UNIVERSITY OF CALIFORNIA

Los Angeles

Building Blocks of the Small Data Ecosystem: From Data Acquisition and Processing to User-Facing Applications

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Faisal Sabah Alquaddoomi

2018

ABSTRACT OF THE DISSERTATION

Building Blocks of the Small Data Ecosystem:
From Data Acquisition and Processing to
User-Facing Applications

by

Faisal Sabah Alquaddoomi
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2018
Professor Jens Palsberg, Chair

Small data are the digital traces that we produce regularly when interacting with services or devices. These data are rich, multi-modal, and often personal in nature. This thesis describes the design space of small data systems, enumerating the concerns in acquiring and deriving actionable insights from the data. An idealized architecture is presented, with specific realizations of that architecture, and comparisons between these realized applications, comprising the majority of the work. While the systems discussed have a common core of primitives, the work discusses the peculiarities of each application domain and how these differences eventually lead to separate application stacks. The work closes with observations on ways to continue the exploration of this space – specifically toward small, self-contained stacks where sharing is entertained only when necessary – that draws on the work done so far and the lessons learned in that process.

The dissertation of Faisal Sabah Alquaddoomi is approved.

Mani B. Srivastava

Warren S. Comulada

Wei Wang

Jens Palsberg, Committee Chair

University of California, Los Angeles

2018

To my family, friends, and colleagues who supported me throughout my academic career.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would first like to thank my advisor, Deborah Estrin, for her support throughout my graduate career. Deborah always made it clear that her students' success was a priority; she has always been abundantly generous with her time and attention, and has always been willing to give her thoughts on any idea we would bring to the table. She would figure out **with** us how to make those ideas work, giving careful consideration to everyone's input. She had faith in me throughout and inspired me to try, just to see if I could make it, which is what ultimately carried me through.

I extend my gratitude to my committee: Warren S. Comulada, Mani B. Srivastava, Wei Wang, and especially to my committee chair, Jens Palsberg, who helped tremendously in synchronizing my paperwork with the university while I was abroad. Their comments and criticisms were invaluable in bringing my thesis to fruition.

Finally, I would like to thank my colleagues at CENS: Jinha Kang, Sasank Reddy, Nithya Ramanathan, and others whose encouragements and appraisals shaped my early academic interests. I'd also like to thank the colleagues and mentors with whom I worked in the Small Data Lab at Cornell Tech, specifically Longqi Yang, Fabian Okeke, Hongyi Wen, JP Pollack, and especially Andy Hsieh with whom I worked very closely in my PhD. I'd also like to thank my friends at ETH Zürich who entertained my outlandish ideas and addressed my misgivings to equal degrees.

VITA

2005–2007	Web Developer, Student Information Systems and Technologies (SIS&T), UCSB.
2007	B.S. (Computer Science), UCSB.
2011	M.S. (Computer Science), UCLA.
2013	Teaching Assistant (CS 31 Intro to Computer Science), UCLA.
2014–2016	Visiting PhD Student, Cornell Tech, NYC.
2017–present	Visiting Scientist, ETH Zürich.
2017–present	Mobile and Web Developer, BRCA Exchange, with the UC Santa Cruz Genomics Center.
2011–present	Ph.D. (Computer Science), UCLA, expected Summer 2018.

PUBLICATIONS

C.K. Hsieh, H. Tangmunarunkit, F. Alquaddoomi, J. Jenkins, J. Kang, C. Ketcham, B. Longstaff, J. Selsky, B. Dawson, D. Swendeman. "Lifestreams: A modular sense-making toolset for identifying important patterns from everyday life" In *Proceedings of the Conference on Embedded Networked Sensor Systems*, pp. 5:1-5:13, 2013

F. Alquaddoomi, C. Ketcham, D. Estrin. "The Email Analysis Framework: Aiding the Analysis of Personal Natural Language Texts." In *LinkQS: Workshop on Linking The Quantified Self*, 2014

H. Tangmunarunkit, C.K. Hsieh, Brent Longstaff, S. Nolen, John Jenkins, Cameron Ketcham, Joshua Selsky, F. Alquaddoomi, Dony George, Jinha Kang, Z. Khalapyan, Jeroen Ooms, Nithya Ramanathan, and Deborah Estrin. "Ohmage: A General and Extensible End-to-End Participatory Sensing Platform." In *ACM Transactions on Intelligent Systems and Technology 6(3), no. 38*, 2015

M.S. Aung, F. Alquaddoomi, C.K. Hsieh, M. Rabbi, L. Yang, J.P. Pollak, D. Estrin, T. Choudhury. "Leveraging Multi-Modal Sensing for Mobile Health: A Case Review in Chronic Pain." In *IEEE Journal of Selected Topics in Signal Processing 10(5)*, pp. 962–974, 2016.

J. Casillas, A. Goyal., J.A. Bryman, F. Alquaddoomi, P.A. Ganz, E. Lidington, J. Macadangdang, D. Estrin. "Development of a text messaging system to improve receipt of survivorship care in adolescent and young adult survivors of childhood cancer" In *Journal of Cancer Survivorship*, vol. 11, pp. 505–516, 2017

N.R. Davidson, K.R. Godfrey, F. Alquaddoomi, D. Nola, J.J. DiStefano III. "DISTING: A web application for fast algorithmic computation of alternative indistinguishable linear compartmental models" In *Computer methods and programs in biomedicine*, vol. 143, pp. 129–135, 2017

C.K. Hsieh, F. Alquaddoomi, F. Okeke, J.P. Pollak, L. Gunasekara, D. Estrin. "Small Data: Applications and Architecture" In *The Fourth International Conference on Big Data, Small Data, Linked Data and Open Data (ALLDATA)*, 2018

F. Alquaddoomi, D. Estrin. "Ranking Subreddits by Classifier Indistinguishability in the Reddit Corpus" In *The Tenth International Conference on Information, Process, and Knowledge Management (eKNOW)*, 2018

CHAPTER 1

Introduction

Small data are "digital traces" of our interactions with services and devices. They are routinely collected by service providers as large as Google or Facebook, or as small as individual email account providers. Small data also include the logged interactions with the proliferation of devices we use on a daily basis: cellphones, laptops, point-of-sale systems, wearable devices, etc., all produce these digital traces. While this data is routinely collected, gaining access to it can be a struggle. Furthermore, putting the data to use in a way that is comprehensible and effective for the end-user, scalable to large populations, agile enough to iteratively refine the requirements, and still engaging at the individual level, is an open research question. This thesis strives to put these efforts into context, to categorize and describe the data sources, transformations, and end-user applications that together form the **small-data ecosystem**, and to enumerate both our advancements in this space and what remains to be done.

Small data occupies a unique context in the larger scope of data processing systems. While it is similar to big data in that it is collected in aggregate, the way in which small data is accessed differs: users access and share their own data at the individual level, employing token-based APIs or data federation/portability mechanisms to do so. Systems which concern themselves with small data incorporate the user's input as a source of context during data processing, and the analyses that occur on the data are personal rather than population-level. Other actors may be involved in the use of small data; behavioral researchers, for instance, may work with small data owners to initiate research studies. Unlike big data, these datasets are provided and curated "from the ground up" through involvement of the producers and owners of their data. Granted that the data are personal, the tools to curate this data for processing and release must also be personal; they must identify potentially privacy-compromising data in a manner that is comprehensible to the end-user and also provide fine-grained filtering tools for redacting sensitive or irrelevant data.

Chapter 2 defines the small data ecosystem as a particular kind of software ecosystem; unlike a traditional software ecosystem, it consists of both small data and the systems that process it, which co-evolve through producing more sophisticated derived and joined data streams. The chapter lists the actors involved in the small data ecosystem, the dimensions of small data, and common considerations when implementing small data systems. Specifically, small data systems differ in the manner in which they implement componentization, caching, and normalization of their input streams. They also differ in how they bridge trust domains between the data source, intermediate components, and applications at the end of the pipelines.

The remainder of the thesis is divided into two parts: "Systems for Managing Small Data", which describe two framework-level projects for supporting user-facing applications, and "Selective and Informed Sharing", which introduces a system for enablign users to selectively share small data with researchers in the context of research studies.

Part one presents the opportunity to centralize small data applications' concerns in a shared middle layer. Chapter 3 describes an early that manages acquisition and feature extraction from a restricted scope, email; this work was published as "The Email Analysis Framework: Aiding the Analysis of Personal Natural Language Texts." Alquaddoomi, F., Ketcham, C., & Estrin, D. in LinkQS: Workshop on Linking The Quantified Self (LQS 2014). Lifestreams DB, introduced in Chapter 4 is a more complete realization of this middle layer; it was published as "Small Data: Applications and Architecture" Hsieh, C.K., Alquaddoomi, F., Okeke, F., Pollak, J.P., Gunasekara, L. & Estrin, D. in the Fourth International Conference on Big Data, Small Data, Linked Data and Open Data (ALLDATA 2018). This work was co-authored by myself and Cheng-Kang Hsieh; the latter author implemented the back-end system and performed experiments, while I wrote the front-end components and made design decisions about the architecture.

Part two discusses the necessity for individuals to be able to selectively share their personal data. Chapters 5 and 6 introduce our system for securely processing and filtering data from Google's Takeout archive. The former chapter, submitted as "Takeout Processor: a Multi-Modal Data Acquisition and Filtering Pipeline" Alquaddoomi, F., Wen, H., & Estrin, D. to ICDE 2018, introduces the architecture and a pilot study. The latter, "Evaluating the Feasibility of a Personal Data Filtering Interface" Alquaddoomi, F., Tseng, E., & Estrin, D. which is under consideration for NordiCHI 2018, presents an in-depth feasibility study conducted on a more general audience, i.e., Amazon Mechanial Turk users. The conclusion summarizes the works' contributions to the small data space, and outlines what remains to be done.

CHAPTER 2

The Small Data Ecosystem

2.1 Introduction

2.1.1 What are Small Data?

Small data are persistent digital footprints that users create as byproducts of interacting with other people and the world through online services and devices. These byproducts include stored chat histories, location traces, or website usage logs. The user's intent is not to produce these interaction data; instead they occur as a natural result of the other intended action. For instance, when people communicate over email the intent is to exchange information, not to produce a durable log, but the requirement that mail persist until the other party reads it, with no counter-requirement to expunge it, leads to its accumulation. These data accumulate not only within the service providers' databases, but also in web browsers and mobile devices that facilitate interaction with those services; small data from a single intent can end up stored in different forms across the entire path from the user to the service.

We formally refer to the user's wants and needs as **intents**, and the action of using a service to satisfy the intent as a **realization** of that intent. This terminology is intentionally reminiscent of Android's "intents" [And18], indications to the OS that the user wishes to take an action. The interaction data produced from the realization of these intents, as well as any derived information from those interactions, is collectively called **small data**. It should be noted that "intent" is a subjective term. Since technology is deeply enmeshed into our lives, it can be unclear to the user, if apparent at all, when an intent is being realized. For instance, most individuals' phones are constantly collecting their physical location and usage data; while this data facilitates high-level "intents" (e.g., for a service provider to provide location-based recommendations), they may not align with the users' conscious intents.

As individuals increasingly embrace technology, these small data increasingly become approximations of a person's behavior and internal state. Analyzing these small data can be extremely fruitful for organizations – see Amazon's or Netflix's success with product and media recommendation for examples. Companies like Google even offer the analysis of an individual's small data as a product, as in Google Feed [goo17], which surfaces interaction data from email, location history, and web-browsing data to provide recommendations. Seeing products related to one's purchase history, being reminded of appointments, or receiving media recommendations to one are all **derived intents**, realizable through small data. **Small Data Applications** are, then, applications which make use of this interaction data to facilitate the realization of secondary intents with services that use small data to realize those intents.

2.1.2 What is a Software Ecosystem?

The term *software ecosystem* was first used academically in 2005 by [MS05], defined as "Traditionally, a software ecosystem refers to a collection of software products that have some given degree of symbiotic relationships." A more specific definition was provided in 2009 by [Bos09], stating:

> A software ecosystem consists of the set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions.

This second definition, which describes software as supporting the requirements of actors in the ecosystem, is more apt in describing the evolutionary nature of software: as needs arise, software is developed to fulfill those needs, often leveraging or competing against other software in the ecosystem.

[JC13] expands the term "software" in the previous definitions to include a wider family of software-related concepts. In a non-exhaustive list, they mention **standards**, **hardware**,

products (i.e., actual software), and **platforms** all interacting within a software ecosystem. This expansion of the term is useful in the context of small data, since software, standards, platforms, and actors' needs and expectations all co-evolve within the ecosystem.

For the purposes of our discussion, software ecosystems consist of actors, specifically users and developers, who facilitate the realization of the users' intents through software. The software is the evolving entity in the ecosystem; it changes over time as users' intents change and opportunities to realize those intents become available.

2.1.3 The Small Data Ecosystem

Users' needs and the services that satisfy those needs can clearly be described as a traditional software ecosystem. Products emerge onto the market, users make use of them, and the products gradually evolve as users' needs change over time. We refer to this familiar ecosystem as the **primary ecosystem**; in this ecosystem, users' interaction data are incidentally produced, collected, and stored across the heirarchy of systems that users employ in realizing their intents.

The small data ecosystem, i.e. the **derived ecosystem**, underlies this primary ecosystem; it ingests small data generated in the primary ecosystem to fuel the creation and evolution of systems that use small data to realize these secondary intents. The main actors in this secondary ecosystem are individuals who are concerned with making use of small data. This includes the user who initially generated the data, trusted domain experts (for example, coaches, clinicians, therapists, etc.) who may be privy to the data in support of the user, or researchers who are studying behavioral biomarkers in small data. It also includes the software developers who support the requirements of these actors with small data applications and services. Service providers from the primary ecosystem and policymakers play supporting roles; they both determine what small data is initially available, but do not themselves contribute to the ecosystem. Figure 2.1 depicts the percolation of data from the primary ecosystem down into the small data ecosystem. The contents of the dotted box in the figure are the architectural patterns discussed in section 2.3.

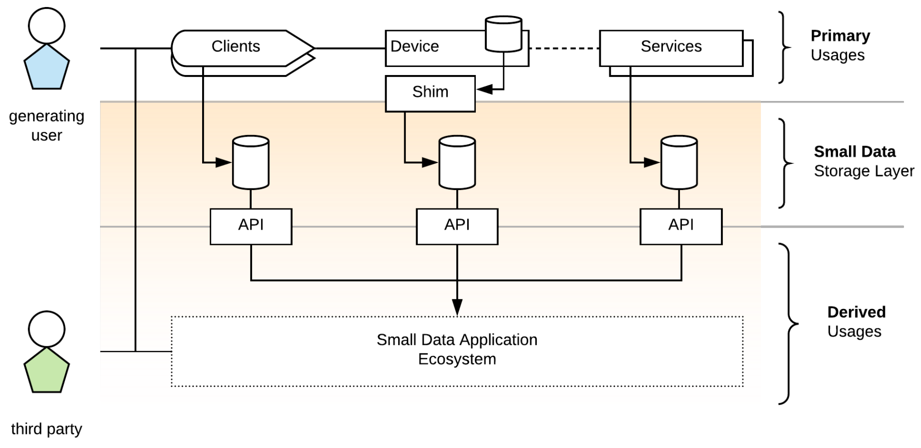


Figure 2.1: The Primary Ecosystem and Small Data Ecosystem.

The entities that are exchanged in the ecosystem are both small data software and derived traces output by that software that are fed back into the ecosystem. Figure 2.2 shows the cyclical relationship between small data and the applications and processes that consume and exhaust new types of small data. In an open ecosystem, derived data streams can spur the creation of new types of software, and vice versa.

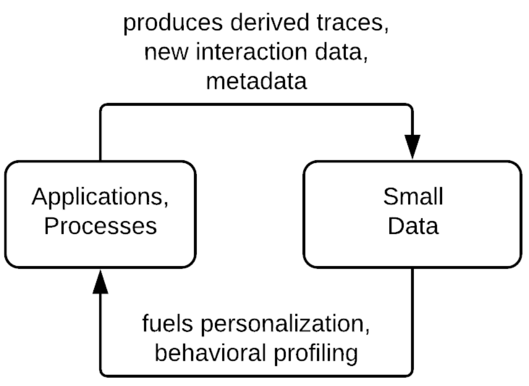


Figure 2.2: The cyclical relationship between Small Data and Small Data Applications and Processes

2.2 Data Dimensions, Application Types

Small data has three unifying qualities: each data point is associated with a point or interval in time, it can take on a variety of forms, and it can be stored at many points in the path between the user and a service. We refer to these attributes as time, form, and origin, respectively.

2.2.1 Data Dimension: Time

Since small data are considered from the individual perspective of a single generating user, **time** is a major axis in the small data space. Aligning by time allows us to capture the set of events that were affecting that user at a given time. Grouping co-occurring events is an essential step to user behavior modeling: since the events co-occur in time and space they all potentially contribute to the user's latent state, which conditionally affects future actions.

Timestamps are not necessarily absolute; for instance, if timestamps are collected with timezone information, the timezone may be inaccurate due to the user traveling and not having set the timezone correctly. A related problem occurs for timestamps that are stored in Coordinated Universal Time (UTC): when modeling user behavior, we may instead want to know the time of an event relative to that user's current timezone, not to an absolute time point. These issues can be partially addressed by joining the data against the user's location trace, from which the local timezone can be inferred.

Data streams that are regularly collected are called **continuous**; examples include heart rate monitors or location traces. Depending on the application, the required frequency for a stream to be considered "continuous" can differ. In constrast, **episodic** data streams are the logs of incidental momentary actions, for example sending an email or accessing a webpage. Data streams may also consist of intervals of time rather than discrete points; a stream of activities, e.g., walking, at rest, or driving, will consist of intervals during which the activities occurred. Aligning temporal streams to establish context requires reconciling the impact of events: an episodic event affects and is affected by a neighborhood of events around it, not just in the moment in which it occurs. Continuous and interval-based streams may similarly have expanded temporal extents, depending on the model that incorporates them into the user's context.

Continuous data may also vary in the granularity with which it is collected, which raises the question of how to join streams with differing granularities. Again, it is up to the end-user application to determine whether it is sufficient for the streams to be aggregated according to the coarsest granularity, or if data can be imputed in the coarser streams to approximate the finer-grained ones.

