing the model to learn from data that is more representative of the real world than the curated training set that it was originally constructed on.

To implement this, a researcher would start by submitting a **gradient task** containing a set of pre-trained model weights and ask the users to compute the gradients with respect to the weights using their private data, as described in Section 4. Differential privacy mechanisms are applied to the gradients on the client side before they are sent to the platform for aggregation and applied to the pre-trained model. Previous work has shown that it can be difficult to train a neural network from scratch within a reasonable privacy budget; however, fine-tuning a model that has already been trained is significantly easier and can be accomplished even with a modest privacy budget [1].

Our platform makes it easy to apply these transfer learning techniques to private user data, especially for image classification problems, by providing multiple pre-trained convolutional neural networks that can be retrained using **gradient tasks**.

1.3.4 Institutional Data Collection

Finally, we see an opportunity for our platform to support institutional data collection. Everyone from academic institutions to Fortune 500 companies collects data about their employees, students, and staff to track everything from the usage of office space to the amount of time spent on different tasks. This type of data collection is intrusive and has been widely criticized; however, it is still used by various institutions since it allows administrations to better monitor resource usage and optimize their internal performance.

Our system makes the process less intrusive and more equitable by giving users control over their data. Specifically, we believe that our system will:

1. Make it easier for the parties to implement data collection mechanisms.

2. Provide a flexible set of tools for machine learning and data science.

3. Allow individuals to manage requests for data on a case-by-case basis.

We see an opportunity for our platform to make institutional data collection opt-in rather than mandatory. Ideally, this would allow institutions to collect the same data that they already do in a more open and transparent manner, while simultaneously giving individuals more control over what their data is used for.

1.4 Enabling Technologies

We argue that now is the ideal moment for building a platform that enables personal data collection and differentially private machine learning for four reasons:

1. **Availability of user data.** First, this work comes at a time when organizations are moving towards giving individuals more access to their own data for a variety of reasons, ranging from recent privacy legislation such as GDPR and CCPA to competition with other firms. (For example, users can now access their rideshare history through Lyft, their order history from Starbucks, and search history on Google.) This, in combination with the ease of deploying software, has made it possible to build data collection applications which operate at the user level and allow individuals to create their own personal data stores.

2. **Open source software.** Second, the prevalence of open source machine learning libraries, from scikit-learn to PyTorch [34], has made it possible for individuals to design, train, and deploy machine learning models. In contrast to other technologies such as search engines, machine learning research puts a heavy premium on open-source code, resulting in a rich ecosystem of libraries that contain cutting edge research.

3. **Automated machine learning pipelines.** Third, work on automated machine learning has advanced to a stage where even non-domain experts can build competitive machine learning models through techniques such as automated feature extraction [26] and hyperparameter optimization [20].

4. **Differential privacy research.** Finally, privacy research has advanced by leaps and bounds in the past decade. Concepts such as differential privacy [15], local differential privacy [7], and federated learning [1] provide formal mathematical guarantees of user privacy. This body of work makes it possible for individuals to perform computations on their combined data in a way that minimizes their individual risk.

The combination of these factors makes it possible to (1) build a system for securely and locally collecting individual user data, (2) design and implement complex machine learning pipelines with ease, and (3) execute machine learning procedures on individual user data while offering differential privacy guarantees.

1.5 Thesis Roadmap

This thesis is organized as follows:

- Chapter 2 introduces related work in privacy, machine learning, and systems.
- Chapter 3 presents the high-level design of our system.
- Chapter 4 explains the types of tasks supported by our system.
- Chapter 5 demonstrates the performance of our system.
- Chapter 6 proposes user studies to demonstrate the usability of our system.
- Chapter 7 highlights promising areas for future work on this system.

Chapter 2

Related Work

In this section, we provide an overview of key results in differential privacy, some differentially private mechanisms and models that have been explored, and highlight some open source software and systems which provide implementations of differential privacy mechanisms.

2.1 Differential Privacy

Differential privacy is a statistical technique proposed by Dwork and McSherry [14] to measure the theoretical maximum privacy loss that may be incurred if a statistic derived from a dataset is released. This framework allows us to move away from treating privacy as a binary concept and towards treating it as a continuum. As defined in [12], a mechanism M is (ϵ, δ)-differentially private if it satisfies the following condition for all subsets of possible values $S \subseteq Range(M)$ and databases x and y that differ by a single row:

$$P(M(x) \in S) \leq e^{\epsilon} P(M(y) \in S) + \delta$$

We note that multiple variations on ϵ-differential privacy have been proposed [15]; the above (ϵ, δ)-differential privacy definition is what we will primarily use in this work. Across all of these definitions, the ϵ parameter corresponds to the maximum privacy loss that may be incurred. Unfortunately, the interpretation of ϵ varies across datasets and domains and automatically choosing the appropriate ϵ value for a particular dataset remains an open problem [28].

The four typical settings for applying differential privacy are (1) the *local differential privacy* setting where users use randomized response to add noise to their own data prior to contributing it, (2) the *centralized aggregator* model where a central authority is responsible for collecting the raw data and then making the result differentially private, (3) the *interactive* setting where multiple queries are sequentially submitted to a central authority, and (4) the *public disclosure* setting where the differentially private results are released to a public audience [29]. Our system is a hybrid of the first three settings, giving data collectors and requestors the flexibility to select the appropriate setting for their particular use case.

2.2 Mechanisms and Models

General strategies for achieving differential privacy include (1) randomized response-based methods (a.k.a. local differential privacy) which adds noise to each individual's data as described in [25], (2) the propose-test-release framework which attempts to iteratively identify the sensitivity of a function [13], and (3) the sample-and-aggregate framework which uses sampling to bound the contribution of each user to the final output [32]. These general strategies, in combination with basic building blocks such as the Laplace mechanism and exponential mechanism, provide a rich toolkit for designing differentially private algorithms for everything from summary statistics and binary search to machine learning and combinatorial optimization.

Next, we select some works which lie at the intersection of machine learning and differential privacy. These papers typically start with an existing machine learning algorithm, modify it by adding noise to the parameters or perturbing the objective function, and prove that the resulting model is differentially private. A few popular examples of machine learning models for which differentially private versions have been derived include linear regression [35], logistic regression [6], Gaussian naive bayes

[38], decision trees [23], and neural networks [1].

We note that the majority of the works described in this section focus primarily on theoretical proofs of algorithms that can be used to achieve differential privacy rather than providing practical software implementations.

2.3 Systems and Software

Finally, we highlight some recent works which aim to make differential privacy practical for real-world applications by releasing open source software and systems.

One of the earliest works that aims to make differential privacy easy to use is [31], which presents a LINQ-like language for writing queries that automatically provide differential privacy; however, it imposes significant constraints on the types of queries that can be executed. Another work on making queries differentially private is [24], which proposes an algorithm for automatically rewriting SQL queries to be differentially private.

Towards practical differentially private machine learning, the authors of [30] provide practical advice about how to implement differential privacy for iterative (e.g. gradient based) training procedures and implement their ideas in the open source Tensorflow Privacy library that enables users to train differentially private neural networks. The authors of [8] take a orthogonal approach and propose training models in a non-private way but offering differentially private predictions as a service.

Other open-source projects that aim to make differential privacy tools more accessible include Google's RAPPOR [16] which provides tools to collect statistics about categorical values, IBM's DiffPrivLib [21] which provides implementations of differentially private machine learning models, and Google's Differential Privacy project [39] which provides an optimized C++ library for computing various statistics in a differentially private manner.

We note that although the works described in this section provide open source software implementations of their algorithms, they tend to be focused on a specific problem as opposed to providing an end-to-end platform for differential privacy.

Chapter 3

System Architecture

In this section, we describe (1) the *collectors*, which are responsible for capturing user data, (2) the different stakeholders who interact with Tiresias, and (3) the software architecture of the platform. At a high level, our system consists of the *server*, which provides an interface for requesting, managing, and executing tasks, the *client*, which is responsible for storing user data locally and performing the necessary client-side computations, and a set of *collectors*, which are responsible for gathering user data and extracting relevant features, as shown in Figure 1-2.

Individuals who are interested in running *tasks* on the user data — a.k.a. *requestors* — can inspect the predefined data schemas associated with each *collector* and use this information to design and write tasks, as described in Section 4. The *requestor* can then submit the task to the *server* which will allow *data contributors* to explore all open tasks and choose which ones to accept. Once enough *data contributors* have accepted the task, the task is executed and the differentially private results are made available to the original *requestor*.

3.1 Collectors

The *collectors* are responsible for generating clean relational datasets locally on the *client*. For example, a web browsing collector would track the internet history of the user, extract relevant fields, and store them as a set of tables. Other examples of *collectors* that could be implemented include scripts that extract data from external applications such as *Uber*, *Starbucks*, and *Chase*, as well as applications that monitor specific activities such as browsing history, application usage, or even physical location.

Each *collector* is expected to interface with the *client* through a standard interface which requires them to (1) declare a fixed data schema that describes the dataset that it will be creating and (2) insert data into the *client*. The data insertion can either be user-triggered (i.e. a script that extracts the Lyft ride history whenever the user runs it) or periodic (i.e. an application that reports the user's IP address at one-hour intervals).

3.1.1 Collector Usage

When the *data contributor* installs the *client*, they can choose which of the available set of *collectors* to install. Before enabling a *collector*, the user can inspect the publicly declared data schema (or even the open-source code itself) to see what type of data will be collected and stored locally. As shown in Figure 3-1, these *collectors* work together to populate the user's personal data store with useful features.

Note that different users can choose different sets of *collectors* to enable depending on their personal preferences, allowing users to customize their experiences. Some users may be reassured by the fact that all of the data is stored locally on their personal device, and choose to simply enable all the available *collectors* to capture and organize as much data as possible. Others may choose to install a subset of *collectors* so as to only capture data that is of particular interest to them.

3.1.2 Collector Development

When the *collector* is first created, it must declare a static set of relational tables (i.e. their data schema). These tables are stored and maintained by the *client* application, which also exposes an interface to allow a *requestor* to further process the data as part of a task. Any software developer who is interested in contributing to the Tiresias platform can build a *collector* that runs on the user's device upon installation, collects data about a specific domain (i.e. browsing history), and extracts useful features. They can then submit a pull request to the Tiresias repository in order to publish their *collector*. Note that all *collectors* must be open source and must make their output data schema publicly available for inspection.

3.2 Stakeholders

In this section, we describe the two primary stakeholders in our system: (1) the *data contributors* who contribute their data and (2) the *requestors* who submit the *tasks* that are computed on this data. The *data contributors* and *requestors* interact with a client-server architecture which we describe in Section 3.3.

3.2.1 Data Contributors

Contributors are users who choose to pre-process and store their personal data *locally* using the Tiresias client. By default, each *data contributor* maintains complete ownership over their own data, as it is stored locally on their own devices; at any point in time, however, a *contributor* can choose to contribute their data to open tasks on the Tiresias platform in a differentially private manner.

In order to become a *data contributor*, a user needs only to do the following:

1. Install the Tiresias client.
2. Choose, install, and run *collectors*.
3. Browse open tasks on Tiresias *server*.
4. Select tasks to privately contribute data to.

The *server* provides a listing of open tasks that users can contribute their data to, and the *data contributor* is able to browse these tasks and choose which ones to accept. Once the *data contributor* accepts a task, the client and server work together to perform the appropriate computations depending on the task type, modeling algorithm, and differential privacy configuration.

3.2.2 Data Requestors

Requestors are users who are interested in creating tasks which require computation on user data. Examples of tasks range from evaluating statistical queries (e.g. identifying the average age of visitors to a particular website) to training machine learning models (e.g. predicting whether a pull request will be accepted).

To become a *requestor*, a user simply needs to make an API call to the server to create a new task. In general, a *requestor* will take the following steps:

1. Install the Tiresias library.

2. Explore the publicly available data schemas (Section 3.1).

3. Design a task by writing a JSON object (Section 4).

4. Submit the task to the server.

Once enough *data contributors* have accepted the task, the *requestor* can then call the API again to retrieve the output of their task, whether that is summary statistics or a fully-trained model.

3.3 System Design

In this section, we introduce the design of our server and client. The *server* is hosted by a service provider and is responsible for performing some of the necessary computations to support differential privacy, statistics, and machine learning, as well as providing API endpoints to allow *requestors* to create tasks and contributors to submit data. The *client* runs on the *data contributor's* device and is responsible for collecting data, extracting features into relational tables, and communicating with the *server* in order to accept and process tasks.

3.3.1 Server

The *server* is responsible for coordinating interactions between *data contributors* and *requestors*. It maintains a list of open tasks that have been submitted by *requestors* for processing as well as a list of data schemas that correspond to the data that is being collected. The *data contributors* can look at the open tasks and select which tasks to accept, while the *requestors* can look at the data schemas in order to compose and submit tasks.

Depending on the type of task, the *server* is also responsible for performing some of the computation necessary to support differential privacy, train a machine learning model, compute the appropriate statistics, and so on. We discuss the computational aspects of the *server* in greater detail in Section 4.

The *server* interacts with the *data contributors* and *requestors* through the following REST endpoints:

- `GET /api/task`

  This endpoint returns a list of all open tasks. Each task is represented as a JSON object which contains information about the data that is being requested, how the data will be processed, and what it will be used for.

- `POST /api/task`

  This endpoint allows requestors to create a new task. The task is represented as a JSON object as described in Section 4 and the endpoint returns a `task id`.

- `GET /api/task/<task_id>`

  This endpoint allows requestors to check on the status of their tasks. If the task has been completed, the results field contains the output of the task, which can range from a scalar value to a trained machine learning model.

- `POST /api/task/<task_id>/submit`

  This endpoint allows contributors to submit their data to a specific task.

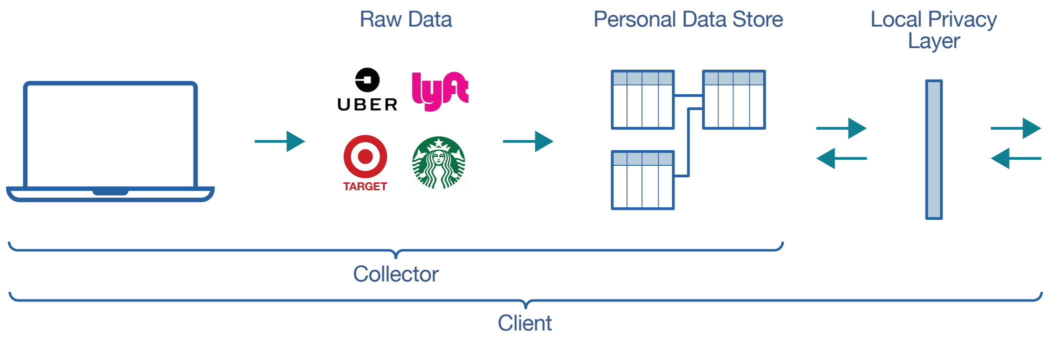


Figure 3-1: The client transforms raw data collected from the user's device into a personal data store. Then, if the user accepts a task, it retrieves the appropriate data from the personal feature store, applies local privacy mechanisms if appropriate, and transmits the processed data to the server.

When a task is first created, all users are able to see it and choose whether to contribute the requested data and/or features. The task remains in this state until the minimum count threshold is reached (e.g. enough users have chosen to contribute their data), at which point the task is executed and the results are returned to the *requestor*.

This module is implemented using a multi-threaded web server backed by an in-memory object which tracks the status of every task as well as the data associated with it. A separate task handler thread is responsible for identifying tasks that are ready for processing, performing the appropriate computations, and deleting the users' data once the results are ready.

3.3.2 Client

The *client* application runs locally on the user's device and is responsible for securely storing the user's data as well as supporting the differential privacy mechanisms described in Section 4.

Data Collection

A typical *collector* is implemented as an application that collects user data (e.g. location history) and uses the API endpoints shown in Figure 3-2 to store the featurized data within the Tiresias client. For example, when a *collector* such as a browsing history tracker is installed, it calls the first endpoint once to register its schema and then periodically calls the second endpoint to insert data about the user's activity.[1]

Internally, within the client, the data store is organized as a collection of SQLite databases which includes a *metadata* database and one or more *collector* databases. The *metadata* database keeps track of which data collectors are installed and the data schemas for each. The *collector* databases correspond to each data collector that is installed; when the collector inserts data, it is stored in these databases.

When the data is being queried (i.e. when a task is approved), the appropriate set of *collector* databases is attached to the *metadata* database and the query is executed. This design enables requestors to use the JOIN operator to write tasks that operate on data originating from multiple collectors.

Task Management

Using the *client*, the *data contributor* can view the open tasks on the platform and select which tasks to accept through either the Python API or the graphical user interface. As discussed in Section 4, each task is represented by a JSON object which the *data contributor* can inspect to understand what data will be shared, what the data will be used for, and the privacy settings that will be used. This JSON representation is supplemented by an automatically generated natural language description of the task, as shown in Figure B-2. Furthermore, the user can preview the data that will be sent to the platform, as shown in Figure B-4.

Once the *data contributor* has accepted a task, the requested values are retrieved from the user's personal data store and the appropriate local preprocessing steps are performed before the data is sent to the *server*. For example, if the contributor chooses to approve a task that uses local differential privacy, then the *client* applies the appropriate differential privacy mechanisms. For other types of tasks, the client-side computation may be more expensive; in the case of a gradient task, for example,

[1]We emphasize the fact that, although the *collectors* communicate with the *client* using a REST API for simplicity and compatibility, all communication happens over localhost and there is no communication with the outside world during the data collection process.

POST /app/<app_name>/register

This endpoint registers a new data collection application with the given schema.

Request Body:

```
{
    "tableX": {
        "description": "This table contains X.",
        "columns": {
            "column1": {
                "type": "float",
                "description: "This column contains ..."
            },
            "column2": {
                "type": "string",
                "description: "This column contains ..."
            }
        }
    }
    "anotherTable": ...
}
```

POST /api/<app_name>/insert

This endpoint allows the data collection application to insert data into its feature store.

Request Body:

```
{
    "tableX": [
        {"column1": 0.0, "column2": "hello"}
    ],
    "anotherTable": ...
}
```

Figure 3-2: These two REST API endpoints (which are only accessible over localhost) are used by collectors to (1) declare their data schema and (2) insert data into the client.

the *client* computes a perturbed estimate of the gradients for a deep learning model, which can require significant computational power.

Chapter 4

Tasks and Mechanisms

The goal of this section is to provide a high-level overview of the four primary types of tasks (and corresponding differential privacy mechanisms). Note that additional instance-specific tasks are possible and desirable - for example, we may want to provide a highly optimized deep learning task which uses multiple rounds of computation to support the moments accountant [1] - but these general task types are intended to capture the vast majority of use cases.

The first two types of tasks — **basic tasks** and **integrated tasks** — assume that the data contributors trust the server but do not trust the requestors. These methods require that the data contributor send their featurized data to the server which then applies differential privacy mechanisms before releasing the differentially private results to the requestor.

The next two types of tasks — **bounded tasks** and **gradient tasks** — relax the assumptions so that the data contributors do not need to trust the server. For these types of tasks, the data contributors apply local differential privacy mechanisms in the client and send only perturbed data to the server.

4.1 Basic Tasks

This type of task involves gathering a feature or a value from each contributor and computing an aggregate statistic on the server. For this type of task, the differential

Field	**Description**
`epsilon`	This field takes a numerical value greater than 0.0.
`delta`	This field takes a numerical value between 0.0 and 1.0.
`min_count`	This field takes an integer value greater than 10 corresponding to the minimum number of data contributors who must submit data before the task is processed by the server.
`featurizer`	This field is expected to be a string containing a valid SQL query that extracts a feature from the data contributor's personal data store.
`aggregator`	This field is expected to be one of the following aggregation mechanisms: `mean`, `median`, `count`, `sum`, and `variance`.

Table 4.1: This table describes the fields of the JSON object for a basic task.

privacy mechanisms are implemented on the *server*; therefore, the contributor *needs to trust the server* to correctly compute a differentially private estimate of the statistic.

4.1.1 Task Representation

This task, when created by a requestor, is represented as a JSON object with the fields listed in Table 4.1. The `epsilon` and `delta` fields correspond to the privacy parameters in the (ϵ, δ) definition of differential privacy. The `min_count` field corresponds to the minimum number of data contributors who need to accept this task before it is processed.

This class of tasks involves two components, a client-side `featurizer` operation which computes and returns a single scalar value and a server-side `aggregator` operation which belongs to a set of predefined differentially private aggregation functions shown in Table 4.2. We provide a complete example of a basic task which computes the median age in Figure 4-1.

4.1.2 Computation

When processing basic tasks, the *client* is responsible for extracting the appropriate features/variables from the user's personal data store and transmitting it to the *server*. The *server* then estimates the value of the aggregation function in a differentially

Example: Basic Task

```
{
    "type": "basic",
    "epsilon": 1.0,
    "delta": 1e-5,
    "server": "trusted",
    "min_count": 100,
    "featurizer": "SELECT age FROM profile.demographics",
    "aggregator": "median"
}
```

Figure 4-1: This shows the JSON representation of a basic task whose goal is to compute a differentially private estimate of the median age across all of the data contributors who accept this task and contribute their age to it.

Statistic	Relevant Work
Count	(Dwork et al., [15])
Median	(Nissim et al., [32])
Mean	(Li et al., [29]), (Wilson et al., [39])
Sum	(Li et al., [29]), (Wilson et al., [39])
Variance	(Wilson et al., [39])

Table 4.2: This lists the different operations that are currently supported on our platform through basic tasks.

private manner before releasing the result to the *requestor*.

By default, the *server* uses a generic sampling-based procedure designed to support arbitrary aggregation functions. This procedure works by (1) selecting random subsets of the values from *data contributors*, (2) computing the value of the function on each subset, (3) estimating the approximate bounds for these values, and (4) computing the median of these values in a differentially private manner to produce an estimate of the function value.