2.2.2 Data Dimension: Form

When data is emitted from the primary ecosystem, it is often in a "primitive" form; it has not been joined against other sources, clustered, or labeled by machine learning algorithms. As this primitive data percolates through the layers of small data systems, it produces derived data which acquires semantics from its processing: raw accelerometer traces become labeled activities such as "walking" or "driving", GPS points become associated with semantic labels such as "home" or "work". Text may be collapsed into vectors of topic model weights or summarized by its sentiment. Combinations of app usage and mobility streams might be combined into a behavioral biomarker that correlates to a latent psychological state. The **form** of the data is where the data sits on the spectrum of data processing from primitive to refined.

Producing these refined data streams, especially when joining across different streams, involves normalizing them to make them comparable. Efficiently managing the derived streams is a matter of caching. Both of these topics are discussed in section 2.3 below.

2.2.3 Data Dimension: Origin

The **origin** of the data is simply the physical location where it is stored. Since small data are generated, often in parallel, across a variety of systems, from mobile phones to browsers to webservers and web applications implemented on top of them, small data end up accumulating in data stores with varying access mechanisms and security principals associated with the data. In some cases, there is no feasible access mechanism to access the data; for example, sensor data on a mobile phone may be held in device-local storage. If the platform is programmable, then a developer who wishes to expose the data can implement a “shim”, a small local component that pushes the data to a location with an appropriate access mechanism.

The origin of the data affects two system considerations: caching, in the case that the origin is expensive to query, and bridging trust domains in cases where the downstream components are outside of the trust domain of the origin.

2.2.4 End-User Application Types

The small data applications explored in this work fall into three general contexts: user-facing, domain-expert-facing, and researcher-facing. Each context implies different system requirements, discussed below. Do note that this set is not exhaustive: there are potentially as many contexts as there are parties interested in viewing data. Examples of these application types are included in section 2.4.

User-facing Applications in this context expose an interactive interface to the user who generated the data. Since they are interactive, they must support low-latency retrieval from the underlying data sources. The data that they present is summarized into a form that enables the user to act upon the data, implying a deep pipeline of transformations. The user-facing context assumes that the user is both the generator of the data and consumer of the results; this leads to a simpler authentication model, as well as brings in the possibility of the user contributing additional recalled contextual information to the application.

Trusted Domain Expert-facing Similar to the user-facing applications, these applications also present a summary, but in this case to inform an action by the domain expert. Unlike the user-facing context, it is implied that the generating user and the trusted domain expert are not the same person. The application’s authentication model must allow for access delegation to the domain expert. The application should be designed with consideration for the user's privacy, either via selective sharing and/or through an agreement of confidentiality between the user and domain expert.

Researcher-facing Research requires more flexible means of interacting with the data; systems in this context expose a query interface which enables project-specific data summarization and joining to be implemented by the researcher themselves. In the case where the researcher is working closely with the small data application developer, the desired analyses may be implemented into the pipeline directly, in which case the application produces dumps of derived data. Studies are generally composed of a cohort of users, in which case the authorization model must include the concept of users and researchers being associated through the study object.

Developer-facing Like the researcher context, developers are interested in receiving derived data in a structured, machine-readable format, either through a query interface or as a data dump. This context differs from researchers in that analysis is not the final result: instead, the developer creates components with which users or other components interact. Developers also are typically not privileged to view the data; the system must have a mechanism to authorize the developer's component, for example by a user providing identification to a security principal through an interface.

It bears noting that the same individual or group may interact with multiple contexts. For instance, members of the quantified self movement may engage in writing software to collect and analyze their data while also making use of the tools they and others have created; the movement has encouraged discussion of the combined roles of the individual as both subject and analyst, as described in [Swa13a]. A research team conducting a study with an intervention component may act as both researchers and domain experts, as in the FOCUS schizophrenia study where researchers both analyzed smartphone data and provided interventions via a custom app [BBB14].

2.3 System Considerations

The following vignettes introduce situations in which a concern is met with a particular architectural pattern. The section first contrasts stovepipe and componentized architectures, discussing the strengths and weaknesses of each. The discussion is broadened to common concerns for small data systems: normalization, caching, and bridging trust barriers between small data sources.

2.3.1 Stovepipe Architecture

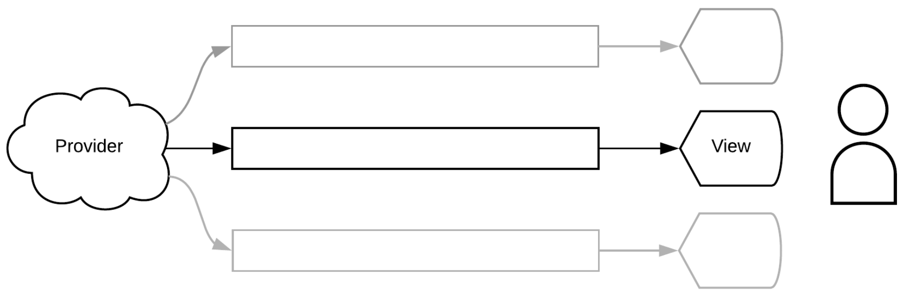


Figure 2.3: Stovepipe Architecture. Parallel architectures that assumedly duplicate functionality are shown in gray.

The stovepipe architecture, consists of a single monolithic pipeline from source to sink. The pipeline's internal components are opaque to the rest of the ecosystem, exposing only the actor-facing interface at the end. The term "stovepipe" implies that opportunities exist to share functionality and data between parallel stovepipes, but the architecture is by its design incapable of taking advantage of them, leading to duplication of effort. Stovepipe architectures often result in data silos, since there is no operational requirement to share data and thus no incentive to spend effort on developing interfaces. Consequently, stovepipe architectures hamper data portability: even in the ideal case where the architecture's data is accessible to third parties, without strictly defined interfaces the structure and semantics of data are prone to changing over time, making it difficult to interpret or move the data outside of its host system.

Despite these drawbacks, stovepipe architectures are easy to implement and modify, and their lack of external dependencies makes them robust. Stovepipe architectures are the overwhelming norm in application development since they assume nothing about the other entities in the ecosystem. They are an appropriate choice in emerging fields, where they can serve as initial research into what functionality might be factored out and shared between future, more modular systems.

2.3.2 Componentized Architecture

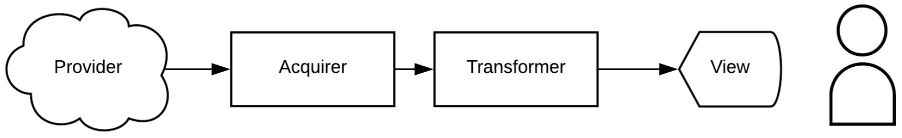


Figure 2.4: Componentized Architecture

In contrast to the stovepipe architecture, a componentized architecture breaks the monolothic pipeline up into discrete, externally-visible components connected by well-defined interfaces. The architectures presented here draw inspiration from the Open mHealth initiative [CHS12], which decomposes the pipeline into three stages: acquisition from a data provider, any number of transformations, and presentation to a data consumer. A small data application consists of a composition of any number of acquirers, transformations which join or otherwise modify the intermediate data, and a user-facing interface. These components form a streaming pipeline, with elements closer to the source referred to as "upstream components" and elements closer to the interface as "downstream components".

There are minor, although notable, drawbacks to using a componentized architecture. Most significantly, components are coupled to both upstream and downstream components. Unanticipated changes in the upstream components may break the downstream components, leading to some synchronization overhead when changes inevitably need to be made. There is also overhead inherent in defining and versioning interfaces. Finally, components which serve multiple downstream consumers must strike a balance between satisfying their various consumers' requirements. In a stovepipe architecture, internal components in the fixed pipeline can be coupled with full satisfaction of their requirements.

2.3.3 Caching

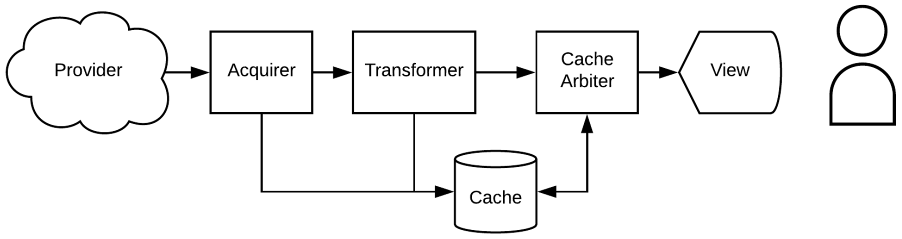


Figure 2.5: Cache layer, which allows the architecture to store expensive accesses, computation results, or indexing schemes.

In user-facing applications fluid interactivity is key. Granted that the user's attention is a resource, there is an acute need to reduce latency between operations. Caching satisfies that requirement by storing copies of data that is anticipated to be used in an intermediate storage. The cache arbiter determines whether data requested by the front-end is in the cache, returning it if so. If not, it requests it from the upstream pipeline and potentially stores it to the cache. The arbiter also evicts data from the cache that is no longer needed according to the caching poicy.

Latency can occur for a number of reasons. The source may be distant from the interface, in which case data that is expected to be used frequently can be cached for faster retrieval. The data in the source may also be indexed under a different assumed usage model than what the application requires; the cache then provides an opportunity to reindex the data for faster access under the actual usage model. Finally, it may be expensive to compute some derived data stream from the source. The cache can ameliorate that delay by memoizing previously-computed results should they be needed again. In all of these cases, the cache acts as a non-critical latency reducer; in the failure case where the requested entity is not present in the cache, it is simply acquired at the full latency cost and returned, updating the cache in the process. This scenario collapses if the source is intermittently accessible; in those cases the source can be replaced with a reliable mirror of the source's contents.

While there are myriad design decisions to make when implementing a cache (for instance, its size, the number of levels if it is a multi-level cache, tradeoffs between these caching levels, et cetera), the most important of them is the eviction policy when the cache is saturated. Generally, for the event-based streams found in small data sources, a good starting policy is to retain events in the time period in which the user has expressed interest. A window around the present is a reasonable default, with the window moving in anticipation of sequential accesses around times that the user has requested.

2.3.4 Normalization

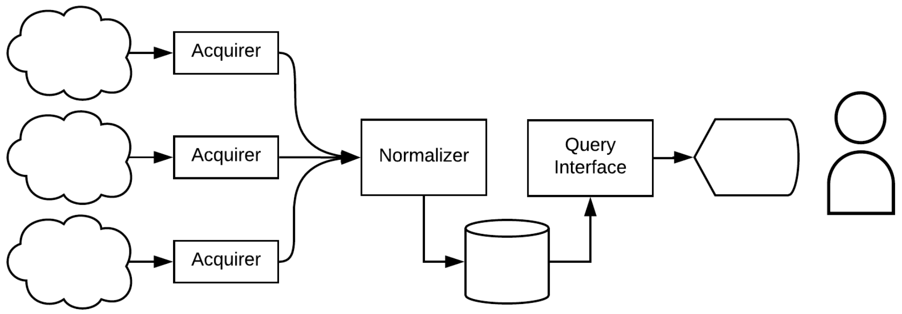


Figure 2.6: Intermediate Normalization Layer. This allows joins to be issued by downstream components between related data streams.

Normalization is principally about enabling joins between streams, which allows data from different domains or modes to be analyzed together. In small data, the main concern is joining on the time axis, then normalizing other columns implicated in the join. In order to perform a join, all the columns implicated in the join need to be comparable. For the time axis, this implies putting all of the timestamps into the same timezone; the semantics of comparing other types is very much type-dependent.

Generally, determining if two data types are comparable requires agreement between the data creator and the party performing the join. Representations that include metadata (e.g., RDF) require less app-specific implementation to ensure comparability, since include embedded references to the ontology in which their comparisons are defined. This occurs frequently in health domains, where two organizations may use different instruments with their own assumptions for collecting a biomarker; the ontology allows measurements of the same underlying concept to be compared, and thus joined.

For applications with a small volume of metadata and a closed set of inputs, it may be sufficient to embed the knowledge required to enable comparisons between the streams in the pipeline or application instead of in the data. For example, a SQL database relies on the application programmer to associate tables of primitive data types together by explicitly defining shared keys; more complicated joins are handled explicitly on a per-stream basis in the applications themselves. As of this writing relational databases such as PostgreSQL are many times more efficient than the leading RDF triplestores; if efficiency is more of a concern than joining an open set of streams, a relational database is a feasible choice.

2.3.5 Bridging Trust Domains

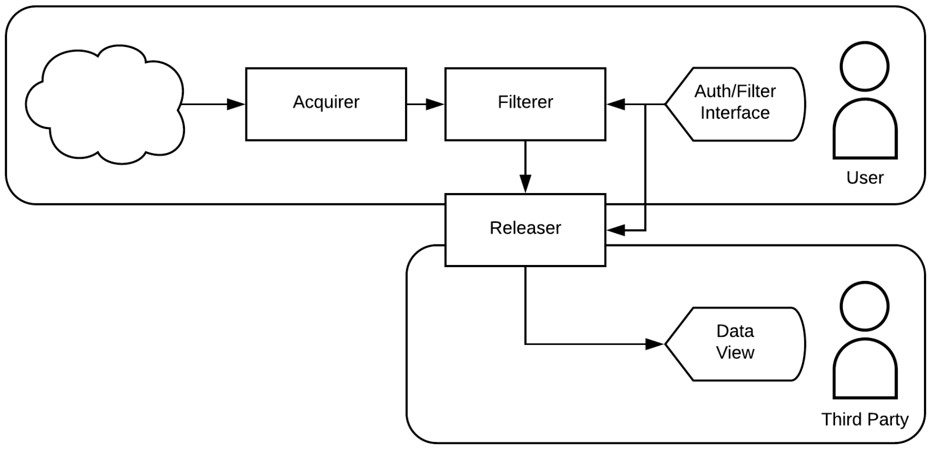


Figure 2.7: Bridging Trust Domains

Any discussion of sharing personal data would be incomplete without addressing the topic of trust. In order to preserve the generating user's privacy, data must be associated with both an access control specification and a security principal, an entity that can be authorized to a system. It is assumed that an organization's internal components exist within an implicit "trust domain"; external components can be admitted into the trust domain by authenticating themselves as a valid security principal for the data that they are requesting. This authentication mechanism can take many forms; traditionally authentication is performed by presenting a shared secret, e.g., a username and password. In small data applications the user rarely wants to grant full permission to act as themselves; instead, the privileged user can provide a "bearer token" to components that they want to grant conditional access to their data. These bearer tokens both identify the component for auditing purposes and restrict the breadth and duration of its privileges to minimize the potential for misuse.

Alternatively, rather than providing access to a personal data stream via authentication and authorization, the trusted domain can provide an interface to a transformed version of the data that is less sensitive than the original. For instance, a sensitive location trace might be transformed into the average amount of time the user was outside their home, a less sensitive statistic which could be shared more freely with untrusted external components.

Figure 2.7 depicts the latter system, in which an intermediate filtering component straddles the trust barrier between two organizations. Under input from the user, it selectively makes privileged and/or filtered data available to components in the other domain. In this case, the user's involvement can either be the granting of a bearer token to the external components in the other domain, or it can be parameters for filtering the data to reduce its sensitivity.

2.4 Existing Applications

The list below contains a selection of existing third-party small data applications. Many of the listed applications are stovepipe architectures yet share inputs or functionality, which further motivates the utility of an open ecosystem of composable processing elements. Table 2.1 breaks down the listed applications by the data dimensions presented in section 2.2.

The last three entries, Lifestreams DB, Takeout Processor, and the Email Analysis Framework (EAF), are our own explorations in the field.

- **Moment:** Tracks mobile app usage for habit management.
- **CrossCheck:** Enables clinicians to monitor and support outpatients with schizophrenia.
- **mPower:** Allows individuals to crowdsource research data relating to Parkinson's disease.
- **Now/Feed:** Integrates recommendations based on one's Google data into a stream of context-dependent items.
- **Amazon:** Suggests products based on prior viewing, purchasing activity.
- **Moves:** Collects raw sensor data and produces semantic location/activity information.
- **ResearchKit, ResearchStack:** Provides a framework for apps to securely collect health-related data.
- **FitBit:** Tracks physical activity and provides summaries, goal-setting tools.
- **Strava:** Tracks bike rides and provides summaries, crowdsourced bike routes.
- **flow-e:** Scrapes email to build a todo list.
- **TripIt:** Scrapes email to build travel itineraries.
- **Metro:** A platform for users to collect and sell their usage data.
- **Lifestreams DB:** Support small data apps by providing normalization, caching, and a query API on top of a user's small data streams.
- **Takeout Processor:** Allows users to release their Google Takeout archive to researchers. Provides exploration and manual filtering options to users. Normalizes the Google Takeout archive and makes it queryable for researchers.
- **EAF:** Parses email and extracts summarizing statistics, stylometrics for downstream applications to analyze.

Table 2.1: Small Data Applications with Data Dimensions

Application	**Time**	**Form**	**Origin**	**Result**	**Context**
Moment	continuous	app usage, phone unlock events	device-generated	app usage over time	user
CrossCheck	continuous, episodic	device sensors, self-report	device-generated, self-reported	expert-facing interface for monitoring need for interventions	domain expert
mPower	continuous, episodic	device sensors, self-report	device-generated, self-reported	persistence to backend datastore	researcher
Now/Feed	continuous, episodic	email, location, browser history, within-app interactions	organization-internal	interface: lists of recommendations, reminders	user
Amazon	episodic	within-site interactions: product viewing, purchasing history	organization-internal	product recommendations	user
Moves	continuous	device sensors	device-generated	semantic location, activity	user, developer
ResearchKit, ResearchStack	continuous	device sensors, self-report	device-generated, self-reported	persistence to backend datastore	developer
FitBit	continuous	specialized hardware sensor	device-generated	fitness dashboard	user
Strava	episodic (continous intervals)	device sensors (location + accelerometer)	device-generated	trip history interface, bike route browsing interface	user
flow-e	episodic	email	integration with gmail	task list	user
TripIt	episodic	email	integration w/many email providers	travel itinerary dashboard	user
Metro	continuous, episodic	browser history, website usage	browser	interaction data	researcher, developer
Lifestreams DB	continuous, episodic	an extensible set of streams; currently supports email, location, accelerometer	integration w/gmail, Moves	SPARQL query interface via HTTPS API	developer

Application	**Time**	**Form**	**Origin**	**Result**	**Context**
Takeout Processor	continuous, episodic	Google Takeout archive; currently parses location, search, browser history	integration w/Google Drive	PostgreSQL query interface, dashboard of analysis modules	researcher
EAF	episodic	email	integration w/gmail	RESTful HTTPS API	developer

2.5 Conclusion

Small data are digital footprints that individuals generate when they interact with the world through online services and devices. These data are increasingly leveraged by 1) service providers to model users' behavior and improve the services they offer, by 2) researchers, clinicians, and other trusted domain experts for staging studies and interventions, and by 3) users themselves to understand and modify their own behavior. This set of derived usages of the data creates a secondary ecosystem, fueled by data from the ecosystem of users, services, and devices, in which small data and the systems that process it co-evolve to serve these secondary purposes.