For some specific aggregation functions such as the count, however, there exist custom mechanisms which outperform the generic sampling-based procedure; in these cases, the *server* will automatically use the best available implementation. We provide a detailed description of the *sampling-based procedure* and *bounds approximation algorithm* in the following subsections and refer readers to Table 4.2 which lists the aggregation functions that are supported and relevant literature.

Sample And Aggregate

Many of these tasks are implemented using the the sample-and-aggregate framework [32] which provides a mechanism for computing a noisy estimate of a function f on dataset D where the amount of noise is calibrated to both the function and the dataset to guarantee (ϵ, δ)-differential privacy.

The two key ideas behind the sample-and-aggregate paradigm are (1) estimating the function f on subsets of the data and (2) aggregating the estimates with a differentially private mechanism. To efficiently perform this aggregation, Nissim et al. define the β-smooth sensitivity of a function f as

$$S_\beta(x) = \max_{y \in D^n}(LS_f(y) \cdot e^{-\beta d(x,y)}) \tag{4.1}$$

and prove that it is an upper bound on the local sensitivity LS of the function [32].

By calibrating the scale of the noise distribution to the smooth sensitivity, we can guarantee (ϵ, δ)-differential privacy. The smooth sensitivity of the median (derivation in [32]) is given by

$$S_\beta = \max_{k=0,\ldots,n}(e^{-k\beta} \max_{t=0,\ldots,k+1}(x_{m+t} - x_{m+t-k-1})) \tag{4.2}$$

where we assume that $x_1, \ldots, x_n$ is in sorted order and $m = \frac{n+1}{2}$.

Then, to produce an (ϵ, δ)-differentially private mechanism for computing the median, we can simply add d-dimensional Laplacian noise scaled according to:

$$\beta = \frac{\epsilon}{4 \cdot (d + ln(2/\delta))} \tag{4.3}$$

$$M\hat{edian}(x) = Median(x) + \frac{2S_\beta(x)}{\epsilon} \cdot \text{Laplace}(0, 1) \tag{4.4}$$

Therefore, for a given scalar function f, we can evaluate the function f on subsets of the dataset D to produce values $f_1, \ldots, f_n$ where each value is a reasonable estimate of $f(D)$. Using the above mechanism, we can then compute the median of these estimates to obtain a stable and differentially private estimate of $f(D)$.

Approximate Bounds

Many differential privacy mechanisms require information about the domain of the values; for example, a simple differentially private estimate of the sum can be obtained by adding Laplacian noise scaled to $\frac{high-low}{\epsilon}$. In some specific situations, these bounds are known or can be assumed a priori (e.g. if the variable corresponds to a human age, then it is reasonable to assume it is in the range 0 to 150); however, in general, these bounds are not known by the requestor or the server and need to be estimated.

In practice, many implementations of differential privacy algorithms will directly estimate the lower and upper bounds from the data, resulting in some privacy leakage; for example, in the open source implementation of [21], if bounds are not specified, then they are directly estimated from the data and a privacy leakage warning is logged.

An alternative approach is to use some of the privacy budget to first obtain a differentially private approximation of the bounds, clip the data to this range, and then use these bounds for the subsequent operation, as implemented in [39]. At a high level, this approximation procedure works as follows:

1. Compute a log histogram of the values sent by the data contributors.

2. Add noise to the log histogram to make it differentially private.

3. Identify the first and last bins of the histogram which are above a given threshold; these bins correspond to a differentially private estimate of the minimum and maximum values present in the data.

4. Clip the values sent by the data contributors so they lie between the minimum and maximum values obtained by this procedure.

This results in a new set of values with a known domain that can be supplied to the downstream differential privacy mechanism.

Field	Description
`epsilon`	This field takes a numerical value greater than 0.0.
`delta`	This field takes a numerical value between 0.0 and 1.0.
`min_count`	This field takes an integer value greater than 10 corresponding to the minimum number of users who must contribute data before the task is processed.
`featurizer`	This field is expected to be a string containing a valid SQL query that extracts named variables from the user's personal data store.
`model`	This field is expected to be one of the following models: `GaussianNB`, `LinearRegression`, and `LogisticRegression`.
`inputs`	This field contains a list of variable names corresponding to the variables returned by the `featurizer`.
`output`	This field contains a variable names that matches one of the variables returned by the `featurizer`.

Table 4.3: This describes the fields of the JSON for an integrated task.

4.2 Integrated Tasks

This type of task involves (1) gathering a collection of named features from each contributor and (2) using these features to train a model on the server. For this type of task, the majority of the computation occurs on the *server* and the contributor *needs to trust the server* to correctly train and release a differentially private model.

4.2.1 Task Representation

This class of tasks involves two components, a client-side `featurizer` operation which computes and returns an collection of features and a server-side `model` operation which is typically a type of machine learning model which has been modified to be differentially private.

The JSON fields associated with an integrated task are listed in Table 4.3. We note that the `featurizer` returns a dictionary of key-value pairs; the `inputs` and `output` fields reference these key-value pairs and specify which features are the covariates and which feature is the outcome. A complete example of an integrated task which trains a Gaussian naive bayes model is shown in Figure 4-2.

Example: Integrated Task

```
{
    "type": "integrated",
    "epsilon": 10.0,
    "min_count": 100,
    "server": "trusted",
    "featurizer": "SELECT x1, x2, y FROM profile.example",
    "model": "GaussianNB",
    "inputs": ["x0", "x1"],
    "output": "y"
}
```

Figure 4-2: This example shows the JSON representation of an integrated task which trains a Gaussian naive bayes classifier in a differentially private manner.

4.2.2 Computation

We provide a listing of machine learning models that are supported by this type of task in Table 4.4. We omit a detailed description of the mechanisms used to support these differentially private machine learning algorithms and refer readers to the relevant works cited above. For example, the naive bayes integrated task shown in Figure 4-2 uses a differentially private implementation of the Gaussian Naive Bayes classifier as proposed in [38].

In addition, we note that although our implementation of these algorithms is based on DiffPrivLib by IBM [21], we make substantial modifications, particularly in the case of bounds handling. In the authors' implementation of [21], when the bounds are not explicitly set by the user, they are directly estimated from the data, resulting in possible privacy leakage. In our implementation, when the bounds aren't specified, we use the log histogram technique from [39] to obtain a differentially private approximation of the bounds, clip the data to those bounds, and provide those bounds to the machine learning algorithm.

Operation	Relevant Work
Linear Regression	(Holohan et al., [21]), (Sheffet, [35])
Logistic Regression	(Holohan et al., [21]), (Chaudhuri et al., [6])
Naive Bayes	(Holohan et al., [21]), (Vaidya et al., [38])

Table 4.4: This lists the different operations that are currently supported on our platform through integrated queries.

4.3 Bounded Tasks

This type of task involves gathering a collection of named features or variables from each contributor where the values are perturbed by the *client* before they are sent to the server. For this type of task, the *client* is responsible for the majority of the computation and the *server* simply relays the values to the *requestor*.

4.3.1 Task Representation

The JSON fields associated with a bounded task are listed in Table 4.5. This class of tasks requires a client-side `featurizer` operation which computes and returns a collection of features where the domain for each feature is specified in `bounds`. Features may be either continuous, in which case the domain is specified by the lower and upper bound, or categorical, in which case the domain is specified as a list of all possible values that it can take on. We provide a complete example of a bounded task in Figure 4-3.

4.3.2 Computation

When processing bounded tasks, the *client* is responsible for performing all of the computations to support local differential privacy. Since all the features have a known domain, the *client* can simply apply the following mechanisms to generate a differentially private copy of each feature.

To generate an ϵ-differentially private version of a numerical variable with bounded

Field	Description
`epsilon`	This field takes a numerical value greater than 0.0.
`delta`	This field takes a numerical value between 0.0 and 1.0.
`min_count`	This field takes an integer value greater than 10 corresponding to the minimum number of users who must contribute data before the task is processed.
`featurizer`	This field is expected to be a string containing a valid SQL query that extracts named variable from the user's personal data store.
`bounds`	This field contains a dictionary mapping each variable name to the bounds for that variable. The bounds can be either a set of values, in the case of a discrete variable, or a range of values, in the case of a continuous variable.

Table 4.5: This describes the fields of the JSON for a bounded task.

range $[a, b]$, we can apply the classic Laplace mechanism

$$\hat{x} = \text{Laplace}(x, \frac{b-a}{\epsilon}) \tag{4.5}$$

which has been explored in numerous previous works [15, 10].

Similarly, the Gaussian mechanism for (ϵ, δ)-differential privacy is given by

$$\sigma = \sqrt{2ln(1.25/\delta)}\frac{(b-a)^2}{\epsilon} \tag{4.6}$$

$$\hat{x} = \mathcal{N}(x, \sigma^2) \tag{4.7}$$

which can be preferable to the Laplace mechanisms in situations where the noise distribution is expected to be Gaussian.

To generate a differentially private version of a categorical variable with N possible values [25], we can simply sample from the following distribution

$$P(\hat{x}|x) = \begin{cases} \frac{e^\epsilon}{N+e^\epsilon-1} & \text{if } \hat{x} = x \\ \frac{1}{N+e^\epsilon-1} & \text{otherwise} \end{cases} \tag{4.8}$$

Example: Bounded Task

```
{
    "type": "bounded",
    "epsilon": 1.0,
    "min_sample_size": 100,
    "server": "untrusted",
    "featurizer": "SELECT species, age FROM profile.pets",
    "bounds": {
        "species": {
            "type": "set",
            "default": "dog",
            "values": ["cat", "dog"],
        },
        "age": {
            "type": "range",
            "low": 0.0,
            "high": 100.0
        },
    }
}
```

Figure 4-3: This shows the JSON representation of a bounded task which applies local differential privacy mechanisms to the species and age values. The untrusted server field refers to the fact that the differential privacy mechanism is applied on the client side.

which trivially satisfies ϵ-differential privacy since

$$\frac{P(\hat{x} = x|x)}{P(\hat{x} = z|x)} \leq e^{\epsilon} \tag{4.9}$$

for all z such that $z \neq x$. We can derive a similar mechanism for (ϵ, δ)-differential privacy which samples from the following distribution

$$P(\hat{x}|x) = \begin{cases} \frac{e^{\epsilon}+\delta(N-1)}{N+e^{\epsilon}-1} & \text{if } \hat{x} = x \\ \frac{(1-\delta)(N-1)}{(N-1)(N+e^{\epsilon}-1)} & \text{otherwise} \end{cases} \tag{4.10}$$

and allows us to make comparisons with (ϵ, δ)-differential privacy algorithms.

4.4 Gradient Tasks

This type of task requires the *client* to compute a gradient update for a neural network using their personal data. Then, the *client* adds noise to the gradient update before sending it to the *server* where it is merged with gradient updates from all other data contributors in order to produce the final gradient update that can be used to optimize the neural network.