In this chapter, we define small data, software ecosystems, and how these two concepts are combined to form the small data ecosystem. We describe the actors in the ecosystem as well as the co-evolving artifacts that they exchange, specifically systems that process small data and the raw and derived traces that fuel them. In the remaining sections, we examine the dimensions of small data streams (here defined as their time, form, and origin) and discuss some common small data application contexts. Section 2.3 provides a set of architectural patterns to address the concerns that arise when implementing small data applications. Finally, we provide a list of existing small data applications and how they align to our dimensions.

The remaining chapters describe our explorations in the space of small data systems and applications, specifically three infrastructural components: the Email Analysis Framework (EAF), Lifestreams DB, and the Takeout Processor (TOP). Each project touches upon all of the mentioned architectural concerns mentioned in this chapter: componentization, caching, normalization, and bridging trust domains. While they have some design choices in common (e.g., all are internally composed of discrete parts), they differ in the audiences they serve and thus also in the schemes they employ to overcome these concerns.

Part I

Systems for Managing Small Data

Small data applications as a group share many common concerns: they must all authenticate to and acquire data from personal data sources, perform some kind of processing on that data, and then efficiently present that data through an interface. Nevertheless, a system which handles these common concerns must remain modular, since the data sources which comprise the small data ecosystem are constantly changing and growing.

The first chapter describes the Email Analysis Framework, a mostly self-contained small data pipeline that ingests email, provides fine-grained filtering options, and extracts both linguistic features and content in a more readily machine-usable format, which are exposed to prospective third-party applications through its own OAuth-authenticated API.

The second chapter proposes uniting these common concerns into a "narrow waist" of the small data ecosystem, and introduces Lifestreams DB, an implementation of this proposal. Lifestreams DB is presented as a "DPU container": rather than being an essential component of a pipeline, it allows the architecture as a whole to remain modular, while at the same time opportunistically combining shared data and functionality between applications when possible.

CHAPTER 3

The Email Analysis Framework (EAF)

Accepted for publication in LQS 2014 as "The Email Analysis Framework: Aiding the Analysis of Personal Natural Language Texts."; Alquaddoomi, F., Ketcham, C., & Estrin, D. in LinkQS: Workshop on Linking The Quantified Self (LQS 2014). Included here with typographical modifications.

3.1 Introduction

As we interact with the world, we produce a profusion of data across different modalities. Of particular interest is the data we produce in communicating with other human beings, which could if collected and analyzed provide insight into our relationships with others as well our own internal state. This data often takes the form of free text which by its nature is qualitative, and thus challenging to analyze with quantitative methods. It is also frequently strewn across various services. Some of these services expose the data for public consumption, as in the case of social networking sites like Twitter, Facebook, or posts on personal blogs. Other services are more private, such as email and text messaging, and special care must be taken to gain access to the data as well as to preserve its privacy.

To summarize, the primary concerns are to securely collect, integrate, and analyze this often sensitive qualitative data. This chapter proposes the implementation of a framework, the "Email Analysis Framework" (EAF), that consumes a user's sent email and produces a set of quantitative models and statistics informed by the field of natural language processing. While the scope of the project is currently to collect and process email, the intent is to expand the framework to collect and integrate other sources of free text, for instance from social networking sites. It is hoped that the EAF will be used as a proxy for these qualitative data sources, providing a foundation upon which user-facing tools can be built to derive insights about this data for the individual in a privacy-preserving way.

This chapter principally describes the structure and design choices in acquiring, analyzing, and disbursing sensitive data. Applications are discussed in 3.4, which currently consist of a completed sample EAF consumer that produces trivial visualizations as well as two more significant applications that are currently in development.

3.2 Approach and Related Work

As described in [Swa13b], the overarching intent of quantifying the self is to collect, integrate, and analyze data streams that may be indicative of an individual's physical, emotional, and psychological state. The purpose of this analysis is to promote awareness of how these measurable quantities both affect and can be affected by the individual's state, and to provide support for decisions that change that state. As mentioned previously, free text is both relatively easy to collect and clearly carries much information about how we feel about others and ourselves; indeed, it has been demonstrated that even our choices of words reflect our psychological state [PMN03]. While this data may be present, it is in an opaque form that must be parsed into usable quantitative data.

The analysis of free text has been extensively addressed in the field of natural language processing (NLP). NLP concerns itself with the broad task of comprehending (that is, unambiguously parsing) and extracting structured information from human language, which is accomplished through two main approaches: rule-based (aka grammatical) and statistical methods. The EAF primarily makes use of these statistical methods, specifically n-gram language modeling, to build a sequence of generative models of an individual's use of language over time.

n-gram models are sufficiently descriptive of an individual's use of language that they can be used to discriminate one author from another purely by comparing descriptive statistics computed over them, such as the entropy or the perplexity of the distributions [PSK03, ZZV06]. Descriptive statistics, such as the entropy of a language model mentioned previously, are of special appeal to privacy because they provide an essential determination about the author without compromising the original content from which the statistic was derived.

A user's email is a unique corpus in that each document (i.e. email) is tagged with a host of metadata, including the time it was sent. Thus, computing language models over brackets of emails close in time can provide "snapshots" of the evolution of a user's use of language over time. These snapshots can be compared against each other to determine if there are shifts in the style of the user's written communications which could perhaps correlate to life events. There may be regularities in the changes of these models, or similarities to other people's models with whom the individual interacts. The snapshots can be filtered by recipient or by communication mode to determine if the audience or medium determines the way an individual writes, or if there are detectable groupings. Many more examples could be proposed for these temporal language models, especially when other sources of time-based data (location, activity, calendar events, etc.) are introduced. One of the EAF's main goals is to provide infrastructure to build and maintain these models, as well as allow them, and the descriptive statistics derived from them, to be released at the user's discretion for comparison to other data sources.

There are other frameworks which provide similar analytical capabilities, notably the General Architecture for Text Engineering (GATE) [CMB02]. There are also numerous libraries and toolkits [FL04, Mal10] that include the same features that the EAF provides – in fact, the EAF makes use of the popular nltk library [Bir06] to perform many of its functions. The EAF differs from these projects in its context: it is a deployable system focused on centralizing the secure acquisition and processing of emails for many users. It provides user-facing administrative interfaces to control it, and app-facing APIs to make use of its results. The EAF's intent is to allow users to make sense of their own data, and uses a fine-grained opt-in permission system fully controlled by the user to help protect against malicious or unintended use of the user's email data.

In the context of email analysis, the MIT Media Lab's Immersion project [mit] shares the EAF's goal of using one's email for the purpose of personal insight and self-reflection.

Unlike the EAF, the Immersion project restricts itself to analysis of the user's social group through reading the "From" and "To" fields of email header – no examination of the body text is performed. Further, the output of the Immersion project is an infographic and not raw data that can be reused by other components, whereas the EAF's purpose is to facilitate analysis by other tools.

3.3 Architecture

The EAF's first task is to transform a user's sent email messages into a series of tokens, where each token is tagged with the time at which it was sent. This series of time-tagged tokens constitutes a "stream", from which the n-gram models mentioned previously are built. The stream is quantized into intervals; the ordering of tokens within these intervals is not preserved from their originating messages (outside of their order in the n-grams), with the express intention of making it difficult to reconstruct the original text. After quantization, the stream is then made available at the user's discretion to third-party applications ("consumers"), with the ability for the user to configure per-consumer filters that control what information that consumer can access. A few candidate consumers are discussed in the "Applications" section 3.4. In order to mitigate the danger of storing sensitive user credentials, the EAF makes extensive use of the OAuth2 [HH11] standard, both as an OAuth2 consumer (of Gmail, currently) and as an OAuth2 provider. The use of OAuth2 also allows the user the freedom of revoking access to the EAF should they wish to discontinue its use, or to revoke access to third-party apps that had been authorized to consume the EAF's API. After the initial synchronization, future emails that the user sends are automatically acquired by the system by periodically polling the provider.

3.3.1 Structure

The EAF consists of three main components, as depicted in figure 3.1: a web interface through which the user authorizes access to their Gmail account and performs administrative tasks, a task processor which acquires the user's email and produces a token stream from it, and a second web interface which faces consumers of the token stream. Both web interfaces are implemented in Django 1.7, a framework for rapid development of web applications in Python. Authorization to third-party services is facilitated by Django-Allauth, a project that allows Django's built-in authentication system to interoperate with a host of OAuth2 providers, including Gmail. The task processor makes use of Celery, a distributed task queue processor that is often used in concert with Django. Both components communicate via a shared database, specifically PostgreSQL, which was chosen for its performance under heavy, highly concurrent loads.

The framework exposes a RESTful HTTPS interface to allow third-party applications to consume the token stream. The implementation of this interface was aided by the Django-REST-framework, and the specifications of the interface follow the openmHealth DSU specification v1.0, [omh]. The user-facing web interface makes use of the RESTful interface itself for querying the token stream. In order to allow registered third-party sites to gain access to the user's email data for analysis and visualization, the EAF acts as an OAuth2 provider; third-party sites must involve the user in their request for a temporary access token, which they can subsequently use to make requests on the user's behalf.

3.3.2 User Interaction

Prior to using the system the user first creates an EAF site account which acts as an aggregation point for the multiple email accounts they might want to use. At the moment this account creation is performed automatically when the user authorizes their first email account; the account they authorize (or any other linked account) then implicitly logs them in to their site account, although this behavior is subject to change in the future.

In their interaction with the system, the user proceeds through three stages:

1. **Authorization**, in which the user is prompted to release temporary credentials used to access their email account via OAuth2.

2. **Acquisition**, during which the user monitors the progress of the system as it down-

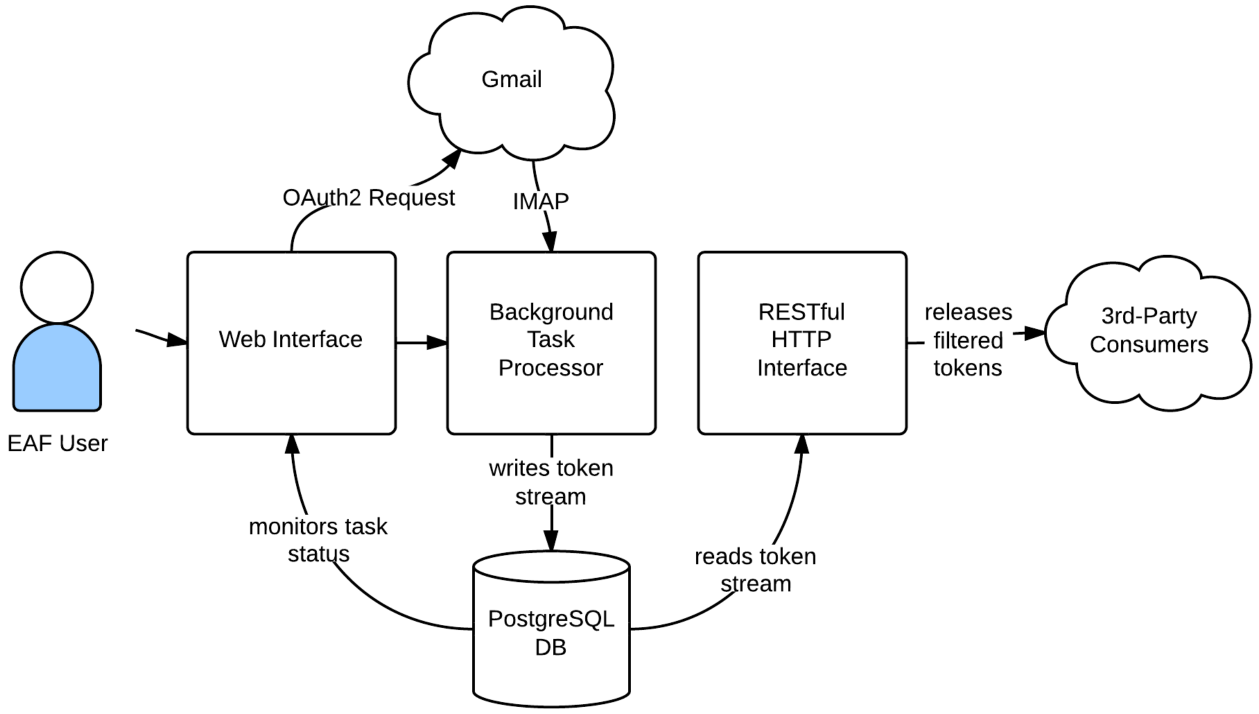


Figure 3.1: Structure of the Email Analysis Framework

loads their emails and performs filtering/transformations before inserting them into the database as a stream of tokens.

3. **Release**, in which the user selects which consumers can access their token stream and what filtering/transformations will be applied for that consumer.

3.3.2.1 Authorization

The authorization stage is initiated when the user visits the web interface. Using a standard OAuth2 handshake, the user is redirected to Google's Gmail authorization page, where they log in (or use a previously logged-in session) and then accept the permissions which the framework requests, specifically access to the user's email. If the user provides their consent, they are returned to the EAF where they can proceed to acquisition. If the user does not provide consent or some other error occurs, they are returned to the framework with an error message and are prompted to try again. Multiple email accounts can be associated with a single EAF site account, in which case selecting an account from the list of registered accounts begins the next stage, acquisition.

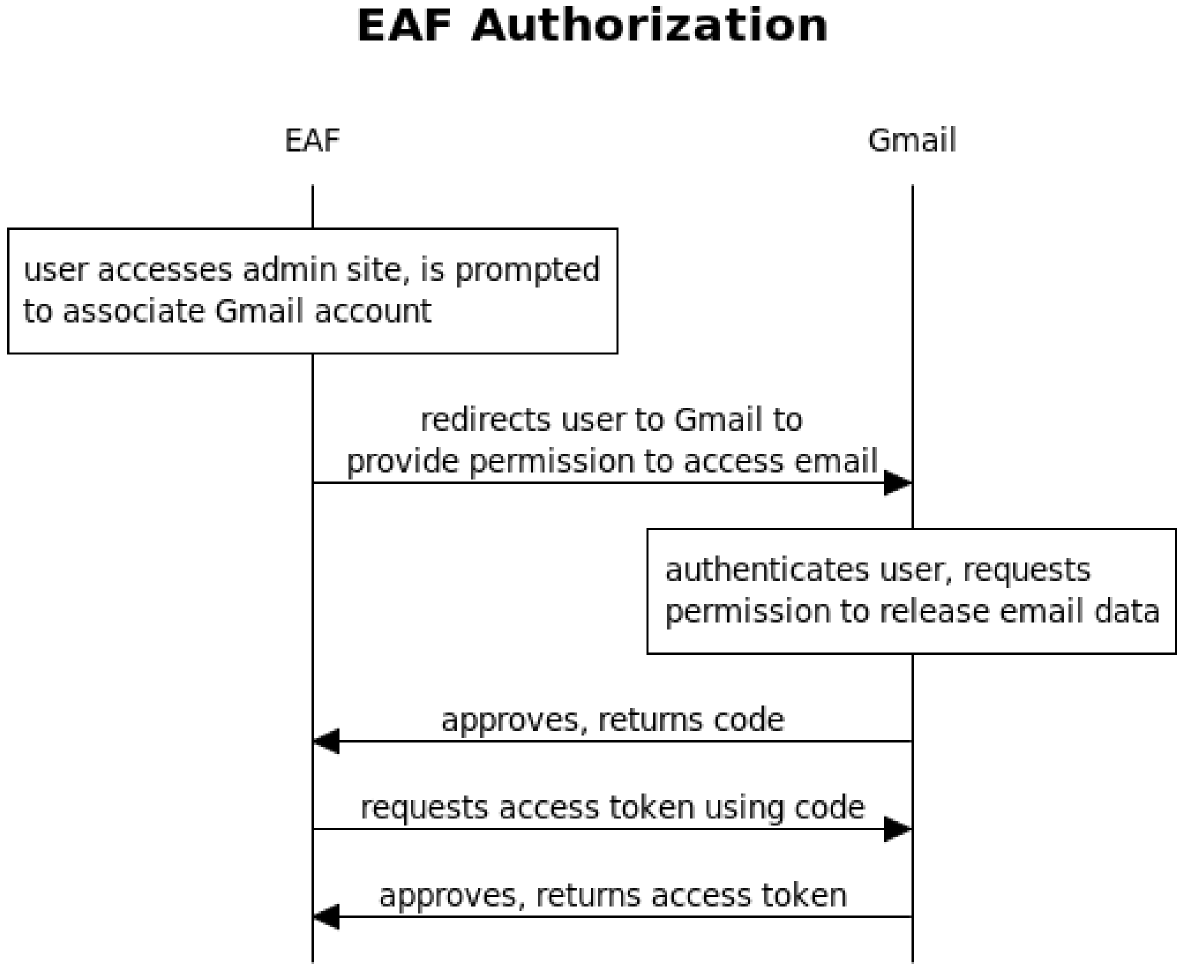

EAF Authorization
EAF
Gmail
user accesses admin site, is prompted to associate Gmail account
redirects user to Gmail to provide permission to access email
authenticates user, requests permission to release email data
approves, returns code
requests access token using code
approves, returns access token

Figure 3.2: Gmail Authorization

3.3.2.2 Acquisition

Initial Acquisition Acquisition starts directly after authorization and is handled by the background task processor. The user is shown a view of the task's progress which is periodically updated. The process can be quite lengthy, especially in the case where there is a large backlog of messages to process, so the user is permitted to close the view and return to the site at their convenience to check in on the task's progress. Upon completion, the framework sends a notification email which includes the total duration of the acquisition task. At this point, the user can view the results of the acquisition process in a small built-in visualization dashboard that shows a few summarizing statistics about their token stream plotted over time. Incremental acquisition tasks that occur after the initial acquisition do not trigger a notification.