4.4.1 Task Representation

The JSON fields associated with a gradient task are listed in Table 4.6. The `model` field contains a serialized representation of a PyTorch model while the `loss` field contains a serialized representation of a PyTorch loss function. To evaluate a gradient task, the client performs the following:

1. Loads the appropriate features from their personal data store.
2. Evaluates the `loss` function by passing the features to the `model`.
3. Uses backpropagation to compute the gradient update.
4. Adds noise to the gradients to provide differential privacy.

We provide an example of a gradient task in Figure 4-4.

4.4.2 Computation

We implement a fully distributed model of federated learning where the participants compute gradients and apply privacy mechanisms locally [1, 17, 30]. Suppose the dataset contains n examples and we want to compute a single gradient update. In our setting, since we want to apply the noise on the client side, we use a slightly modified version of the mechanism described in [30]. Starting with a set of n users and a target batch size of $b = qn$, we apply the following procedure:

1. Each user randomly chooses whether to contribute to the gradient task with probability q.

Field	Description
`epsilon`	This field takes a numerical value greater than 0.0.
`delta`	This field takes a numerical value between 0.0 and 1.0.
`min_count`	This field takes an integer value greater than 10 corresponding to the minimum number of users who must contribute data before the task is processed.
`featurizer`	This field is expected to be a string containing a valid SQL query that extracts features from the user's personal data store.
`model`	This field is expected to contain a encoded representation of a PyTorch model. The Tiresias library provides helper functions for generating this representation.
`loss`	This field is expected to contain a encoded representation of a PyTorch loss function. The Tiresias library provides helper functions for generating this representation.
`inputs`	This field contains a list of variable names corresponding to the variables returned by the `featurizer`.
`outputs`	This field contains a list of variable names corresponding to the variables returned by the `featurizer`.

Table 4.6: This describes the fields of the JSON for a gradient task.

2. If chosen, the contributor computes the gradient g_i for their data and clips the maximum L_2 norm with $\hat{g}_i = g_i \cdot min(1, \frac{S}{||g_i||_2})$.

3. Each contributor then computes $\hat{g}'_i = \hat{g}_i + \mathcal{N}(0, \sigma^2 S)$ where $\sigma = \frac{1}{\epsilon}\sqrt{2ln1.25/\delta}$.

4. The server aggregates all the gradients and computes the gradient update as $\hat{g} = \frac{1}{n}\sum_i \hat{g}'_i$.

If only the final gradient update is released, then this procedure results in a mechanism that is $(q\epsilon, q\delta)$-differentially private with respect to the database. However, from the user's perspective, even if their individual gradients are revealed (e.g. due to a vulnerability on the server which exposed their data), they are still guaranteed (ϵ, δ) local differential privacy.

Note that if a requestor wants to fully train a neural network (as opposed to computing a single gradient update), they will need to (1) create a task to compute a gradient update for the model, (2) update the model with the result of the task, and (3) repeat steps 1-2 until the model converges. If we apply the basic composition

Example: Gradient Task

```
{
    "type": "gradient",
    "epsilon": 10.0,
    "delta": 1e-5,
    "lr": 0.01,
    "min_sample_size": 100,
    "featurizer": "SELECT x1, x2, y FROM profile.example",
    "model": torch.nn.Sequential(
        torch.nn.Linear(2, 10),
        torch.nn.ReLU(),
        torch.nn.Linear(10, 1),
    ),
    "loss": torch.nn.functional.mse_loss,
    "inputs": ["x0", "x1"],
    "output": ["y"],
}
```

Figure 4-4: This shows the JSON representation of a gradient task which computes the gradient update for a simple feedforward neural network. The untrusted server field refers to the fact that the differential privacy mechanism is applied to the gradients on the client side. Note that our platform natively supports PyTorch models and provides helper methods for serializing PyTorch models and loss functions.

theorem, then the total privacy budget for fully training a neural network is simply the sum of the privacy parameters for each individual task that was used in the training process.

We observe that this result is suboptimal as alternative accounting methods such as the Moments Accountant from [1] can typically provide tighter bounds on the privacy loss. Unfortunately, these methods require the set of contributors to be static throughout the training process, which is not guaranteed in our setting as users are free to accept or decline any individual task. Therefore, these alternative accounting methods are not currently supported by the Tiresias platform.

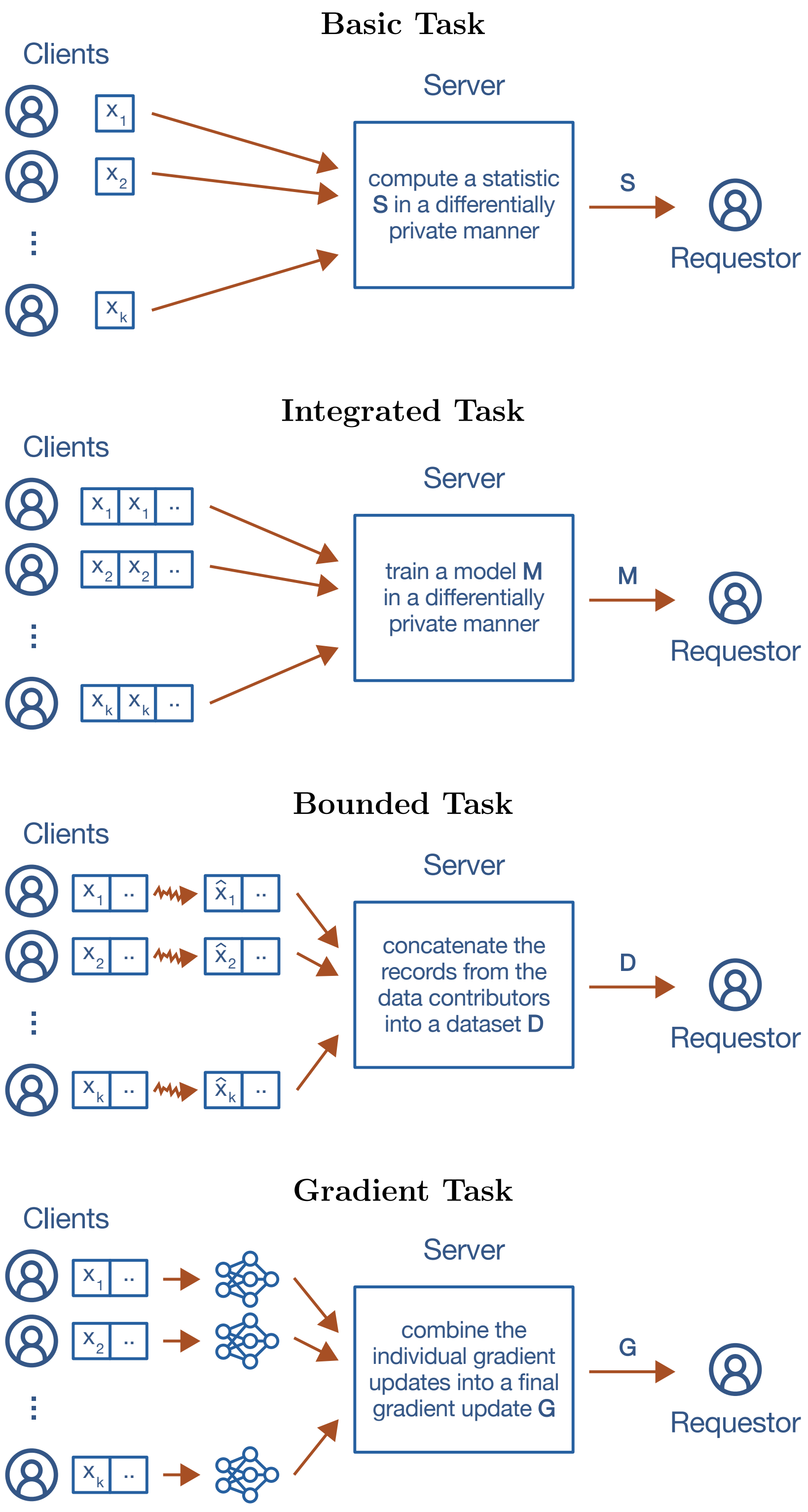


Figure 4-5: This summarizes the different types of computation that happen on the client and server for each type of task.

Chapter 5

Experiments

In this section, we demonstrate how a variety of different tasks, models, and mechanisms can be executed on the Tiresias platform and some of the applications they can be used for. We start by providing a broad overview of the different models that can be trained using our platform in Section 5.1, demonstrate the effectiveness of our platform for solving a specific real-world task in Section 5.2, show how more complicated procedures such as transfer learning can be implemented on top of our platform in Section 5.3, and provide some baseline performance measures for the scalability of our system in Section 5.4.

5.1 Supporting Benchmarking

In this set of experiments, we aim to demonstrate some of the different machine learning models that are supported by our system as well as some of the different ways that they can be trained. To do this, we will make use of the built-in benchmarking framework which allows machine learning researchers to quickly experiment with new machine learning models and differential privacy techniques by evaluating them in a simulated environment.

This simulated environment works by taking a dataset containing N rows and pretending that each row belongs to a different data contributor. We can then train the model using the implementation provided by the Tiresias platform and report the

Dataset	Task	Model	Epsilon	Accuracy	Time
Wine	Bounded	Logistic Regression	32.0	0.704	0.002
			64.0	0.931	0.002
		Random Forest	32.0	0.583	0.013
			64.0	0.811	0.014
		Support Vector Machine	32.0	0.670	0.019
			64.0	0.910	0.016
	Integrated	Logistic Regression	32.0	0.901	0.010
			64.0	0.974	0.010
		Naive Bayes	32.0	0.917	0.040
			64.0	0.926	0.039
	Gradient	Multilayer Perceptron	32.0	0.498	4.685
			64.0	0.745	4.718
Cancer	Bounded	Logistic Regression	32.0	0.633	0.008
			64.0	0.689	0.020
		Random Forest	32.0	0.659	0.020
			64.0	0.689	0.037
		Support Vector Machine	32.0	0.646	0.038
			64.0	0.704	0.009
	Integrated	Logistic Regression	32.0	0.929	0.026
			64.0	0.952	0.009
		Naive Bayes	32.0	0.874	0.062
			64.0	0.881	0.063
	Gradient	Multilayer Perceptron	32.0	0.911	19.45
			64.0	0.934	15.81

Table 5.1: This table shows the results for training a variety of models (using different types of tasks) on two classification datasets. The accuracy and running time are reported for two different privacy levels.

performance of the model on a held-out validation set.

5.1.1 Classification

One of the key features of the Tiresias platform is its flexibility which allows users to experiment with and deploy a variety of different machine learning approaches. In this section, we demonstrate some of the different methods for training a classification model which include:

1. **Bounded task.** Models constructed using bounded tasks apply local differential privacy mechanisms on the client side to perturb the data before it is sent

to the researcher who then trains the model.

2. **Integrated task.** Models constructed using integrated tasks use differentially private implementations of popular machine learning models that run directly on the data.

3. **Gradient task.** Models constructed using gradient tasks work by applying differential privacy mechanisms to the gradients on the client side before it is sent to the server who merges the gradients and updates the model.

We start by exploring two datasets: the wine dataset which contains 178 samples with 3 classes and 13 features and the breast cancer dataset which contains 569 samples with 30 features [9]. The results from these experiments are shown in Table 5.1 and we observe that, in general, the mechanisms where the server (as opposed to the client) is responsible for offering the differential privacy guarantees result in the best overall performance.

We note that these results are intuitive as the **bounded tasks** are designed for high-privacy low-utility use cases such as computing summary statistics and the **gradient tasks** are designed for deep learning applications (as opposed to tabular classification/regression) whereas the **integrated tasks** apply techniques which are explicitly designed to solve this type of tabular machine learning problem.

5.1.2 Regression

Next, we demonstrate the corresponding set of task types and models for regression problems. We show that a variety of methods — from basic linear regression (through either a bounded task approach or an objective perturbation approach) to feedforward neural networks — are supported natively by our platform.

We evaluate these approaches on two public domain datasets: the diabetes dataset which contains 442 samples with 10 features and the California housing prices dataset which contains 20,640 samples with 8 features [9]. The results from these experiments are shown in Table 5.2 and we note that the performance of different model and task types is consistent with those found in the classification experiments.

Dataset	Task	Model	Epsilon	R^2	Time
Diabetes	Bounded	Linear Regression	32.0	0.087	0.001
			64.0	0.240	0.001
		Random Forest	32.0	0.122	0.021
			64.0	0.202	0.020
		Support Vector Machine	32.0	0.110	0.001
			64.0	0.275	0.004
	Integrated	Linear Regression	32.0	0.472	0.001
			64.0	0.483	0.008
	Gradient	Multilayer Perceptron	32.0	0.105	13.18
			64.0	0.269	12.22
House Prices	Bounded	Linear Regression	32.0	0.140	0.002
			64.0	0.325	0.001
		Random Forest	32.0	0.162	0.028
			64.0	0.347	0.026
		Support Vector Machine	32.0	0.085	0.020
			64.0	0.248	0.018
	Integrated	Linear Regression	32.0	0.704	0.001
			64.0	0.717	0.001
	Gradient	Multilayer Perceptron	32.0	0.260	12.26
			64.0	0.329	14.79

Table 5.2: This table shows the results for training a variety of models (using different types of tasks) on two regression datasets. The accuracy and running time are reported for two different privacy levels.

5.2 Pull Request Prediction

In this set of experiments, we train a logistic regression model to predict whether a pull request will be accepted. The data for this experiment comes from the *Github Archive*[1] and is preprocessed to simulate an environment where (1) users installed the Tiresias client on their device, (2) they are using a collector to capture data about their Github usage[2], and (3) they are choosing to contribute their data to the tasks detailed below.

We randomly selected 100 individuals who created 10 or more pull requests in February 2019 and extracted all the rows pertaining to these individuals from the

[1]`https://www.gharchive.org`

[2]We note that all the data used in this experiment is publicly available; however, for the purposes of this experiment, we assume that each individual is only aware of their personal Github activity.

dataset. For this experiment, each of these individuals represents a data contributor who installed a Github data collector and captured the following:

1. Information about the repository such as the number of stars and followers.

2. Information about the individual who submitted the pull request such as the number of repositories and commits.

3. Information about the historical pull request accept/reject rate for the target repository.

All of these features, including a binary variable indicating whether the pull request was eventually accepted, are stored in the client and used to train the model.

We explore a few different ways to train a model for predicting pull request acceptance using our system and evaluate the performance of each approach along two axes: test accuracy and running time. The Tiresias platform provides multiple ways to train a logistic regression model; the three methods we evaluate in this experiment (and compare to the baseline) are:

1. **Baseline.** In the first scenario, we use plain logistic regression (without hyperparameter optimization) to set a baseline for the level of performance that can be achieved using this dataset.

2. **Integrated task.** In this scenario, we use the objective perturbation method from [6] to train a logistic regression model.

3. **Bounded task.** In this scenario, we use local differential privacy to anonymize the data before it is sent to the server, offering stronger privacy guarantees at the cost of accuracy.

4. **Gradient task.** In this scenario, we use federated learning to compute differentially private gradient updates locally on the user's device and aggregate them to produce the trained model.

Method	Accuracy	Running Time
Baseline	0.94	0.10
Bounded Task	0.83	0.31
Integrated Task	0.92	0.16
Gradient Task	0.88	6.30

Table 5.3: The performance of various implementations of logistic regression when predicting pull request acceptance with $(8.0, 10^{-5})$-differential privacy.

As shown in Table 5.3, each of these methods presents a different set of trade-offs between privacy, accuracy, and speed. It's clear that the **integrated task** approach provides the most utility for the requestor as it is fast and accurate but it requires the data contributor to trust the platform to correctly anonymize the data. On the other hand, the **bounded task** approach provides the least utility for the requestor due to the relatively low accuracy but it doesn't require the user to trust the server. Finally, the **gradient tasks** approach provides a reasonable balance between accuracy and privacy, giving requestors more accurate results while requiring no trust from the user but at the cost of computational efficiency.

5.3 Transfer Learning

Next, we demonstrate our platform's ability to support more complex tasks such as transfer learning. In this section, we show how we can take the SqueezeNet [22] and VGG [36] neural network architectures which can be pretrained on publicaly available data and then use Tiresias to perform transfer learning in a differentially private setting.

Specifically, we will demonstrate how a **gradient task** can be used to adapt these pretrained models to the STL-10 [3] dataset which contains 5,000 training images from 10 classes. Using this dataset, we simulate a setting where there are 5,000 users, each of whom has a single training example that they want to keep differentially private.

We implement two types of transfer learning: **feature extraction**, which holds

Model	Method	Epsilon	Accuracy	Running Time
SqueezeNet	Model Fine-Tuning	8.0	0.071	2,899
SqueezeNet	Model Fine-Tuning	16.0	0.095	2,927
SqueezeNet	Model Fine-Tuning	32.0	0.410	5,864
SqueezeNet	Feature Extraction	8.0	0.747	1,195
SqueezeNet	Feature Extraction	16.0	0.783	1,163
SqueezeNet	Feature Extraction	32.0	0.781	1,212
VGG16	Feature Extraction	8.0	0.904	20,989
VGG16	Feature Extraction	16.0	0.919	21,220
VGG16	Feature Extraction	32.0	0.920	20,274

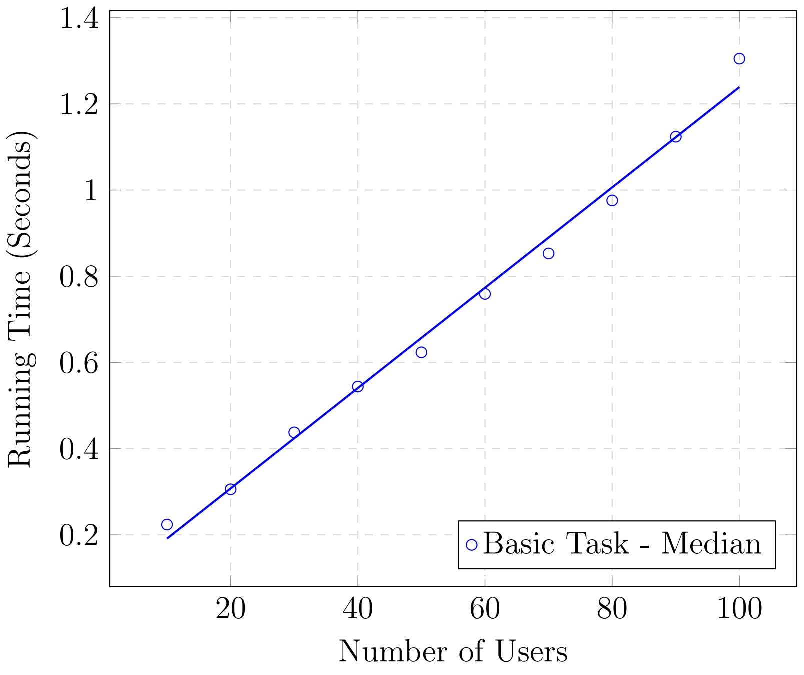
Table 5.4: The performance of various transfer learning techniques and privacy levels on the STL-10 [3] dataset.

the weights of the pretrained network fixed except for the final classification layer and **fine-tuning** which computes gradients for the entire pre-trained network. Each of these models were trained for 32 epochs with $\delta = 10^{-5}$, learning rate 0.01, and lot size 100. As shown in Table 5.4, we find that our platform is able to support common transfer learning scenarios using gradient queries, achieving test accuracy values competitive with non-private transfer learning algorithms.

5.4 Scalability

Finally, we demonstrate the scalability of our platform by performing a series of experiments on Amazon Web Services. The Tiresias client, upon installation, generates an example dataset for testing purposes; in this experiment, we measure the amount of time it takes to compute a differentially private task for different numbers of users. Our experimental setup is as follows:

1. We launch **t2.large** instance on AWS with 2 virtual CPUs and 8 GB of memory and use this machine to host our platform.
2. Next, we launched 10 **t2.nano** instances on AWS with 1 CPU and 0.5 GB of memory; we run 10 different instances of the **Tiresias** client on each of these machines to simulate a total of up to 100 users who could potentially contribute data.



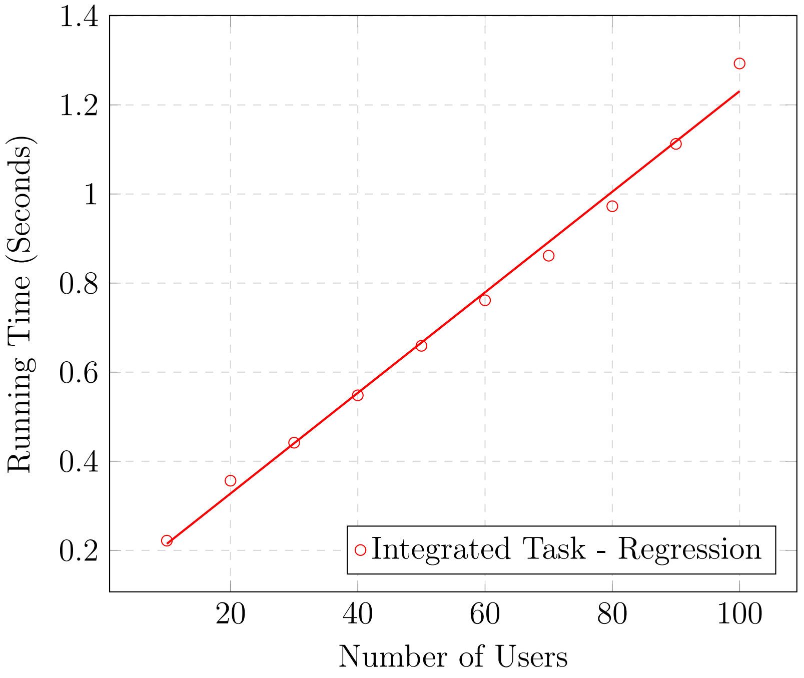


Figure 5-1: This plot shows the amount of time needed to (1) compute the median using a basic task and (2) train a linear regression model using an integrated task for different numbers of users.