Since the framework is intended to model the user's use of language and not the language of other individuals with whom the user is conversing, it is necessary to strip quotations and reply text from the emails prior to processing. Isolating only the user's text in sent mail is accomplished through an adapted version of the email_reply_parser [1] library, developed by GitHub.

Ongoing Acquisition In the background task processor, the acquisition task consists of using the previously-obtained OAuth2 credentials to authenticate to Google's IMAP server. The task then proceeds to download the user's sent email (that is, the contents of "GMail\[Sent Mail]") in chronological order, skipping messages which have been recorded as processed in a previous iteration of the task. Each email is passed through a series of filters, called the "pre-filter chain", which ultimately results in a sequence of tokens that are associated with the email account, the user's EAF site account, and the time at which the email was sent. By default, the first filter in the chain performs tokenization: each email is split on newlines and punctuation into tokens, which are converted to lowercase to reduce the number of possible tokens due to capitalization differences, and stripped of numbers and quotation marks. The second filter is the "ignored words" filter, which allows the user to selectively prohibit certain words from ever entering the database. At the moment, the ignored words must be manually entered, which makes filtering passwords and other sensitive information problematic, given that the ignored list itself is then sensitive. This will be addressed in the subsection on filter types, 3.3.3.

After the filter chain runs, the tokens are then written to the database. Rather than store repeated tokens individually, each token is stored with a count of the number of times it occurred within its message. If same token occurs in different messages, it is stored separately for each message. This choice was made as a compromise between allowing for flexible choice

[1] `https://github.com/github/email_reply_parser`

of the interval into which tokens are combined when the stream is consumed and consuming less space in the database; if the system were designed with a fixed interval rather than a flexible one, the tokens would simply be combined into a histogram for each interval.

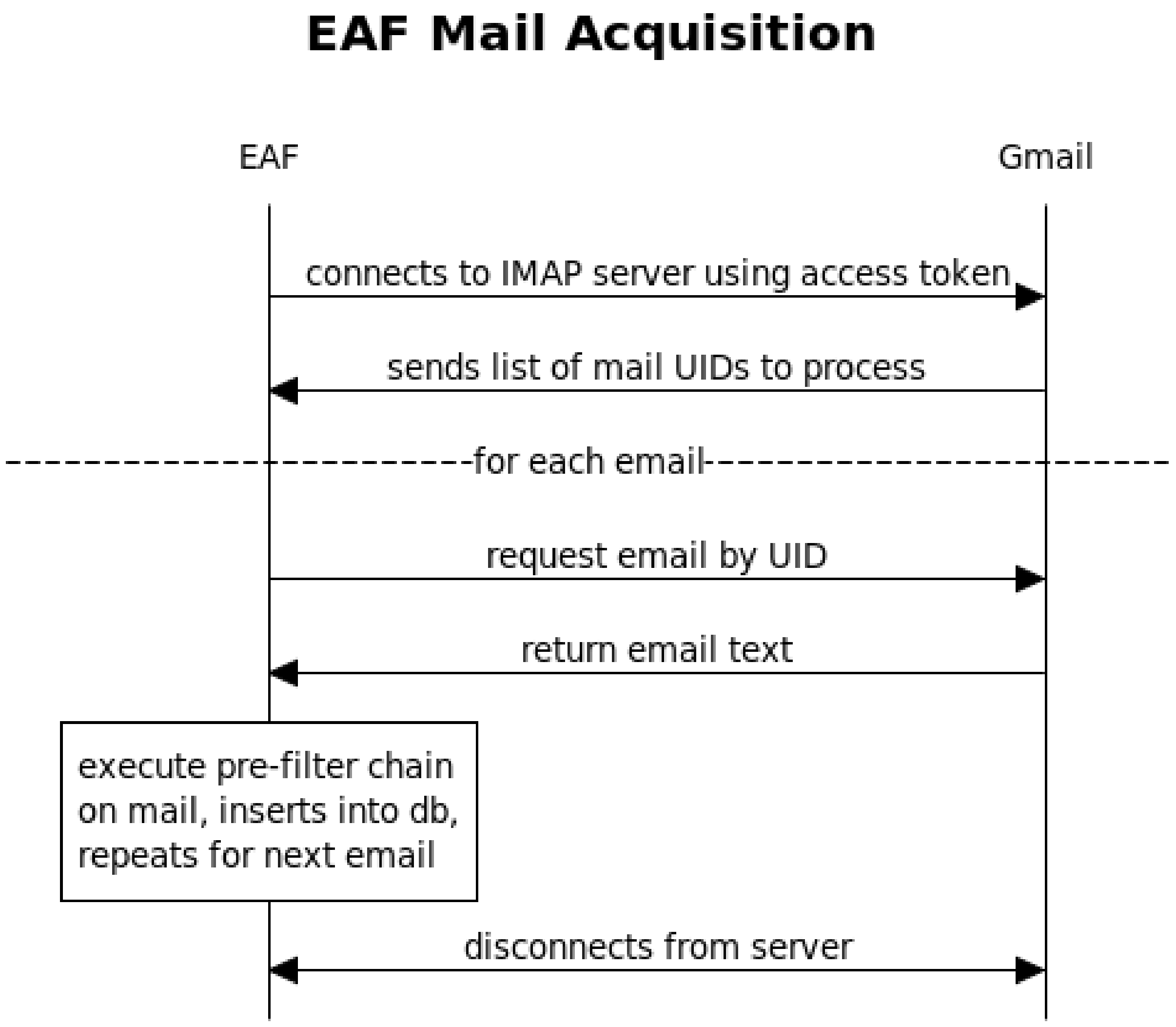

EAF Mail Acquisition
EAF
Gmail
connects to IMAP server using access token
sends list of mail UIDs to process
for each email
request email by UID
return email text
execute pre-filter chain on mail, inserts into db, repeats for next email
disconnects from server

Figure 3.3: Mail Acquisition

3.3.2.3 Release

Once the user has found an EAF-compatible application, they can authorize that application to access their token stream via OAuth2. In this stage the EAF acts as an OAuth2 provider, providing a page to which the third-party application can redirect the user to be prompted for authorization of their identity via Gmail (used also as credentials to access their EAF data) and permission to access their token stream. In the case where the user has multiple token streams, they will be prompted to choose the streams to which they are granting access. On this page, the user selects a filter chain for each stream that will be used specifically with this consumer, or simply opt not to filter the stream at all. The process is detailed in figure 3.4.

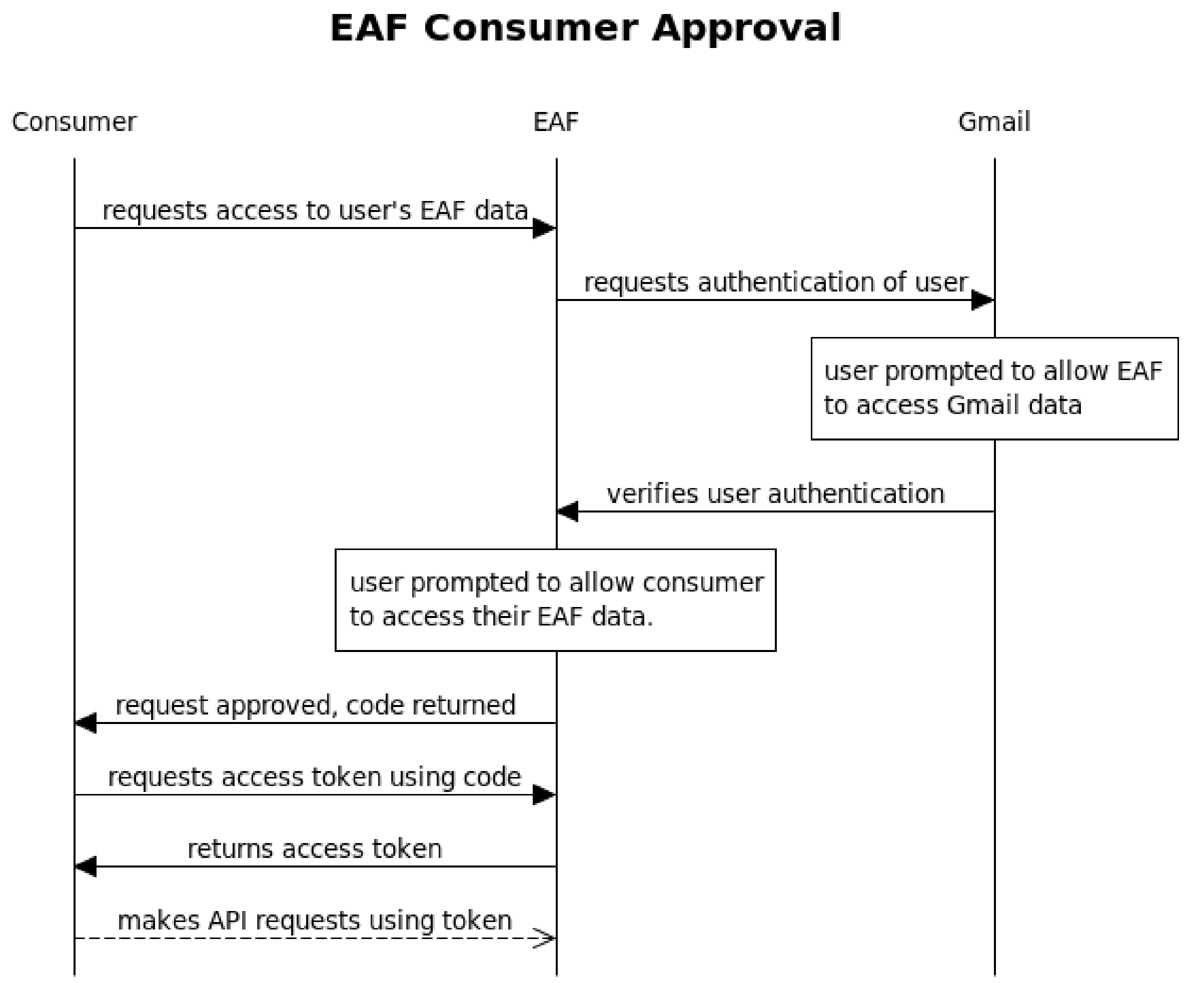

EAF Consumer Approval
Consumer
EAF
Gmail
requests access to user's EAF data
requests authentication of user
user prompted to allow EAF to access Gmail data
verifies user authentication
user prompted to allow consumer to access their EAF data.
request approved, code returned
requests access token using code
returns access token
makes API requests using token

Figure 3.4: Release to Consumer

After this point, the consumer can request updates from the token stream at any time. The EAF audits all accesses, displays the last time of access, and allows the user to revoke the consumer's access at any time or change the filter chain associated with that consumer.

3.3.3 Filtering

As previously mentioned, both the acquisition and release stages employ a sequence of filters that allow the input data to be selectively screened for sensitive terms and otherwise transformed to better suit the requirements of the consumers. The acquisition stage's filter chain is referred to as the "pre-filter chain" and the release stage's is the "post-filter chain". There is only a single pre-filter chain, but there can be as many as one post-filter chain for each registered consumer.

The pre-filter chain always has a special "tokenize" filter as its first filter, which produces the initial sequence of tokens for filtering and transformation, and may only be used in the pre-filter chain. A second special filter that may only be used in the pre-filtering step is the "ignore word sequence" filter, which ignores the sequence of tokens configured in the filter, and was initially created to ignore signature blocks. This filter can only function in the pre-filtering step as the exact sequence of the tokens is lost upon insertion into the database.

Aside from the special "tokenize" filter, there are a few other filters which can only be used in the pre-filtering step, namely:

- **Parts-of-Speech Tagger**, which replaces each token with its detected part of speech (noun, verb, etc.)
- **Fork**, which produces an identical stream to the current one, but with its own sub-filter chain. The tokens that are produced from a fork are tagged with a unique ID corresponding to that fork.

The "fork" filter is especially useful in conjunction with the part-of-speech tagger, as both the original text and the parts-of-speech stream can be either individually released or released together, which allows for analysis of the user's grammar. Note that the parts-of-speech stream does preserve the order of tokens in the original stream, but not the text of the tokens themselves.

The filter framework is modular, with the potential to add new filters easily in the future. At the moment, a few parameterizable filters are implemented to change the case of tokens, strip specific characters, and to remove words that are either on a user-specified list or not contained within aspell's "en_US" dictionary. Detecting and ignoring named entities is a work in progress.

- **Change Case**, which transforms the case of the tokens;
- **Strip Characters**, which can be used to remove numbers and other special characters;
- **Ignore Listed Words**, which removes tokens on an "ignore" list from the token stream; and
- **Ignore Non-Dictionary Words**, which removes tokens not found in a common English dictionary

By utilizing the "ignore words" filters, the user is allowed fine-grained control of both the contents of the EAF's database and the views of the token streams presented to different consumers.

3.4 Applications

As mentioned, third-party applications can gain temporary access to a user's data for the purpose of visualizing or otherwise processing it. Granting this access is currently at the user's discretion; the user should make use of the per-consumer post-filter controls to limit the release of sensitive information to potentially untrustworthy parties. Consumers sign requests to the EAF's RESTful JSON API with an access token, obtained through the process described in the "Release" section above 3.3.2.3.

3.4.1 Example: Mail Visualization Front-End

In order to demonstrate the functionality of the framework, a visualization front-end was developed that consumes the framework's API and produces a few example visualizations. The front-end also serves as a reference implementation of EAF client authentication; it first requests permission from the user before gaining access to their token stream. The visualization front-end currently offers the following modules:

- **Word Cloud** - a "word cloud" infographic that can be viewed on a per-week basis (figure 3.5).
- **Rhythm** - a table of the days of the week as the columns and hours of the day as the rows is colored according to the number of emails sent within each interval, with darker colors corresponding to more emails sent (a heatmap, essentially; figure 3.6).
- **Alters** - a bar chart of the number of people contacted per week; when a week is selected, displays a bar chart of emails sent to each person.

These visualization modules are intended to be a jumping-off point for more useful visualizations to be constructed, which would ideally incorporate data from other sources to strengthen inferences about the user's overall state.

3.4.2 Pulse and Partner

In addition to the sample application discussed above, our group is currently developing two applications that make use of statistics computed against the user's email. The first is "Pulse", which makes use of location traces from the user's smartphone as well as the frequency and variety of individuals with whom one communicates to compute a score that indicates **how** rather than **what** the individual is doing. This score is visualized as a waveform over a short window of time (i.e. a week), which can be shared with family members and friends. The second is "Partner", which is intended to measure the degree to which linguistic style matching occurs among individuals who interact with each other face to face, a fairly well-documented phenomenon [NP02a], [ISE11]. Partner makes use of the location traces of two or more individuals as well as computed statistics over their emails to produce two scores, a "proximity" and a "language-style matching" score, which will be visualized as individual timeseries. A third timeseries will display their computed correlation over time.

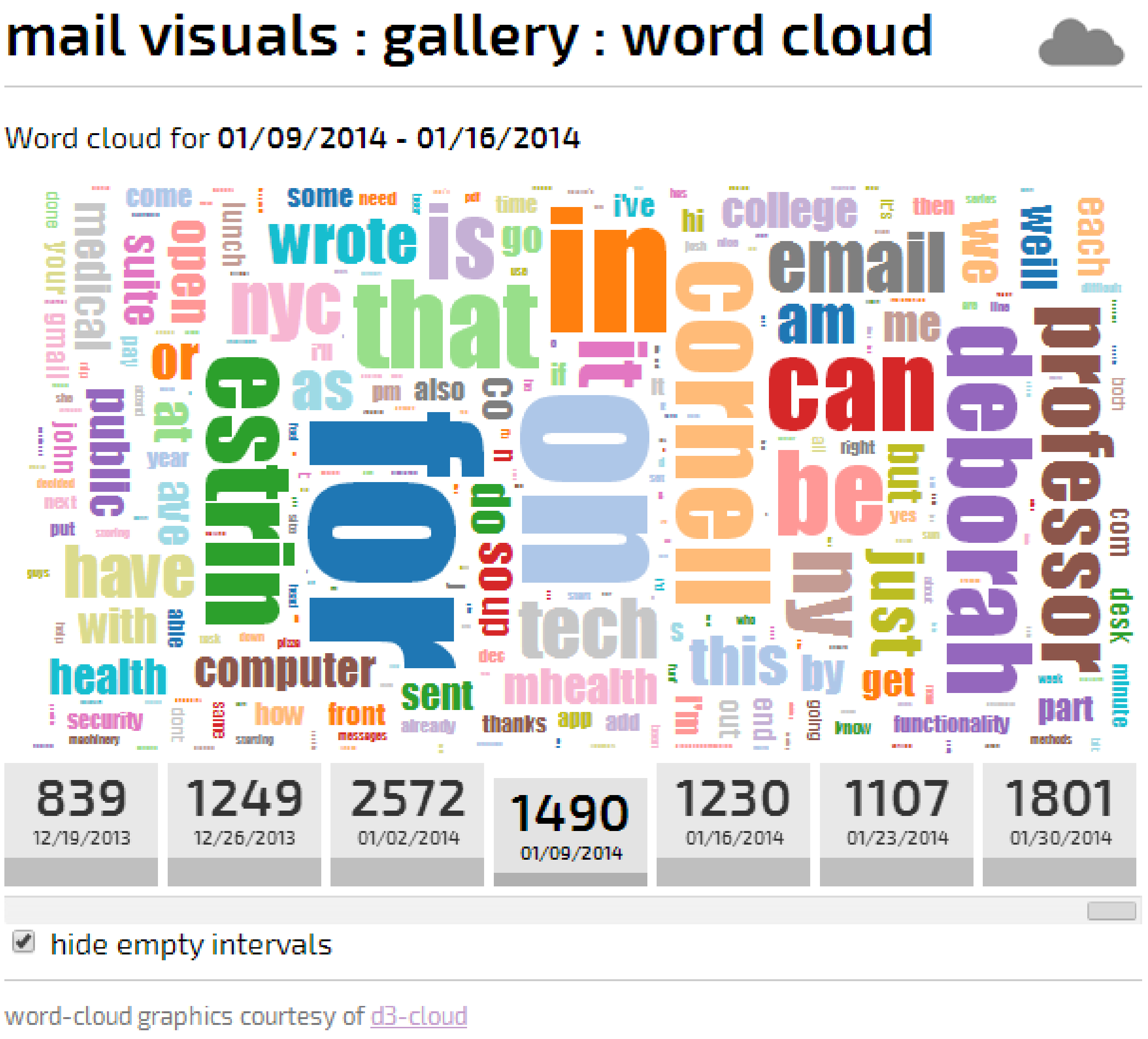
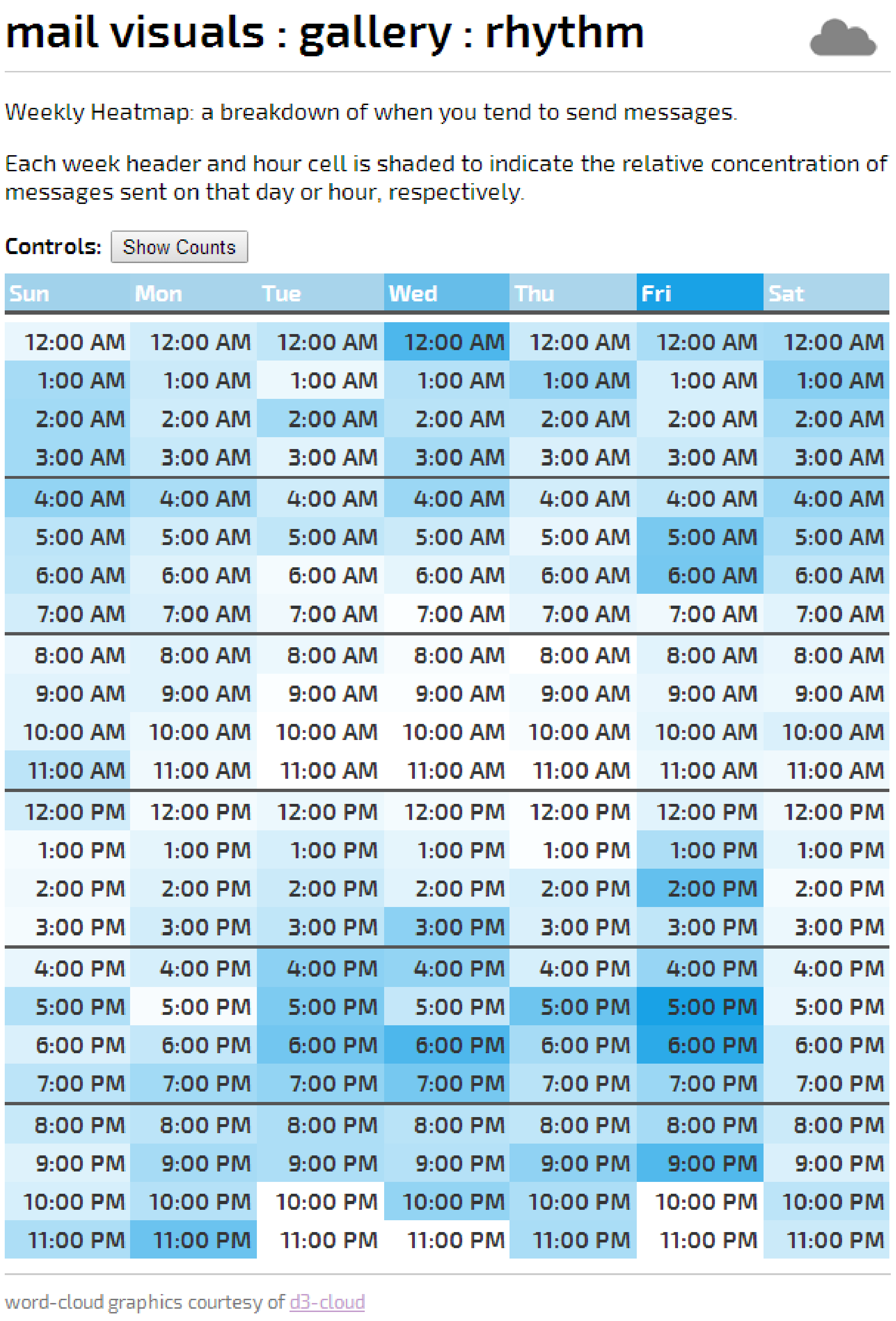


Figure 3.5: "Word Cloud" Visualization

3.5 Conclusion, Future Work

The Email Analysis Framework, a system for extracting structured, easily-consumed data from time-tagged natural-language text was proposed in this work. At the moment it is limited to acquiring text from Gmail, computing, and exposing language models to other tools via a RESTful HTTPS API, but it is hoped to be extended to other sources of personal natural-language text, such as Facebook and Twitter streams. A few candidate visualizations were described to both demonstrate how the data could be used and to stimulate investigation into more novel applications.

In terms of future work, there are extensions planned to all the stages of the framework. As mentioned, the scope of providers is intended to be expanded to other text providers, which will allow analysis to be performed on how different media affect the language model. Additional streams can be extracted in the processing phase, such as identifying named entities and topics, all of which can be analyzed over time, audience, etc. Industry-standard information extraction techniques such as autoslog [RP04] could be applied to discover meeting arrangements, events that occur to named entities or topics mentioned in the emails, and so on. Sentiment analysis could be computed and exposed as another temporal stream, to attempt to model the user's disposition as a function of time. Additional third-party applications are planned, such as a tool for determining points of inflection in the descriptive statistics computed on the language model, and a tool to easily correlate other time-based data against the statistics streams.



mail visuals : gallery : rhythm

Weekly Heatmap: a breakdown of when you tend to send messages.

Each week header and hour cell is shaded to indicate the relative concentration of messages sent on that day or hour, respectively.

Controls: Show Counts

Sun	Mon	Tue	Wed	Thu	Fri	Sat
12:00 AM	12:00 AM	12:00 AM	12:00 AM	12:00 AM	12:00 AM	12:00 AM
1:00 AM	1:00 AM	1:00 AM	1:00 AM	1:00 AM	1:00 AM	1:00 AM
2:00 AM	2:00 AM	2:00 AM	2:00 AM	2:00 AM	2:00 AM	2:00 AM
3:00 AM	3:00 AM	3:00 AM	3:00 AM	3:00 AM	3:00 AM	3:00 AM
4:00 AM	4:00 AM	4:00 AM	4:00 AM	4:00 AM	4:00 AM	4:00 AM
5:00 AM	5:00 AM	5:00 AM	5:00 AM	5:00 AM	5:00 AM	5:00 AM
6:00 AM	6:00 AM	6:00 AM	6:00 AM	6:00 AM	6:00 AM	6:00 AM
7:00 AM	7:00 AM	7:00 AM	7:00 AM	7:00 AM	7:00 AM	7:00 AM
8:00 AM	8:00 AM	8:00 AM	8:00 AM	8:00 AM	8:00 AM	8:00 AM
9:00 AM	9:00 AM	9:00 AM	9:00 AM	9:00 AM	9:00 AM	9:00 AM
10:00 AM	10:00 AM	10:00 AM	10:00 AM	10:00 AM	10:00 AM	10:00 AM
11:00 AM	11:00 AM	11:00 AM	11:00 AM	11:00 AM	11:00 AM	11:00 AM
12:00 PM	12:00 PM	12:00 PM	12:00 PM	12:00 PM	12:00 PM	12:00 PM
1:00 PM	1:00 PM	1:00 PM	1:00 PM	1:00 PM	1:00 PM	1:00 PM
2:00 PM	2:00 PM	2:00 PM	2:00 PM	2:00 PM	2:00 PM	2:00 PM
3:00 PM	3:00 PM	3:00 PM	3:00 PM	3:00 PM	3:00 PM	3:00 PM
4:00 PM	4:00 PM	4:00 PM	4:00 PM	4:00 PM	4:00 PM	4:00 PM
5:00 PM	5:00 PM	5:00 PM	5:00 PM	5:00 PM	5:00 PM	5:00 PM
6:00 PM	6:00 PM	6:00 PM	6:00 PM	6:00 PM	6:00 PM	6:00 PM
7:00 PM	7:00 PM	7:00 PM	7:00 PM	7:00 PM	7:00 PM	7:00 PM
8:00 PM	8:00 PM	8:00 PM	8:00 PM	8:00 PM	8:00 PM	8:00 PM
9:00 PM	9:00 PM	9:00 PM	9:00 PM	9:00 PM	9:00 PM	9:00 PM
10:00 PM	10:00 PM	10:00 PM	10:00 PM	10:00 PM	10:00 PM	10:00 PM
11:00 PM	11:00 PM	11:00 PM	11:00 PM	11:00 PM	11:00 PM	11:00 PM

word-cloud graphics courtesy of d3-cloud

Figure 3.6: “Rhythm” Visualization

CHAPTER 4

Toward a Unified Middle Layer

Accepted for publication in the ALLDATA 2018 conference proceedings as "Small Data: Applications and Architecture" Hsieh, C.K., Alquaddoomi, F., Okeke, F., Pollak, J.P., Gunasekara, L. & Estrin, D. Included here with typographical modifications.

4.1 Introduction

Small data are "digital traces", records of our activities that are stored as we interact with the world around us. These traces are passively produced when we use tools and services that maintain logs: credit cards, grocery receipts, websites and other streaming content services, browsers themselves, etc. They can also be intentionally produced and tracked by wearable sensors, including mobile phone applications. It is well-known that service providers derive value from this information – usage metrics and demographic information, all personal data, are routinely employed to help direct advertisement and optimize products. We argue that this data can and should provide value for the producers of this data as well. As a natural extension of prior ubiquitous computing applications, **small data apps** will emerge as an important class of ubicomp applications that concern themselves with deriving insight from personal data at the user's request and with their oversight.

For example, a small data app may promote healthier eating by coaching users to take the planning actions needed to prepare meals at home. The app would utilize grocery and online food delivery history, browser history, and Moves or Foursquare data to build a model of meal preferences. The user could then receive prompts at their desired frequency about which recipes they are likely to enjoy, and suggestions for additions to their grocery shopping list to enable them to prepare these meals at home. The app could incentivize this with informative comparisons of calorie and cost savings, or could be tied to more intentional gamification. Another small data app could allow independent living elderly to share how they are doing without sharing every detail of what they are doing. The app would make use of passively collected small data streams such as email, activities, and mobile phone usage to create a personalized model of the user's activity, well-being, and degree of social engagement. Rather than exposing the model itself, the app would expose deviations from the model to make family and friends aware of changes to a person's state without divulging detailed information. Such an app can support many types of relationships, including family and friends separated geographically, or other support-network relationships such as social workers, caregivers, and coaches. We describe these concepts in greater detail in section 4.3.

The central role of a small data architecture is to facilitate application-level access to a person's diverse information sources on their behalf. While individual service providers, such as Google, Facebook, and Amazon each have information about many aspects of our behavior, they are limited in how specifically they personalize by the terms of their end-user licensing agreements and a need to preserve users' trust. They also do not each have access to all data of interest. Because of this, there is an opportunity in the market for providers to give users access to their individual data in various forms (application programming interfaces, downloads, email receipts), and for third-party products to emerge that integrate with that user's data in the same way that third party mobile apps make use of mobile-device data. These third party apps would serve the end user without degrading the large-service provider's position, and in fact have the potential to solidify the user's sense of the service provider's utility and trustworthiness. Note that we are promoting that users be given access to their data and not making any statement about data ownership. We are also not addressing the very important policy question regarding service providers making user data available to third parties directly.

As mentioned, service providers have difficulty providing apps that cut across multiple data sources or mine too deeply into their users' data. In contrast, a small data app leverages the user as the common denominator, and can take advantage of the trend for service providers to support application programming interfaces (APIs) for individuals to their data. The user has both the access and authority to collect and aggregate data across these providers, allowing for powerful and comprehensive insights that, by virtue of the fact that they are initiated and consumed by that same user, can be much more focused in their oversight and suggestions. We anticipate and favor broad provision and adoption of systematic programmatic access to personal data for the end users. However, the need for a small-data application architecture need not wait for, nor will it be obviated by, future developments. Already, today, users can obtain access to their data, albeit through idiosyncratic and sometimes ad-hoc channels: e-receipts, diverse APIs, browser plug-ins, etc. Even with access to these data, infrastructure is still required to process these traces into formats that are useful and actionable to the individual. Since most individual users do not develop their own software, we are targeting support for small-data app developers who will implement apps on the behalf of this growing user base; just as they have driven the development of third party apps for smartphones [Per10]. This approach is aligned with the emerging Social Web activities in W3C [Hal14].

Our vision is to create a **small data ecosystem** in which small data apps can be readily developed and deployed atop an infrastructure that standardizes their inter-operation and addresses concerns that are common across apps, such as helping to ensure security and reducing redundancy in storage and computational resources, as well as resolving policy/legal questions that are outside the scope of this work. The vision is, again, driven by the individual as the common denominator, and rightful beneficiary, of access to their data.

We describe the core components of a small data architecture using three exemplar applications, and present a specific system-design for the most central of these components – Lifestreams Database (hereafter "Lifestreams DB"). Lifestreams DB is designed to extract and process diverse digital traces from various sources and make them available to the client applications for further analysis or visualization. **Data interoperability** is an important requirement for such a system as it allows one to gain insights from the combination of data that were originally locked in their own data silos. Lifestreams DB extracts raw data from these data silos, and transforms them into a standardized Resource Description Format

(RDF) that allows one to join these digital traces against each other and with external RDF data sources (e.g., fuse nutrition information with users' online shopping records.)

Unlike many enterprise settings, small data differs in the fact that most of original sources (e.g., Google, Facebook, etc.) persist users' data in their own databases and individually provide security and access control. Therefore, it may be wasteful, or even harmful to the users' security and privacy for Lifestreams DB to permanently replicate these data in one place. Motivated by this distinction, we propose a **soft-state design** that, while providing client applications with virtual access to all the data, only caches a part of it locally, and reproduces the rest on demand. Such a design introduces two important advantages in the context of small data. First, our soft-state model discourages our system from becoming a data "honeypot" that attracts attacks from malicious entities since only a limited amount of information is cached in the system at any given time. Second, it requires much less storage and allows the system to scale to serve a large number of users or integrate with more diverse information beyond its storage capacity. We also provide a mechanism that encrypts sensitive data at rest to further protect the user.

After introducing related work in section 4.2, we present three small data applications in 4.3 and use them to identify cross cutting application requirements. We provide a brief overview of our architecture in section 4.4, then go into depth on the main contribution of this work, Lifestreams Database (DB), in section 4.5. Section 4.6 contains the results of performance analyses for simulated workloads on a sample of simple and complex query types. Finally, section 4.7 provides some observations and outlines future work.

4.2 Related Work

Small data are fueling a new genre of personalization technologies. Recommender systems have been some of the most successful applications in this domain to date as evidenced by recommendations for music in Pandora, consumer goods in Amazon [LSY03], articles in Wikipedia [CFT07], and locations in Foursquare [Fou]. These systems rely heavily on the users' application-specific histories, such as queries, clicks, ratings, and browsing data that result from interacting with their product. Small data can enable far more immersive recommender systems that take into account a larger space of user needs and constraints. In particular, they can benefit from user models derived from both more diverse and longitudinal data (e.g., features and dynamic patterns in: daily travel patterns, consumption from gaming to dining, interests and sentiment expressed in personal communication, etc.). General-purpose recommendation frameworks such as MyMediaLite [GRF11] and LensKit [ELK11] (to name a few) could make use of small data to learn these kinds of broad user models, but they require a front-end component to fetch user's data and drive the framework with appropriately-formatted inputs.

Small data's goal of providing individuals with transformative insights into their behavior is aligned with that of the Quantified Self (QS) movement [Swa13c]. In QS studies, individual experimenters engage in the self-tracking of biological or behavioral information using commercial devices such as Fitbit and myZero sleep trackers, or personal testing services such as 23AndMe, and many systems have been developed to help integrate and visualize QS data [Fit14]. Even prior to QS's popularity, research projects such as Ubifit and BeWell demonstrated the potential of making personal data actionable [CMT08, LLM12]. More recent work, i.e., EmotionCheck [CAJ16], has demonstrated that not only QS data itself, but a user's trust in the tool, can serve as effective leverage for behavioral change. Small data, however, differs from earlier studies in its focus on harnessing data that are (a) generated as byproducts of interacting with services and (b) that are readily available, versus having to be manually collected or otherwise procured. These data can be complementary to or serve as a proxy for some of the data that QS studies collect.

Small data are also related to Personal Information Management (PIM) systems [Jon07]. This line of work covers a broad range of environments from desktops [SGK06, CDH05], to connected-devices in the home [SSC09, GSP14], to e-learning [Rus] and health information management systems [App, Mic, Epi, go]. Our work is complementary to these systems' focus on information organization and retrieval, by providing support for third party applications that would generate additional inputs to these systems through the processing of small data streams that are not yet accessible.

Small data shares similar data input with Personal Automation Engines. For example, Atomate [VMK10] is a system that integrates individuals' social and life-tracking feeds into a unified RDF database, and automatically carries out simple tasks (e.g., messaging) when the incoming feeds satisfy user-defined rules. The service "If-This-Then-That" (IFTTT) [IFT14], expanding on the same idea, compiles a large set of feeds that monitor various online and offline activities and can trigger a wide set of actions when a user-defined condition on a feed is satisfied. On a more application-focused and user-local level, PrefMiner [MHM16] monitors on-device notifications from numerous sources to identify which notifications are important to the user or not. Small data differs from these services in its emphasis on providing insights that require longer-term observation, rather than performing transient event-driven actions. This fundamental distinction results in rather different system requirements, particularly in resource management and security as mentioned in the introduction. That said, our small data application architecture could enable a richer set of inputs to both of these systems.

Our aims are similar to existing systems that provide a modular computational infrastructure and mediate the release of processed personal data, such as openPDS and Virtual Individual Servers [MSW14, SLC11]. While these systems do provide personal data acquisition, storage, and release, they do not explicitly address the problem of normalizing and joining disparate data streams under a shared ontology. Our work complements these systems in providing data modeling and interoperability required to join multiple data streams, as opposed to simply providing analysis of individual data streams.