3. Finally, we submit the median and regression queries with a limit of N contributors for N in {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

4. We perform the above experiment 20 times and report the average running time for each query against the number of contributors.

The results for this experiment are shown in Figure 5-1 and we note that both the median and linear regression queries complete within a reasonable time window of approximately one second. Furthermore, we note that the primary bottleneck is the network bandwidth.

Chapter 6

User Studies

In this section, we design and propose a set of user studies which will aim to demonstrate the usability of our system. We plan to ask users to test our system from the perspective of (1) a data contributor who has used our system to collect some data and now wants to try contributing it to a task as well as (2) a data requester who wants to write and run tasks on private user data.

For these experiments, we generate the following synthetic datasets and prepopulate the client with them. We ask the participants in these experiments to treat the simulated data in the client as though it were their own.

- **Demographics.** Age, Gender, Income, City, State, Zip Code
- **Screen Time**
  - **Events.** Timestamp, Event Type, Application Name
  - **Types.** Application Name, Application Type
- **Browsing History.** Timestamp, Domain

These synthetic datasets will be sampled from a Bayesian network which makes plausible assumptions about the relationships between these variables (i.e. income is correlated with geographical location, screen time is correlated with age, etc.) and inserted into the client using the data collection API described in Section 3.3 as though they were being captured in real time by a data collector.

6.1 Data Contributors

First, we propose a user study to understand how contributors interact with our system. We plan to perform this experiment with three different groups of users consisting of (1) individuals who have experience with machine learning and data science, (2) individuals with a background in computer science and related fields, and (3) individuals who are representative of the general public. Our goal will be to answer the following questions for each type of user:

1. Are contributors able to contribute their data?
2. Do contributors understand what each task is asking for?
3. Do contributors feel comfortable sharing their data in a differentially private manner?

To accomplish this, we will give each participant access to a client preloaded with simulated data as described earlier. We will then submit a suite of tasks to the server, as listed below, and ask the participants to (1) select which of the tasks that they are willing to contribute their data to and (2) answer some questions about each task to demonstrate their understanding.

We implement the following tasks and submit them to the server for the participants in this experiment to examine:

- Compute the median age of all users using a **basic task**.
- Train a model to predict the user's age from the rest of their demographic information using an **integrated task**.
- Train a model to predict the user's age from their browsing history using an **integrated task**.
- Report the average amount of time spent by users on different types of applications using a **bounded task**.

The users will be able to see these tasks, read the automatically generated descriptions of what privacy guarantees each task offers, and select which tasks to contribute to. We will report the percent of participants who accept each task and summarize their responses to our questions.

6.2 Data Requestors

Next, we propose a user study to understand how requestors interact with our system. Our goal will be to answer the following questions:

1. Are requestors able to write tasks to solve specific problems?
2. Do requestors understand the different types of tasks they can create?

For this experiment, we will configure the clients to randomly accept tasks. Furthermore, we will create a list of problems, as listed below, that can be solved using the given datasets and ask each participant in this experiment to try to solve one or more of these problems by writing a task and submitting it to the server.

We present the following problems to the participants in this experiment and ask them to solve them using the tools provided by our system:

- What percent of the population is male?
- Train a model to predict a person's gender from their browsing history.
- Identify the most frequently visited websites for people older than 21.

The participants in this experiment will write and submit tasks to the server interactively using the Python API for Tiresias. We will report the percent of participants who successfully write a task to solve each problem and summarize their responses to our questions.

Chapter 7

Future Work

7.1 Automated Machine Learning

One direction for future work is to analyze and integrate tools for automated machine learning. This includes everything from automated feature engineering, which can be used to transform the raw features and data that is captured by the data collectors into more useful features [26], to hyperparameter tuning for the machine learning models [20].

This research direction is particularly compelling in the case of integrated queries where the machine learning process runs on the server. By providing differentially private implementations of automated feature engineering tools and hyperparameter tuning algorithms, we can enable users to train better models with lower privacy costs. Instead of writing a task that trains a single machine learning model, users would be able to write a task that automatically optimizes over the space of all possible model types and hyperparameters that are supported by the system.

The most closely related works in this area focus primarily on (1) the relationship between differential privacy and generalization/overfitting [11] and (2) differentially private combinatorial optimization [19]. As far as we know, there is no existing literature that directly addresses differentially private feature engineering or hyperparameter tuning, making this an exciting new area for research.

7.2 Collector Management

Another topic that merits further investigation is how collector applications are managed by the platform. Specifically, further work is needed to design a comprehensive approach to:

1. Providing the tools for downloading and installing collector applications in a verifiable and secure manner.
2. Managing the installation and removal of collector applications (specifically in the context of how the data is managed when an application is uninstalled and then potentially reinstalled).
3. Supporting software upgrades to collector applications which could require making modifications to the data schema and migrating the data across different versions of the application.

These challenges are closely related to those faced by application stores for mobile operating systems which must support installation, uninstallation, and upgrades. However, there are also unique challenges in our setting as deploying changes to the data schema will require careful thought to make sure it presents a consistent view to tasks that are in-progress.

7.3 Incentives and Economics

Furthermore, we believe that incentives and economics will also play a vital role in our system. Thus far, we've primarily thought of the users who provide data to our system as data contributors who chose whether or not to accept a task simply by looking at the data being requested, the privacy levels specified, and what the data will be used for. In practice, however, additional incentives may be needed to encourage users to contribute their data.

For example, in the most general setting, tasks could potentially be associated with a monetary bid value where data requestors would try to outbid one another in order to attract users to contribute to their task. This would essentially transform the platform into a data marketplace which can be studied using incentive design and auction theory [18]. By creating a digital marketplace for statistical data products, we create an ecosystem which allows users and researchers to work together to build new types of datasets and applications.

However, even in private settings without monetary rewards (e.g. in an academic setting where the school administrators wishes to collect data about their students and staff without compromising privacy), a better game-theoretic understanding of our system is still valuable since there are multiple conflicting incentives between the **collectors** who want to preserve their privacy, the **requestors** who want to maximize the accuracy of their results, and **platform maintainer** whose motivations for operating the platform may be unclear.

7.4 Developer Experience

7.4.1 Confidence Intervals

A common technique for making differentially private results more understandable is to provide confidence intervals for the output based on the noise distribution that was added to the model. This technique can be found in software implementations such as [39]. A similar body of work exists for the robustness of machine learning models that are trained in a differentially private manner [11, 5, 33, 27]. These works typically offer bounds on the generalization error but thus far have primarily been for purely theoretical interest; however, these bounds could also be of practical interest to users of our system who may be interested in bounds on how far the performance in the real world could deviate from the performance on the test set.

7.4.2 Bounds Approximation

For many of our queries, we use a bounds approximation algorithm whose performance can vary dramatically depending on the configuration. This bounds approximation works by computing a differentially private log histogram and then identifying the first and last bins that are above a given threshold. The scale and base used for the log histogram, as well as the threshold, can have a significant impact on the amount of noise in the result. Furthermore, if the threshold is too high, it's possible that no bins satisfy it, resulting in a bounds estimation error.

This is a motivating example for further work into how differentially private algorithms can be explained to a general audience — if the requestor is attempting to estimate a median and the bounds approximation procedure fails, how can we recover from this error without exposing the underlying complexity to the user?

7.4.3 Remote Debugging and Tooling

Finally, we note that debugging differentially private algorithms presents numerous novel challenges that have not been thoroughly explored. For example, suppose that the featurizer for a particular task crashes for a subset of users (but not all users). It is unclear how the error message and stack trace associated with the crash can be relayed to the requestor while preserving differential privacy; further work needs to be done to adapt existing differential privacy algorithms to provide an intuitive user experience for debugging various failure modes.

Chapter 8

Conclusion

In this thesis, we proposed a novel system for enabling differentially private peer-to-peer machine learning. The Tiresias platform presents a framework for data collection that allows individual users to maintain personal data stores; furthermore, the framework enables researchers to write a wide variety of tasks ranging from basic statistics to machine learning that operate on sensitive user data without compromising their privacy. This enables new types of collaboration, research, and data driven applications by allowing users to safely contribute their data to interesting tasks and allowing researchers to gain access to new types of cross-domain datasets.

Appendix A

Code

A.1 Smooth Sensitivity of the Median

This function computes the differentially private estimate of the median using the approach proposed in [32]. It uses the Laplace mechanism on page 10 and the smooth sensitivity of the median derivation found on page 12. The resulting value is (ϵ, δ)-differentially private.

```
import numpy as np

def median(x, epsilon, delta):
    alpha = epsilon / 2.0
    beta = epsilon / (2.0 * np.log(2.0 / delta))

    x = np.array(x)
    x.sort()
    m = (len(x) + 1) // 2
    ss = []
    for k in range(0, len(x)-m):
        ls = max(x[m+t] - x[m+t-k-1] for t in range(0, k+1))
        ss.append(np.exp(-k * beta) * ls)
    ss = max(ss)

    return np.median(x) + ss/alpha * np.random.laplace()
```

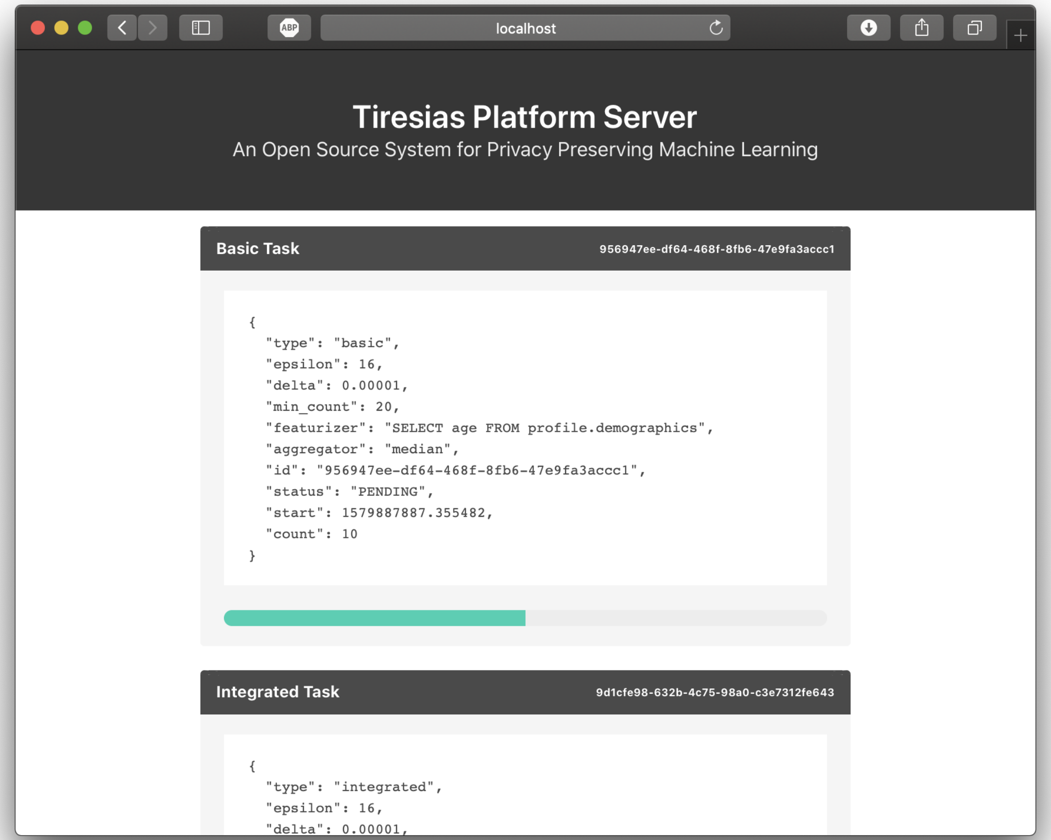
A.2 Approximate Bounds

The `dp_bounds` function estimates the upper and lower bounds of the data values using a variation of the histogram approach from [39]. These bounds can be used to support other mechanisms such as the `dp_mean` function which is based on the noisy average approach from [29] but will use part of the privacy budget to estimate the bounds if it is not given.

```
import numpy as np

def dp_bounds(x, epsilon, scale=1.0, base=2, bins=64, p=0.999):
    threshold = -np.log(2 - 2 * p) / epsilon
    cutoffs = scale * np.power(base, np.arange(0, bins//2))
    cutoffs = (-cutoffs[::-1]).tolist() + [0] + cutoffs.tolist()

    histogram = np.random.laplace(size=bins) / epsilon
    x = sorted(x.tolist())
    for i in range(len(cutoffs)-1, 0, -1):
        while len(x) > 0 and x[-1] >= cutoffs[i-1]:
            x.pop(-1)
            histogram[i] += 1
    histogram[0] += len(x)
    has_value = (histogram > threshold).nonzero()[0]
    return cutoffs[has_value[0]], cutoffs[has_value[-1]]

def dp_mean(x, epsilon, bounds=None):
    if not bounds:
        epsilon = epsilon / 2.0
        bounds = dp_bounds(x, epsilon)
    low, high = bounds
    x = np.minimum(np.maximum(x, low), high)
    noise = np.random.laplace() * (high - low) / epsilon
    return min(max(low, (np.sum(x) + noise) / len(x)), high)
```

Appendix B

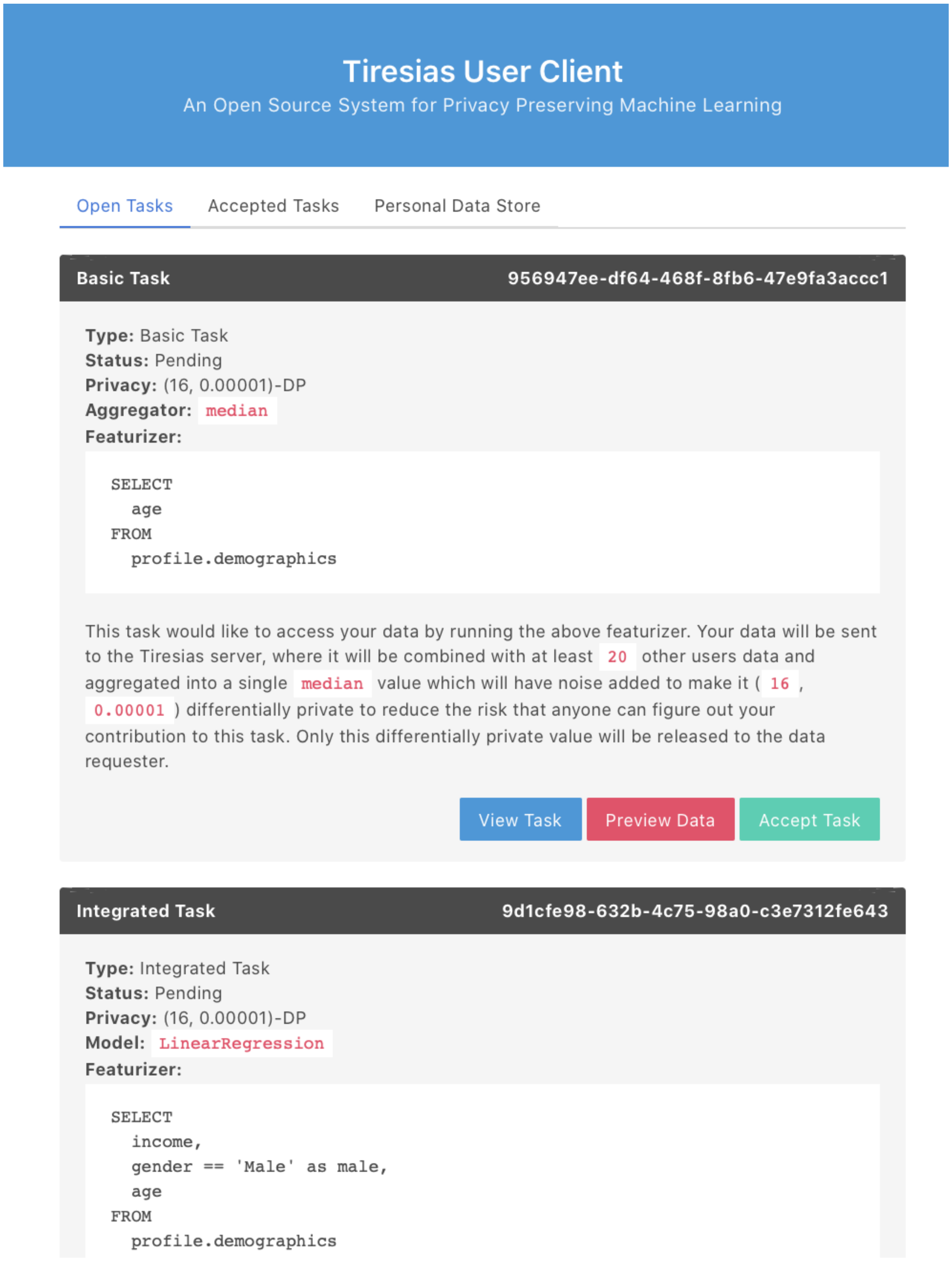
Figures


localhost
Tiresias Platform Server
An Open Source System for Privacy Preserving Machine Learning
Basic Task
956947ee-df64-468f-8fb6-47e9fa3accc1
Integrated Task
9d1cfe98-632b-4c75-98a0-c3e7312fe643

```
{
  "type": "basic",
  "epsilon": 16,
  "delta": 0.00001,
  "min_count": 20,
  "featurizer": "SELECT age FROM profile.demographics",
  "aggregator": "median",
  "id": "956947ee-df64-468f-8fb6-47e9fa3accc1",
  "status": "PENDING",
  "start": 1579887887.355482,
  "count": 10
}
```

```
{
  "type": "integrated",
  "epsilon": 16,
  "delta": 0.00001,
```

Figure B-1: This shows the web interface for the server which presents a list of all open tasks.


Tiresias User Client

An Open Source System for Privacy Preserving Machine Learning

Open Tasks | Accepted Tasks | Personal Data Store

Basic Task 956947ee-df64-468f-8fb6-47e9fa3accc1

Type: Basic Task
Status: Pending
Privacy: (16, 0.00001)-DP
Aggregator: `median`
Featurizer:

```
SELECT
  age
FROM
  profile.demographics
```

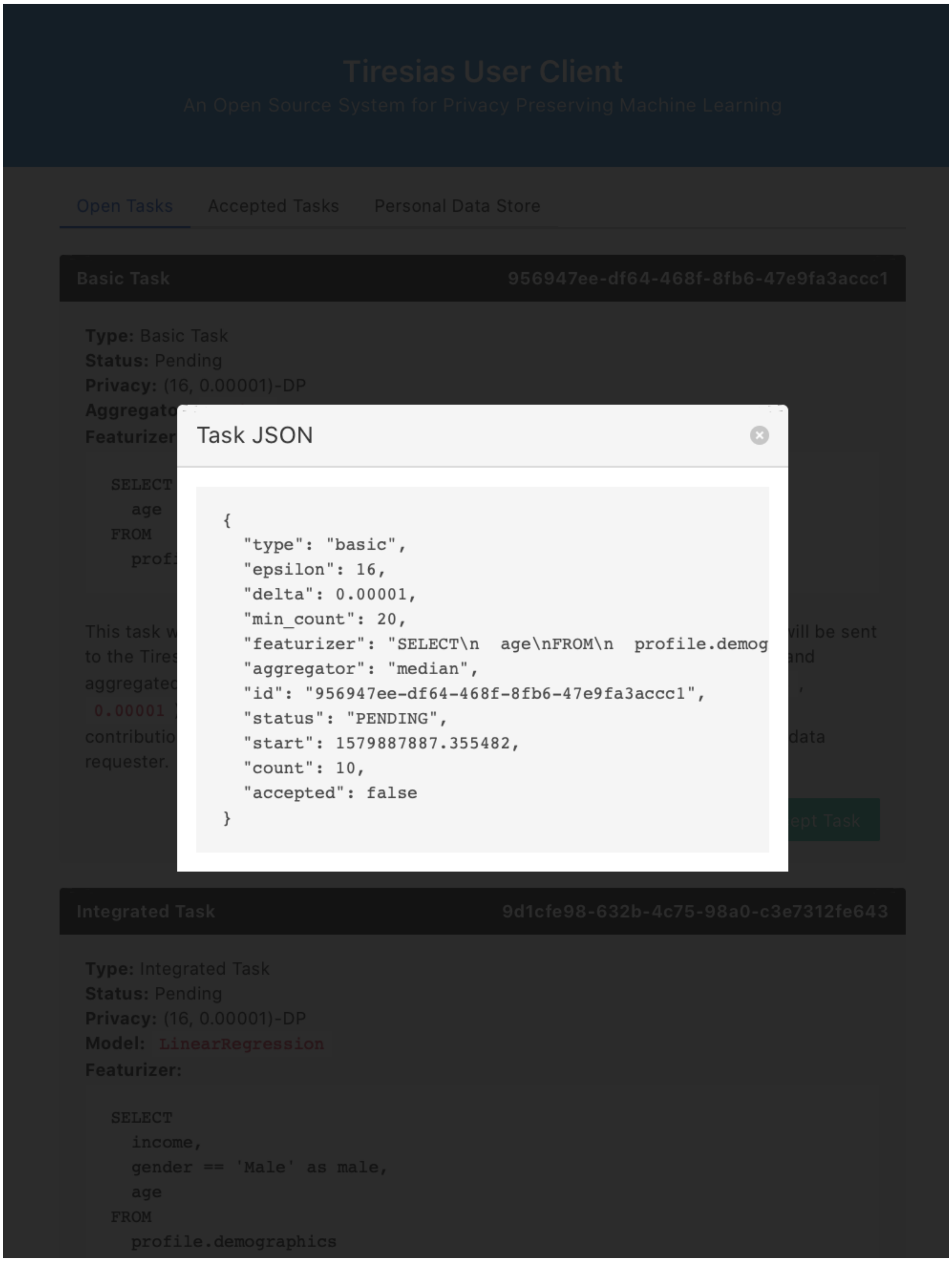
This task would like to access your data by running the above featurizer. Your data will be sent to the Tiresias server, where it will be combined with at least `20` other users data and aggregated into a single `median` value which will have noise added to make it (`16` , `0.00001`) differentially private to reduce the risk that anyone can figure out your contribution to this task. Only this differentially private value will be released to the data requester.