4.3 Small Data Applications

A small data application is an application that operates on multiple personal data streams, produces some kind of analysis of these streams, and presents the result to the user via an interface. Personal data can include static data, for instance the individual's genome or family lineage. We focus particularly on temporal data, either regular or episodic, that must be continually collected and analyzed. The reason for this focus is twofold: first, these information-rich data sources will be most transformative in creating detailed user

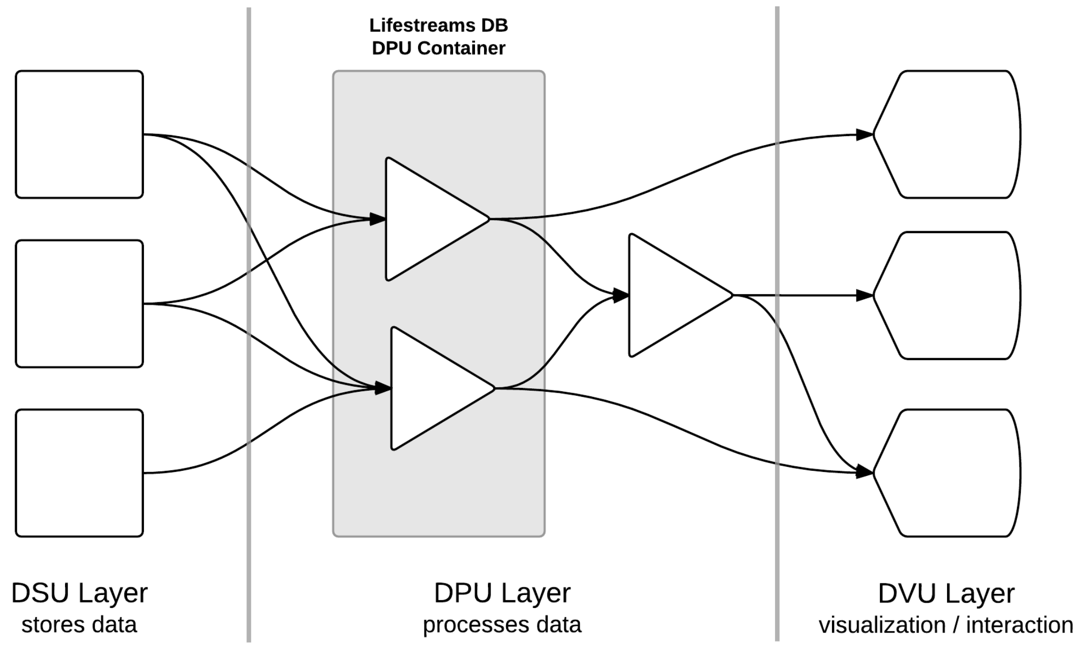


Figure 4.1: Small Data Architecture: illustrates the flow of data between Data Storage Units (DSUs), Data Processing Units (DPUs), and Data Visualizations Units (DVUs, e.g., apps).

models and feedback for diverse applications, and second the temporal data are the more difficult to manage since it is constantly accumulating. Of course, our focus on temporal data does not obviate the value of joining the user's data with other non-temporal data sets - e.g., summarizing nutritional exposure using temporal grocery receipts and relatively-static nutritional databases.

Below, we motivate the requirements of our software architecture using three exemplar small data apps. These applications comprise two data access modes – background and foreground. In the background mode, the application may periodically access a long history of user data to build or update the user's behavioral model. In the foreground, the user experience tends to be based on a more recent window of time, interpreted in the context of the behavioral model.

4.3.1 Ora

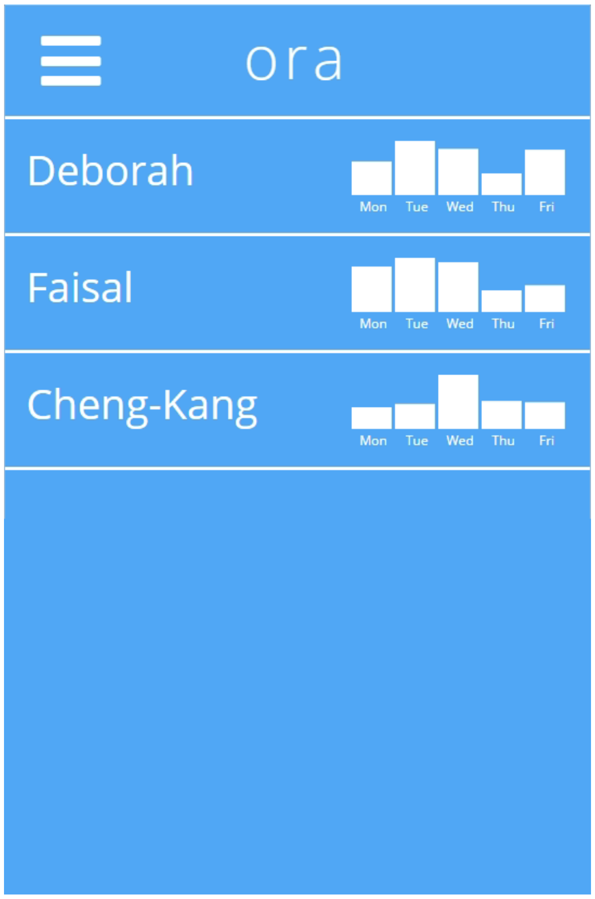
Ora (Figure 4.2) is a tool for sharing how you are doing – without sharing the details of what you are doing – with family, friends, or other people who might be part of your support network (counselors, coaches, etc.) Users interact with Ora via a mobile-optimized website, where they authorize the app to connect to their Gmail and Moves accounts using an OAuth2 grant. Ora extracts descriptive numeric features from these data sources and uses them to build a baseline model that represents the user's usual values for each feature. Deviations from this model are calculated on a per-day basis and summarized into a single numeric value, referred to as a *pulse*, that acts an opaque indicator of the degree to which the user is deviating from the model.

Specifically, the pulse is computed from 20 features extracted from the users' data, including their **geodiameter** (the distance between the furthest two points in their location trace for the day), **exercise duration** (the number of minutes the user was walking or running), **time not at home** (the amount of time not spent at their primary location, typically their home), and the **number of emails sent** in a day. Then, for a set of features F, the baseline for each $f \in F$ is computed as a tuple consisting of the sample standard deviation and mean over a two-month sliding window. For a given day, the pulse (P) is then computed as the sum of the numbers of standard deviations from the mean for each feature.

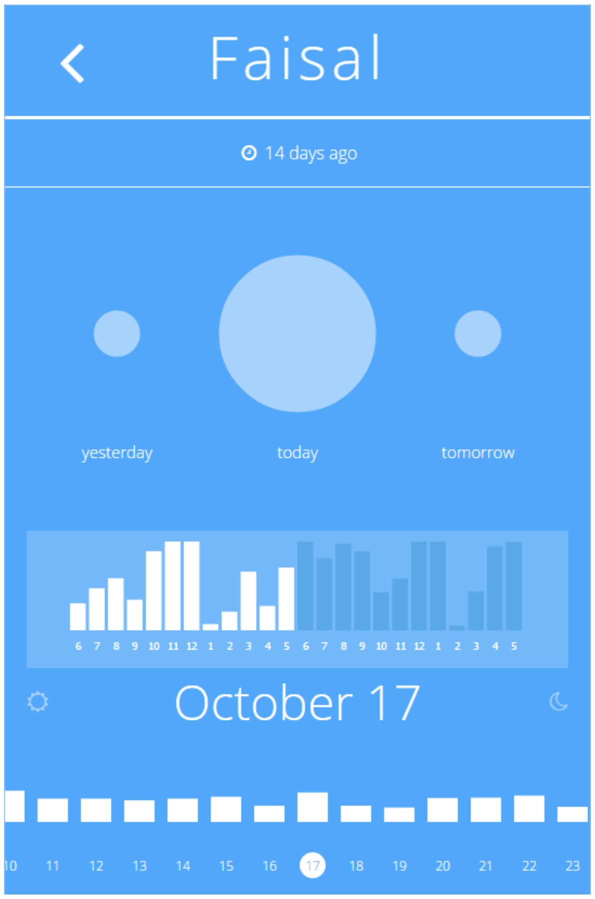
4.3.2 Pushcart

Pushcart (Figure 4.3) uses receipts from services such as FreshDirect or Peapod to determine the nutritional value of the food that a household purchases. This information is provided to a "Wizard of Oz" system in which a clinician, masquerading as a learning algorithm, reviews the purchasing habits of each household and suggests substitutions of more nutritional items during future purchases.

The system's primary source of input is email – after opting in, users register the system to automatically receive a copy of their receipt email, from which the list of items is extracted and then joined against a database of nutritional information for each food item. The user



(a) Ora: List



(b) Ora: User Details

Figure 4.2: Ora: User List and Details View

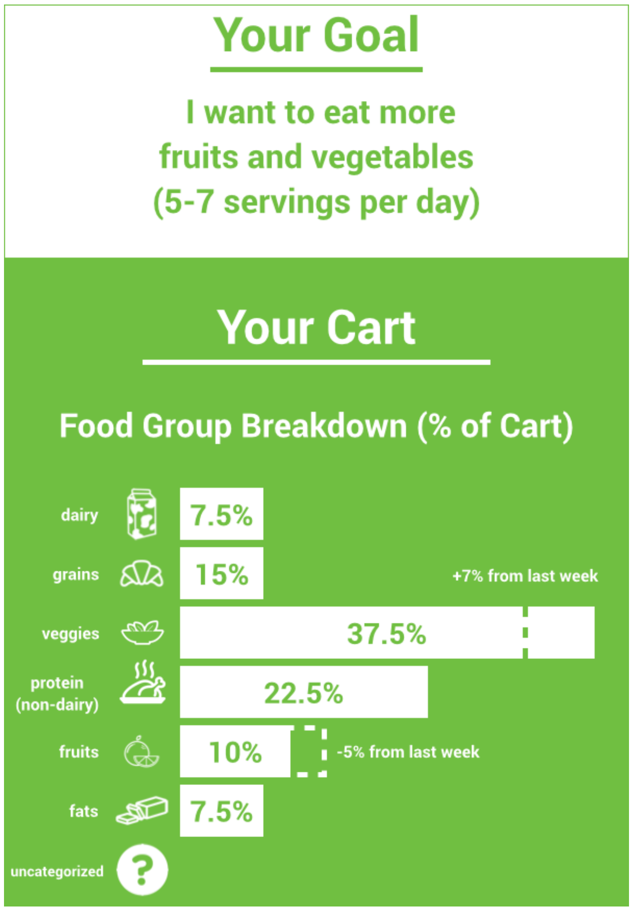


Figure 4.3: Pushcart: Weekly Email Report

interacts with the system through email as well: the user interface is a weekly "report email" that shows a breakdown of purchases in terms of nutritional value, and includes the nutritionist's suggestions.

4.3.3 Partner

Partner is an exploratory app designed around the hypothesis that people who spend time together tend to mimic each others' language patterns, and that the extent of this mimicry is an indicator of good relations; this is a phenomenon known as *linguistic style matching* [NP02b]. The application uses both Gmail and Moves as its data source. After users have registered for the system, it passively collects their email and location data, building a retrospective view of the time they spend physically proximate to each other, the degree of linguistic style matching evidenced by similar values for descriptive metrics used in authorship identification, and the correlation of the two aforementioned values.

Partner relies on a few standard metrics used in authorship attribution, specifically entropy [GZZ13], stylometrics such as the percentage of personal pronouns and ratio of functional words to non-functional words (the "information density"), and the index of qualitative variation (IQV, specifically, the Gibbs M1 index) which serves as a measure of the variability of the user's vocabulary. Each of these features is computed over a categorical distribution of the user's tokens, which is produced from the concatenation of a user's emails into week-long intervals to compensate for the sparsity issues that email presents.

4.4 Architecture

Our architecture is inspired by the concept of a "mashup", an application that merges multiple disparate data sources into a single interface. We started with the typical web mashup, in which data are acquired, processed, and presented solely by and at the client. We then factored out the acquisition and processing into distinct, reusable modules which can be run in the cloud and potentially consumed by multiple clients. Common concerns, such as caching, access control, and data normalization, are provided as system-wide services. While it would be feasible to implement the acquisition and processing components as tightly-coupled, one-off solutions for a single mashup, the redundancy of doing so for each additional app has lead us toward a centralized and reusable architecture.

The architecture is composed of three layers, as depicted in Figure 4.1. There are three main entities: **Data Storage Units** (DSUs), **Data Processing Units** (DPUs), and **Data Visualization Units** (DVUs). These terms mirror the open mHealth standard [ope18]. DSUs include service provider APIs, e.g., Google's numerous service APIs and Facebook's Graph API. DSUs can be accessed directly from DPUs/DVUs, but are often accessed through a "transforming" DPU that converts the API's often proprietary data format into the schemas we use in small data apps. Data flows from DSUs through arbitrary compositions of DPUs – so long as their input and output types are compatible – and terminates in the DVUs.

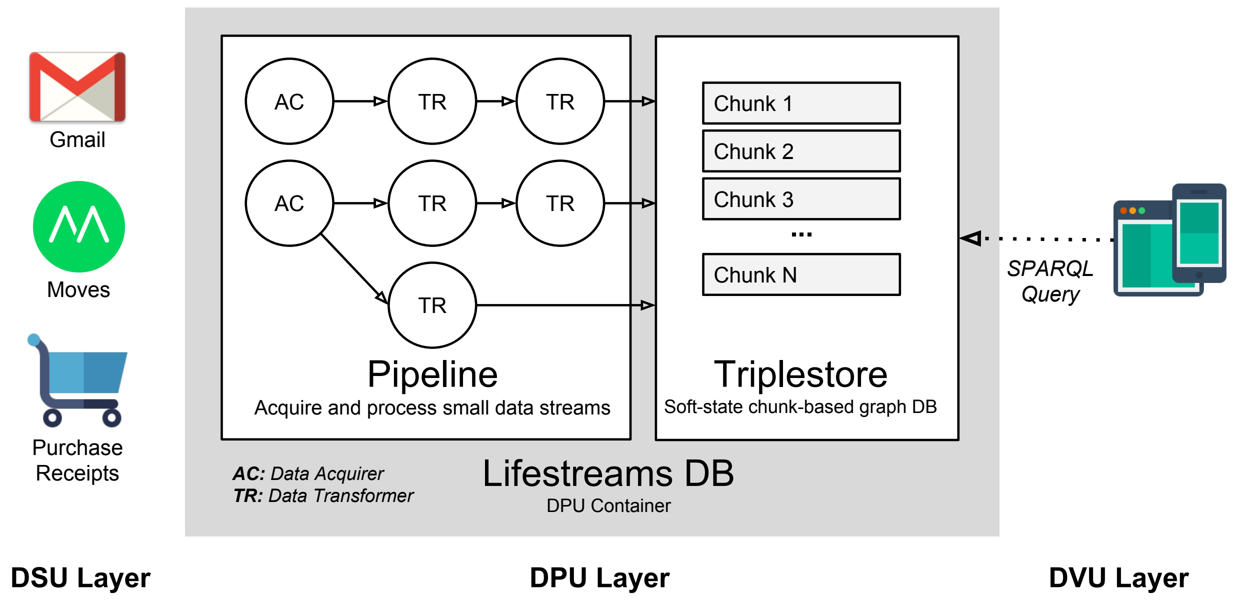


Figure 4.4: Lifestreams DB Pipeline: consists of a set of DPU modules that acquire and process data from various small data DSUs.

Lifestreams DB acts as a container for DPUs, and provides caching, data modeling, access control, and a unified query interface. Its outputs can be directly consumed by DVUs, or by other DPUs that provide additional data processing capability.

This modular pipeline approach is necessitated by the fact that our system will never be complete; there will always be new data sources and means of processing and displaying data, which the architecture should readily accommodate. Further, the implementation of its components is a collaborative effort and we wish to encourage developers to reuse and build upon existing components.

4.5 DPU Containers: Lifestreams DB

Lifestreams DB is an important component in our architecture. Positioned between data sources and small data apps, Lifestreams DB is designed to be the “narrow waist” of the small data ecosystem that provides a unified interface for querying, combining, and fusing diverse small data streams.

Lifestreams DB contains a pipeline of DPUs that Extract, Transform and Load (ETL) an individual's digital traces from different sources using common software APIs and schemas to enable diverse small data applications. Figure 4.4 illustrates the architecture of Lifestreams DB. On the left is ***Lifestreams Pipeline***, a data processing pipeline that contains a set of reusable DPUs that extract raw data from different small data sources and transform raw data into structured, readily usable information. For example, raw actigraphy and geolocation sensor samples from a mobile app are transformed into structured data that describe the time, location, speed, and distance of each activity episode. These extracted data are loaded into ***Lifestreams Triplestore***, an RDF datastore built on top of Jena TDB [The14], that exposes an integrated view of all the diverse RDF data for apps to query. We made two principal design decisions when designing Lifestreams DB: 1) to model data using RDF, and 2) to utilize a soft-state system design. The rationales behind these design decisions are described in the following.

Using RDF for interoperability Data interoperability is key to the success of such a system. Raw data extracted from different data silos need to be transformed into a compatible form to allow one to derive knowledge from them. In Lifestreams DB, we utilize RDF to enable data interoperability. Each DPU outputs data in JavaScript Object Notation (JSON), and the DPUs at the final stage generate RDF data in the JSON-LD format, which will be transformed into RDF triples (i.e., *subject-predicate-object*) before stored in the Triplestore. The advantages of using RDF are as follow. First, it eliminates the need to define database schema, unlike, for example, in a Structured Query Language (SQL) datastore. Data generated by different DPUs are inherently interoperable if the DPUs follow the same ontology to model the data. This property is of significant benefit to a small data ecosystem, since it allows DPUs developed by different people to be plug-and-play without the need to modify the system's database schema. Also, any client application developer, given the ontology, can compose queries to filter, join, and aggregate various types of data generated by different DPUs without knowing specific implementation details such as table and column names, etc.

A Soft-State System Design Architecturally, one major difference between an individuals' digital traces and an enterprise's operational data is that an individual's data are mostly persisted and protected in each original data source's databases (e.g., Google, Facebook). In many cases, there is no need, and is actually wasteful and harmful to the users' security and privacy, for Lifestreams DB to replicate all these data in one place. Thus, we propose a soft-state design that, while providing the client applications with virtual access to all the data, only caches a small portion of it in the system. Data which the user owns (e.g., sensor data from the user's phone or wearable) can be considered in the same way, except that it will reside on a personal DSU instead of in an external organization.