Integrated Task 9d1cfe98-632b-4c75-98a0-c3e7312fe643

Type: Integrated Task
Status: Pending
Privacy: (16, 0.00001)-DP
Model: `LinearRegression`
Featurizer:

```
SELECT
  income,
  gender == 'Male' as male,
  age
FROM
  profile.demographics
```

Figure B-2: This shows the graphical user interface for the client which presents a list of all open tasks that they can contribute to.


Tiresias User Client
An Open Source System for Privacy Preserving Machine Learning
Open Tasks
Accepted Tasks
Personal Data Store
Basic Task
956947ee-df64-468f-8fb6-47e9fa3accc1
Task JSON
Integrated Task
9d1cfe98-632b-4c75-98a0-c3e7312fe643

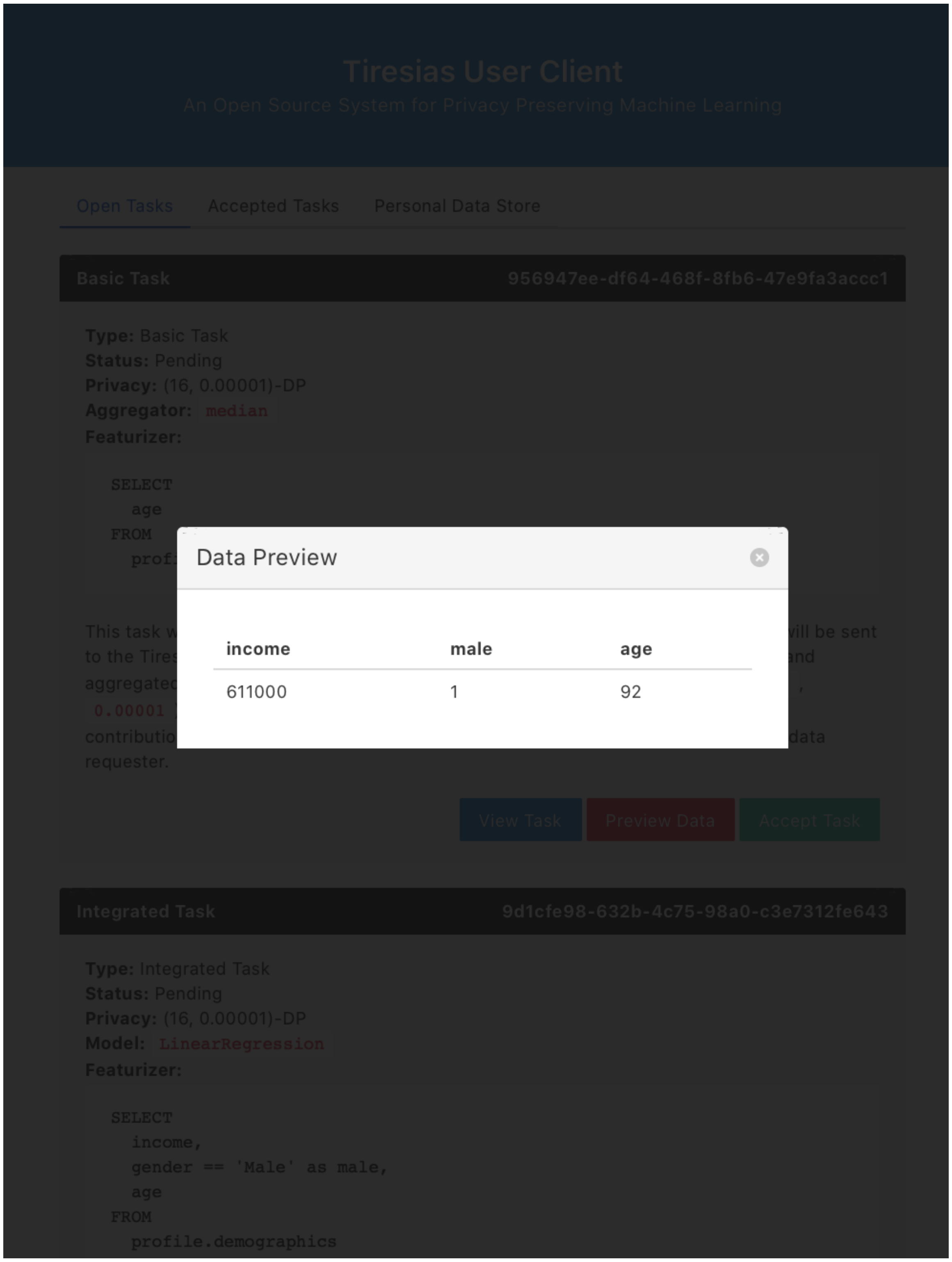
Type: Basic Task
Status: Pending
Privacy: (16, 0.00001)-DP

```
{
  "type": "basic",
  "epsilon": 16,
  "delta": 0.00001,
  "min_count": 20,
  "featurizer": "SELECT\n  age\nFROM\n  profile.demog
  "aggregator": "median",
  "id": "956947ee-df64-468f-8fb6-47e9fa3accc1",
  "status": "PENDING",
  "start": 1579887887.355482,
  "count": 10,
  "accepted": false
}
```

Type: Integrated Task
Status: Pending
Privacy: (16, 0.00001)-DP
Model: `LinearRegression`
Featurizer:

```
SELECT
  income,
  gender == 'Male' as male,
  age
FROM
  profile.demographics
```

Figure B-3: This shows the graphical user interface for the client, specifically the task view which allows the user to inspect the JSON representation of a given task.



Basic Task 956947ee-df64-468f-8fb6-47e9fa3accc1

Type: Basic Task
Status: Pending
Privacy: (16, 0.00001)-DP
Aggregator: `median`
Featurizer:

```
SELECT
  age
FROM
  profi
```

This task w … vill be sent to the Tires … and aggregated … , 0.00001 … contributio … data requester.

Data Preview

income	male	age
611000	1	92

Integrated Task 9d1cfe98-632b-4c75-98a0-c3e7312fe643

Type: Integrated Task
Status: Pending
Privacy: (16, 0.00001)-DP
Model: `LinearRegression`
Featurizer:

```
SELECT
  income,
  gender == 'Male' as male,
  age
FROM
  profile.demographics
```

Figure B-4: This shows the graphical user interface for the client, specifically the data preview modal which allows the user to inspect the data that will be sent to the platform if they accept the task.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA, 2016. ACM.

[2] Alessandro Acquisti, Curtis R. Taylor, and Liad Wagman. The economics of privacy. *Journal of Economic Literature*, 52(2), 2016.

[3] Andrew Y. Ng Adam Coates, Honglak Lee. An analysis of single layer networks in unsupervised feature learning. *AISTATS*, 2011.

[4] Imanol Arrieta-Ibarra, Leonard Goff, Diego JimÃĺnez-HernÃąndez, Jaron Lanier, and E. Glen Weyl. Should we treat data as labor? moving beyond "free". *AEA Papers and Proceedings*, 108:38–42, 2018.

[5] Raef Bassily, Kobbi Nissim, Adam Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. Algorithmic stability for adaptive data analysis. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1046–1059, 2016.

[6] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109, 2011.

[7] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1655–1658, New York, NY, USA, 2018. ACM.

[8] Ashish Dandekar, Debabrota Basu, and Stéphane Bressan. Differentially private non-parametric machine learning as a service. In *DEXA*, 2019.

[9] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[10] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of ComputationâĂŤTAMC*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer Verlag, April 2008.

[11] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toni Pitassi, Omer Reingold, and Aaron Roth. Generalization in adaptive data analysis and holdout reuse. In *Advances in Neural Information Processing Systems*, pages 2350–2358, 2015.

[12] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[13] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380. ACM, 2009.

[14] Cynthia Dwork and Frank D McSherry. Differential data privacy, Apr 2010.

[15] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9, 01 2013.

[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014.

[17] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[18] Arpita Ghosh and Aaron Roth. Selling privacy at auction. *CoRR*, abs/1011.1375, 2010.

[19] Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1106–1125. Society for Industrial and Applied Mathematics, 2010.

[20] Laura Gustafson. *Bayesian tuning and bandits: an extensible, open source library for AutoML*. PhD thesis, Massachusetts Institute of Technology, 2018.

[21] Naoise Holohan, Stefano Braghin, PÃşl Mac Aonghusa, and Killian Levacher. Diffprivlib: The ibm differential privacy library, 2019.

[22] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[23] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N Wright. A practical differentially private random decision tree classifier. In *2009 IEEE International Conference on Data Mining Workshops*, pages 114–121. IEEE, 2009.

[24] Noah Johnson, Joseph P Near, and Dawn Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[25] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. In *Advances in neural information processing systems*, pages 2879–2887, 2014.

[26] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[27] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[28] Jaewoo Lee and Chris Clifton. How much is enough? choosing ϵ for differential privacy. In *Proceedings of the 14th International Conference on Information Security*, ISC'11, pages 325–340, Berlin, Heidelberg, 2011. Springer-Verlag.

[29] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. Differential privacy: From theory to practice. *Synthesis Lectures on Information Security, Privacy, & Trust*, 8(4):1–138, 2016.

[30] H Brendan McMahan, Galen Andrew, Ulfar Erlingsson, Steve Chien, Ilya Mironov, Nicolas Papernot, and Peter Kairouz. A general approach to adding differential privacy to iterative training procedures. *arXiv preprint arXiv:1812.06210*, 2018.

[31] Frank McSherry. Privacy integrated queries. *Communications of the ACM*, 53:89–97, September 2010.

[32] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[33] Luca Oneto, Sandro Ridella, and Davide Anguita. Differential privacy and generalization: Sharper bounds with applications. *Pattern Recognition Letters*, 89:31–38, 2017.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[35] Or Sheffet. Private approximations of the 2nd-moment matrix using existing techniques in linear regression. *arXiv preprint arXiv:1507.00056*, 2015.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[37] Statista. Global advertising spending from 2010 to 2018 (in billion U.S. dollars) , 2017.

[38] Jaideep Vaidya, Basit Shafiq, Anirban Basu, and Yuan Hong. Differentially private naive bayes classification. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 571–576. IEEE, 2013.

[39] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private sql with bounded user contribution. https://github.com/google/differential-privacy, 2019.