The advantages of this design are three-fold: First, a soft-state design requires much less storage to serve the requests, and thus allows the system to scale more effortlessly to serve a larger number of users and integrate with more diverse information beyond its storage capacity. Further, it enables elastic storage provision, where a service provider can provide the service with less storage (at consequently lower cost), and increase the storage provision only when better performance is needed. Second, it makes the system more robust, since there are less points where critical data loss can occur. If the system needs to be brought down, it can be done so without concern over maintaining important state. Third, a soft-state design inherently has better security properties. Since only a small amount of information is cached in the system at any given time, the exposure of any single data breach is limited. In addition, the fact that the data can be repopulated into the database on-the-fly allows us to encrypt sensitive data and only decrypt them when they are demanded.

These advantages do not come without a price. A soft-state system tends to incur much overhead in indexing, reproducing, and reloading data. In Lifestreams DB, we reduce these overheads by utilizing a chunk-based data management strategy that generates and manages data in chunks. Our design is particularly suitable for applications that perform timeseries-based analysis with temporal locality where subsequent accesses tend to access records that are near in time (in our scheme, in the same chunk.) Within these assumptions, we have improved Lifestreams DB's query performance by multiple factors (compared to the base Jena TDB triplestore) and made it perform even better than a hard-state system that stores

Table 4.1: Data Modeling Type Assignments

Data	**Source**	**Subject Types**	**Object Types**
Location/Mobility	Moves API [Mov]	Stay/Travel	Place
Email	Gmail API	Send/Receive	EmailMessage
Purchase	Gmail API	Buy	Product
Calendar	gCal API	Join	Event
Web Browse	Android API	Browse	WebPage
App Usage	Android API	Use	MobileApp
Phone Call	Android API	Call/Receive	Person
Message	Android API	Send/Receive	SMSMessage

all the data with only a fraction of storage space.

In the following, we first describe our RDF-based data modeling approaches and demonstrate its advantages using the SPARQL queries for the real-world small data applications we are developing. Then, we describe the chunk-based management strategy and the techniques we used to realize the proposed soft-state design.

4.5.1 Data Modeling

When modeling data using RDF, one needs to follow a certain *ontology*. In small data, the concepts we come across most often are the various *actions* performed by users, such as sending emails, making purchases, etc. We chose schema.org [Sch18] as the main ontology rather than the other competing candidates, such as Activity Streams [Sne15], for its semantic action type system. Schema.org defines a hierarchical type system that describes different (sub)categories of actions. At the root is **Action**, a generic type that describes the common properties of an action (e.g., agent, time, etc.). It is then subclassed by more specific types, such as **MoveAction**, which, in turn, are subclassed by more specific types, such as **ArriveAction**, **DepartAction**, etc. This hierarchical structure enables one to write queries to reason across different types of actions within specific categories. For example, an app that encourages better sleep hygiene may analyze users' before-sleep routines by querying certain action categories (e.g., the ExerciseAction and all its subclasses) that occurred before the sleep period.

Table 4.1 summarizes eight different kinds of data we have extracted and modeled from four different data sources, based on schema.org's ontology. The purchase records are derived from email receipts on an opt-in basis. The phone-based data are uploaded to ohmage, a mobile sensing DSU. In the following, we demonstrate how our data modeling approaches can satisfy the requirements of the small data applications described previously with simple SPARQL queries.

Ora: Listing 4.5 shows a snippet of Ora Query that computes the geodiameter and the number of emails sent in a day. For brevity, the snippet omits the part that limits the time range to a single day. The first part of the snippet computes the geodiameter by selecting the maximum distance between any pairs of places at which the user stayed. The second part of the query counts the number of **SendAction**'s of which the targeted object is an email. This example is intended to demonstrate how much an application developer can achieve with Lifestreams DB using a succinct and easy to understand query. Also, this example demonstrates how heterogeneous data streams (i.e., Location/Mobility and Email) are modeled and queried in an interoperable and standardized way.

Pushcart: Listing 4.6 shows a snippet of the Pushcart query. It demonstrates Lifestreams DB's interoperability with an external food nutrition database. A RDF dump of the United States Department of Agriculture (USDA) nutrient database is pre-loaded into a separate triplestore [USD]. The query joins the individuals' grocery purchase records with the entries contained in the USDA database using a free-text matching based on the product names, and select the amount of carbohydrates and protein contained in each of the purchased items.

Partner: Partner is an example of an app which, in addition to Lifestreams DB, requires a more domain-specific DPU. It relies on Lifestreams DB to compute the amount of time two participants spent together based on the distance between where two users stay (see Listing 4.7) and uses the Email Analysis Framework (EAF), a DPU for email language analysis [AKE14], to evaluate language style matching. It is also an example where an application can query from not only one but across multiple users' data with RDF *named graphs* that refer to each user.

```
PREFIX schema: <http://schema.org/>
SELECT * {
  SELECT (MAX(?dist) AS ?geodiameter)
  {  ?stay_x a schema:StayAction;
             schema:location ?loc_x.
     ?stay_y a schema:StayAction;
             schema:location ?loc_y.
     BIND (
        fn:distanceInMeter(?loc_x, loc_y) AS ?dist
    ).}
  SELECT (COUNT(?send) AS ?mail_count)
  {  ?send a schema:SendAction;
           schema:object ?object.
     ?object a schema:EmailMessage.}
}
```

Figure 4.5: A short snippet from Ora query that computes the geodiameter and the number of emails sent.

```
PREFIX text: <http://jena.apache.org/text#>
PREFIX usda: <http://data-gov.tw.rpi.edu/vocab/p/1458/>
SELECT *
{  ?action a schema:BuyAction.
   ?action schema:object ?product.
   ?product schema:name ?product_name.
   SERVICE <http://localhost/usda/endpoint> {
    ?food_item text:query
        (usda:shrt_desc ?product_name 1).
    ?food_item usda:carbohydrt ?carbon;
               usda:protein ?protein.
   }
}
```

Figure 4.6: Pushcart Query joins an individual's food purchase records with the corresponding nutritional information contained in the USDA nutrient database.

```
PREFIX fn: <http://lifestreams.example.org/customFn#>
PREFIX users: <http://lifestreams.example.org/users#>
SELECT (SUM(?overlap) AS ?co_present_time)
{  GRAPH <users:Bob> {
      ?stay_x a schema:StayAction;
      schema:location ?loc_x.}
   GRAPH <users:Alice> {
      ?stay_y a schema:StayAction;
      schema:location ?loc_y.}
   FILTER(fn:distanceInMeter(?loc_x, ?loc_y) < 50)
   BIND (
    fn:overlappingTime(?stay_x, ?stay_y) AS ?overlap
   )
}
```

Figure 4.7: Partner Query computes the time two users spent together based on their location data. Each user's data are referred to by their *named graph*.

4.5.2 Chunk-based Data Management

As mentioned, Lifestreams DB's soft-state design is made possible by a chunk-based strategy. The basic idea behind this strategy is as follows: The DPUs in Lifestreams Pipeline generate data in chunks and load them into Lifestreams Triplestore, which maintains an index to all the chunks (including the ones that are not cached in the system). When a client application submits a query, it will additionally submit a meta-query that selects the chunks it desires. If a chunk selected by the meta-query is not currently available in the system, Lifestreams Pipeline will re-run the corresponding DPUs and reproduce the chunk on the fly from the source. The chunks that contain sensitive data (determined from the data source and the user's preferences) will be encrypted and decrypted on the fly when requested by a query. The chunks are encrypted with 256-bit Advanced Encryption Standard (AES).

Our strategy allows a system to maintain only a small amount of information (i.e., the chunk index) while providing access to a much larger amount of data than can be accommodated by the system's storage capacity. In the following, we describe three major designs that realize this strategy and discuss several query optimization techniques enabled with chunking that can be utilized to provide a better user experience.

4.5.2.1 Chunk Index Design

The chunk index needs to be carefully designed to avoid unnecessary chunk reproduction. For each chunk of data, we extract the following features as its index:

- Distinct object types in the chunk.
- Start time and end time of the aggregate timespan.
- Geo-coordinates of a convex hull that covers all the spatial features in the chunk.

The rationales behind these choices are as follow. First, most of our applications are interested in certain types of actions or objects (e.g., CommunicationActions or ExerciseActions) so object types are a natural choice for indexing. Also, most of small data are time-tagged, and the applications we focus on tend to involve analysis of time series and aggregation based on time or location. The proposed chunk indexing scheme satisfies these requirements.

4.5.2.2 Lifestreams Pipeline: a Reproducible Pipeline

We adopt a functional approach to allow Lifestreams Pipeline to reproduce arbitrary chunks of data from the original sources. The Lifestreams Pipeline consists of two types of DPUs: **Acquirers** acquire raw data from the sources while **Transformers** transform data from one form to another. These DPUs are treated as passive functions invoked by the system. Consider a simple pipeline where one Acquirer and one Transformer linked in sequence. In each iteration, the system invokes the Acquirer with a *state variable* that indicates the chunk we want the Acquirer to fetch. After fetching the corresponding chunk, the Acquirer will return the chunk along with a new state variable that indicates the subsequent chunk to be acquired in the next iteration. The system then invokes the Transformer to transform the chunk, and stores the output chunk along with the state variable. When the chunk is removed, the state variable will be preserved in the system. Therefore, when we need to reproduce the chunk, we just need to re-run the pipeline with the preserved state variable.

An assumption we make here is that the raw data are permanently persisted in the original data sources (i.e., DSUs), and can be re-acquired by the Acquirer at any time. If this is not the case, a *shim* can be implemented to transfer the data to a DSU with such properties (such as Amazon S3). Unlike some chunk-based systems where the chunk sizes are pre-determined, Lifestreams DB allows each Acquirer to decide the chunk sizes according to the characteristics of the APIs it acquires data from. A typical chunk size is daily as it is supported by most data sources. However, as the state variable is updated by the Acquirers themselves, Acquirers can have state variables with different formats or granularity (e.g., hours, weeks.). This feature is important for small data where one usually needs to work with a large variety of external data sources over whose APIs it has no control.

4.5.2.3 Two-Level GDS Chunk Replacement Policy

Similar to many cache systems, Lifestreams DB requires a replacement policy to select chunks for replacement when the available space is low. Our replacement policy minimizes the overall expected query latency by selecting the chunks that are of larger size and less likely to be used again, and can be reproduced in shorter time. There are two ways to make space in Lifestreams DB: (1) compress the chunk, or (2) evict the chunk entirely. Compression on average results in 7.2x size reduction and can be restored more efficiently than reproducing a chunk from the source. Considering this difference, as well as the varying chunk sizes and cost in reproducing different kinds of chunks (see Table 4.2), we developed a Two-level Greedy-Dual-Size (Two-Level GDS) replacement policy that is both cost- and size-aware and appropriately chooses between the two space reduction methods. The basic Greedy-Dual (GD) algorithm assigns each chunk a cost value H. Each time a replacement needs to be made, the chunk with the lowest H value H_{min} will be replaced first, and all the other chunks reduce their H values by H_{min}. Only when a chunk is accessed again will its H value be restored to its initial value. Greedy-Dual-Size (GDS) incorporates differing chunk sizes by assigning H as $cost/size$ of the chunk [CI97].

On top of that, our Two-Level GDS algorithm additionally considers the different characteristics of compression and eviction. When a chunk is first inserted into the cache, its *cost* is set to the estimated decompression latency, and the *size* is the estimated space reduction after compression. When this chunk is selected for replacement, it will be compressed and re-inserted into the cache with its *cost* increased to the estimated latency to reproduce it from the source, and the *size* decreased to its size after compression. Only when this chunk is selected again will it be completely evicted. Similarly, after a chunk is reproduced, it will be first stored in its compressed form. When it is accessed again, it will have a certain probability to be promoted to its decompressed form. The default probability for a compressed chunk to be restored is 0.2. In this way, our algorithm uses compression as the default to make space, but still removes the compressed chunks to reduce cache clutter if they have not been used for long.

4.5.2.4 Chunk-Assisted RDF Query Evaluation

The flexibility of RDF is not without its drawbacks: compared to many SQL datastores, a RDF datastore tends to be slower in query evaluation due mainly to the difficulty of constructing an effective data index [MUA10]. Our chunk-based strategy has several desirable side benefits that mitigate this problem. First, chunk indexes can be utilized as a multi-column index that allows the query engine to take a short path by skipping those data that do not belong to the requested chunks. Second, chunking enables a more effective result cache, which caches the query results and returns the result when the same query is given. Unlike a record-based system, where any modification can potentially invalidate a cached result [MUA10], a chunk-based system only needs to track the modifications of the chunks that generate a cached result to ensure the result's validity. This technique is particularly effective in our system, as most chunks won't change after they have been generated.

4.6 Performance Evaluation

In this section, we evaluate the feasibility and performance of our system using Gmail and Moves data. Using Jena TDB as a baseline, we first evaluate the system performance in different scenarios and with different kinds of data. Then, we evaluate the overall system

Table 4.2: Gmail and Moves data-size and reproduction-time characteristics

Avg. Values of 180 Chunks	**Gmail**	**Moves**
Chunk Size (KB)	20.32	392.44
Compressed Chunk Size (KB)	3.08	54.12
Required HTTP Requests	14.24	1
Reproduction Time (msec)	1423.63	182.17

performance with a real-world query with a workload simulation based on an assumed application usage. The experiment was conducted on an Amazon Web Services (AWS) instance with 8 Intel Xeon E5-2680 processors and 15GB of memory.

4.6.1 Dataset

A dataset of 180 days worth of Gmail and Moves data is used to evaluate the system performance. The data are from three authors of this project who are regular users of these services. There are in total 360 chunks in the dataset, each of which contains a single day's Gmail or Moves data. Table 4.2 summarizes the different characteristic of Gmail and Moves data. For example, while smaller in size, a Gmail chunk requires many more HTTP requests to be issued and thus has longer reproduction time. A Moves chunk, on the other hand, can be reproduced in a much shorter time, but usually is much larger in size due to containing high-resolution location traces. These differences will result in different performance characteristics as shown in the following. These differences are taken into account by our caching policy to achieve efficient resource utilization.

4.6.2 Query Performance

We compare the query performance of our system with our baseline, Jena TDB, based on the following scenarios:

1. The demanded chunks are readily available.

2. The chunks need to be decompressed.
3. The chunks need to be decompressed and decrypted.
4. The chunks need to be reproduced from the data source.

The results suggest up to 14x performance improvement over Jena TDB for a both a simple query and a complex real-world query. The experiment was conducted with all 360 chunks pre-loaded into the triple store. Each data point presented below is an average of 30 runs of the experiment. The error bars in the figures are the 95% confidence interval.

4.6.2.1 Simple Query Performance

We first evaluate the performance with a simple query that counts the number of distinct Action subjects. Figure 4.8a and Figure 4.8b show the results for Gmail and Moves data respectively, where the x-axis is the number of chunks demanded in the query, and the y-axis is the mean query evaluation time. When the demanded chunks are cached in the system, our system outperforms Jena TDB by up to 14x and 10x for Gmail and Moves respectively. This performance gain is mainly attributed to the chunk-skipping optimization mentioned in Section 4.5.2.4. For Gmail data, decompressing shows up to 36x better performance than reproducing, and decryption adds only negligible overhead (less than 1.3%). This difference is not that significant for Moves, since Moves data can be reproduced in a relatively shorter time, but incurs larger overhead to be inserted into the triplestore in either scenario.

4.6.2.2 Real-World Query Performance

Next, we use a real-world query to demonstrate the system performance in a more realistic setting. A query from one of our small data applications, Ora, is used. It consists of 211 lines of SPARQL script, extracting 20 features from Gmail and Moves data (see Section 4.3.1 for a description). Since this more complex query requires a larger number of scans to be made over the search space, as shown in Figure 4.9, the performance gain of our chunk-skipping technique becomes more evident (up to 14x improvement over Jena TDB). In addition, due to the longer overall query time, the overhead in decompression and decryption becomes less significant. Reproducing is still the slowest among the four scenarios, but it still outperforms Jena TDB by up to 1.8x.

4.6.3 Performance with Simulated Workload

The varying performance for different types of data and scenarios stresses the need for a chunk replacement policy that is able to incorporate these discrepancies. We evaluate the effectiveness of the proposed Two-Level GDS algorithm using a simulated workload for Ora. Based on Ora's UI, we assume a binomial process usage pattern where each page shows one week worth of data and is browsed in a reverse chronological order. We assume the user will use the app daily, and after viewing a page, the user has a probability p to browse the next page or a probability of $1 - p$ to leave the app. We set $p = 0.7$ and compare our approach with the well-known Least-Recently-Used (LRU) policy, as well as a baseline Jena TDB instance that retains all the data. The results suggest that overall, our system outperforms LRU and Jena TDB by up to 4.7x using only a fraction of storage.

We generate 120 days worth of data for the workload based on an assumed usage pattern of Ora. We only consider the performance of the last 60 days when the cache space has become saturated. To allow a fair comparison, we modify the traditional LRU in a way that the chunk chosen for replacement will be first compressed and re-inserted into the LRU list. Only if it is chosen again will it be entirely evicted. We refer to this variant of LRU as *Two-Level LRU*. In addition, for the baseline Jena TDB instance, we assume it retains all the 120-day worth of data in the system, which is 50.44MB in size.

Figure 4.10 shows the performance of different approaches with cache sizes varying from 5MB to 20MB. Our Two-Level GDS shows superior performance over Two-Level LRU, especially with a smaller cache size. This advantage comes from the fact that our approach takes both the cost of different space reduction methods and the size of each individual chunk into account. For example, our approach tends to evict a Moves chunk for its shorter reproduction time and larger size. On top of that, if we use 0.5MB of the cache space to cache the query results, we see another 2x performance improvement. Overall, our approach achieves up to 4.7x performance improvement over Jena TDB, using only about 1/10th the storage.

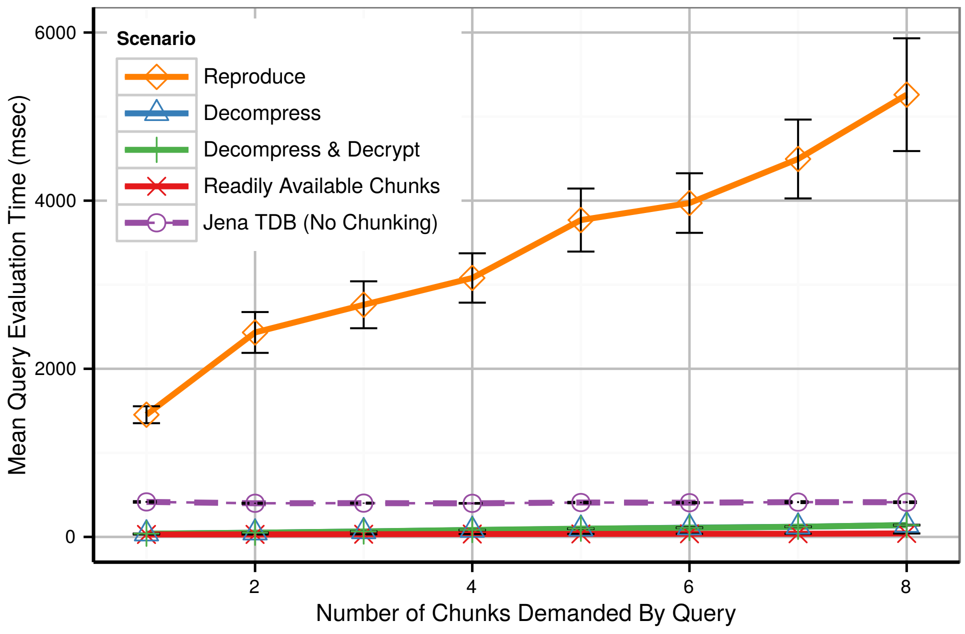
4.7 Conclusion and Future Work

In this work, we introduce the notion of small data apps, and the increasing opportunity of these apps to produce deeper and more comprehensive insights across the union of a user's available data, and across a wide range of ubiquitous computing applications. By virtue of the fact that these apps leverage the user as the common denominator and benefactor, there is both the potential for deeper, more personal insights, as well as the need for a robust infrastructure for accessing such intimate data. We present an architecture to support these small data apps that decouples the data sources from the processing and visualization layers, and accounts for the unique challenges presented in contending with sensitive streaming spatio-temporal data from multiple providers. We describe our implementation of a critical component of this architecture, Lifestreams DB, and several candidate applications built on top of it.

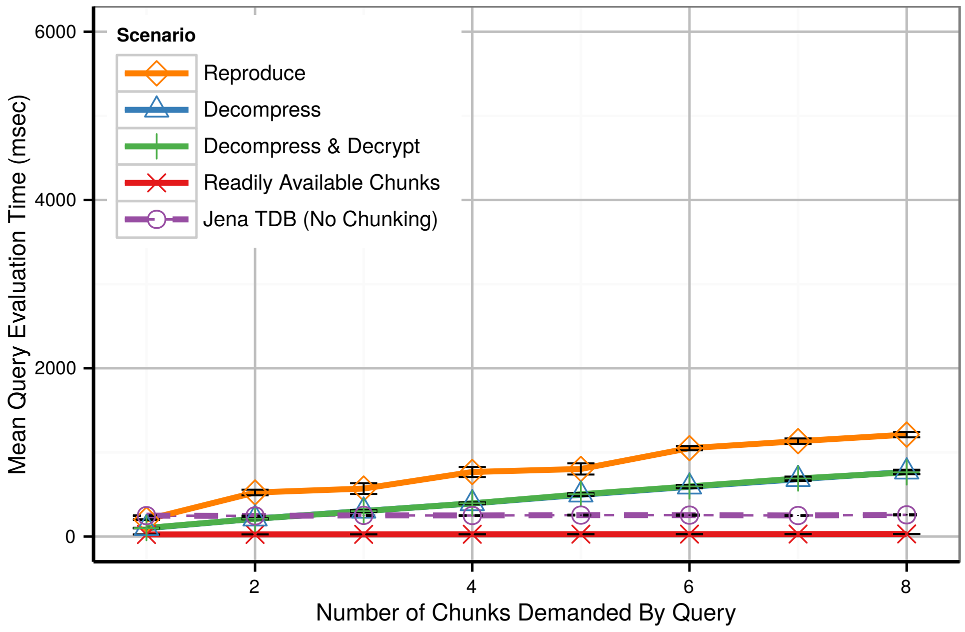
Lifestreams DB includes several improvements over existing RDF datastores in terms of storage requirements and query latency, which are likely attributable to the constraints of our domain (i.e., streaming spatio-temporal data which can be reproduced at a cost in latency from an external source.) The application of chunking to the datastore, and a cache eviction policy that leverages both the cost of reproduction/compression and the size of the data, is demonstrated to improve query latency for both a few candidate queries and in a simulated experiment modeling a user's long-term interaction with Ora, a small data application.

While this work proposes a soft-state architecture to ameliorate the impact of a breach, there is still much work to be done in **secure data storage and distribution** so that breaches are diminished or, preferably, eliminated in the first place. On a related note, there are many improvements that can be made to ensure that the processed data does not compromise the raw data source, and to selectively control who can consume processed data in the case that it is sensitive.

Small data apps address the converse of the big data problem: rather than drawing insights about populations across broad swaths of data for purposes of similar scale (e.g., corporate, governmental, etc.), they draw insights about the individual across their own small data for personal growth and understanding. This work aspires to **foster the growth of the small data ecosystem and the role of small data in fueling ubiquitous computing applications**.



(a) Gmail Data



(b) Moves Data

Figure 4.8: Query Performance of Different Scenarios: Gmail reproduction latency dominates due to the larger number of HTTP requests. Decryption/decompresion add neglible overhead.

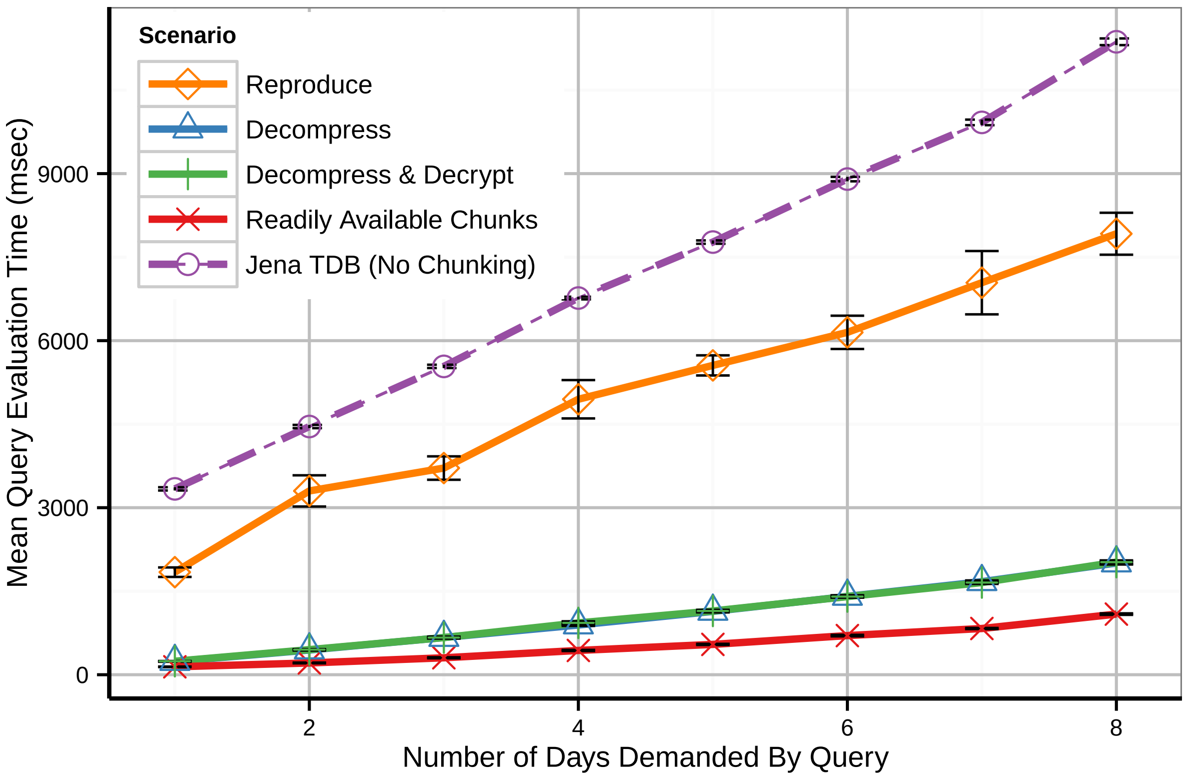


Figure 4.9: Real-World Query Performance: the performance gain of our chunk-skipping technique becomes more evident (up to 14x) for a complex real-world query where more scans need to be made over the search space.

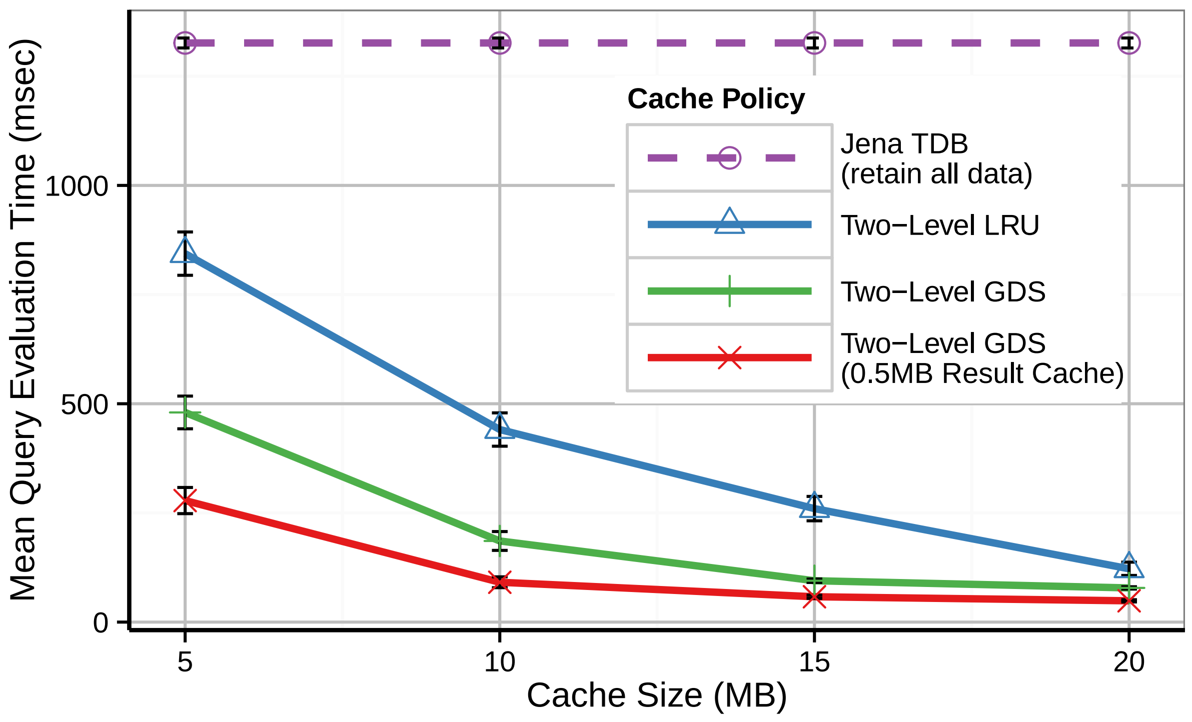


Figure 4.10: Query Performance with Simulated Workload: our Two-Level GDS approach shows superior performance over LRU, and outperforms Jena TDB that retains all 50.44MB of data, by up to 4.7x using only about 1/10th the storage.

Part II

Selective and Informed Sharing

The contributions of previous parts have focused on the user as the principal consumer of processed data. While small data applications should ultimately serve individual users, there is great potential for small data to be used in research studies as well, in which case the principal consumer is a researcher. Nevertheless, the user still acts as the mediator to gain access to their data, and as a valuable source of interactive context about what data is relevant to the study and what they feel comfortable releasing. In order to provide this context, the user must be aware of what information their data actually contains, and must be given the tools to curate their data before releasing it to researchers.

This part introduces a tool to realize the goal of selective and informed sharing. Specifically, the first chapter describes the Takeout Processor, a system built on top of Google's Takeout service. It consists of a backend that parses, normalizes, and efficiently indexes the contents of the Takeout archive, and a user-facing front-end that walks the participant through the process of acquiring, exploring, and filtering their archive within the context of a research study, eventually producing a curated archive for use in the study. The second chapter assesses whether the interface is comprehensible to a general audience through a user study conducted on Amazon Mechanical Turk users. While the interface does not yet fully realize the goal of selective sharing, the results of the user study provide both concrete recommendations on how it can be improved, and a framework for iteratively testing and refining the interface through future studies.

CHAPTER 5

Takeout Processor (TOP) Architecture

Submitted to ICDE 2018 as "Takeout Processor: a Multi-Modal Data Acquisition and Filtering Pipeline" Alquaddoomi, F., Wen, H., & Estrin, D. Included here with typographical modifications and redundant sections removed.

5.1 Introduction

Research studies involving user data can be divided into two types: *prospective studies*, in which users are first recruited and then begin to collect study data, and *retrospective studies* in which user data has been collected prior to the study's initiation. Prospective studies tend to place a burden on the user, either in terms of instrumenting themselves for data collection (e.g. through installing mobile apps or wearing sensors) or in regularly submitting data to the study. One benefit is that data collection can be tailored specifically to the study, but there is also a challenge in developing and maintaining these instrumentation tools. There is also a necessary waiting period before analysis can begin, increasing the study's cost and risk. In contrast, retrospective studies do not explicitly burden the user and analysis can begin as soon as the data is made available to the researchers. Retrospective data is preferable to prospective data in terms of reducing the logistical overhead and time to complete a study, but it is difficult to procure these datasets in a way that both preserves user privacy and is sufficiently informative for the purposes of one's study.

There has recently been a movement among large service providers to "federate" user data (that is, to allow it to be exported from the system in bulk), motivated both by the desire to provide a better service to their consumers and through pressure from consumer advocacy and political entities. For instance, in 2014 the European Parlaiment passed privacy legislature which, in part, enforces a requirement for personal data to be portable between services [Fia14]. Organizations within service providers such as Google's Data Liberation Front [Fit11] have also worked toward making personal data portability a reality. As a result Google created Takeout[1], a website that allows users to export their data across a large number of Google services to a single downloadable archive. Facebook has followed suit with a feature to download your data as an archive[2], delivered as a link in an email.

These federation services provide a unique opportunity for researchers to obtain datasets that are both retrospective and "complete" in the sense that they have not been anonymized or aggregated. Google's Takeout archive is an especially interesting subject granted Google's deep integration into Android and significant presence on iOS devices and the web. The number of potential data modes in the archive – location history, search, email, browsing history, etc. – is unprecedented in a single source. In terms of population coverage, Google's services are demonstrably popular: according to Net Market Share, Google holds a majority market share in both search [marb, mard] and browsers [mara, marc] (i.e. Chrome) on both desktop computers and mobile devices. While Google does not hold a majority in mobile devices, Google's Location History service is, as of April 2017, available for iOS devices and is also included in the archive [tak].

There are many peripheral advantages for researchers in making use of a third-party data collection organization. As mentioned, it removes the burden of instrumentation from the researchers and participants. People already use Google services, whereas installing apps, and moreover having research labs develop and maintain said apps across multiple platforms, is an often untenable cost. It removes the necessity for research labs to store or otherwise be responsible for the data for the majority of the time; researchers need only gain access to it during analysis. The fact that this data belongs to the user and they have the means to release it to researchers answers an important question of purview over the data (that is, it belongs solely to the user to release at their discretion).

[1] `https://takeout.google.com/settings/takeout`

[2] `https://www.facebook.com/help/131112897028467`

There are also a number of challenges presented by these federation services. For security reasons the services – justifiably – require user intervention to obtain the archive, which impedes onboarding study participants. The data is acutely sensitive and thus incurs a risk of loss of confidentiality to the participant. Further, the data arrives as a per-user archive and can be difficult to parse and analyze, especially when performing studies across users. The remainder of this work addresses these challenges, generally by automating as much of the process as possible and guiding the user where necessary, by providing fine-grained filtering options to the user to mitigate unnecessary disclosures, and by providing an efficient pipeline and standardized query interface to ease analysis.

While this work describes our architecture and demonstrates its utility with an example experiment over just location data, our original motivation for this work was to access retrospective data for various health related studies. Future work will explore how retrospective data learning using this Takeout pipeline can be used to develop novel classifiers for patient-symptom and treatment-outcome measures, and how retrospective data can be used as a prior for prospective interventions and studies.

5.2 Prior and Related Work

5.2.1 Takeout in Research and Commercial Products

There are a number of commercial products and at least one research study that currently make use of the Takeout archive. In a study on walksheds (the walkable area around an individual's work and home), researchers employed the location trace extracted from Takeout archives as a form of "Voluntary Citizen Geospatial Data" [LG]. Their collection process was entirely manual, requiring users to contact the study coordinator for an appointment. A4Cloud, a policy institute focused on privacy and security issues around cloud services, has discussed the potential of using Takeout archives as input for user-facing tools that assess disclosure of personal information [ABE15]. In terms of commercial uses, a bevy of products for importing email into other clients and services are able to parse GMail messages in the

Takeout archive. "Location History Visualizer" [loc16] is a commercial product that parses the location history data in the archive and provides tools for visualizing and annotating the data.

5.2.2 Alternative Data Collection Methods

In-depth discussion of alternative methods for prospective collection is beyond the scope of this work, but we mention a few here for completeness' sake. The Moves[3] app collects and visualizes a similar activity-annotated location trace to Google's location history. RescueTime[4], a cross-platform time-management app, includes an option for logging web history (including search history), akin to Takeout's search and browser history logs.

5.3 Architecture

The pipeline consists of three discrete units: a web-based user interface, a distributed task queue for processing asynchronous tasks, and a database for storing the filtered results. The web interface is further divided into a client-side application and a private API that the client application queries. In the following sections we describe each component in rough order from where the end-user interacts with the system to where the acquired data is eventually stored.

The pipeline consists of the following broad phases:

- **Archive Generation:** The user visits Google Takeout and requests an archive, specifying that it should be copied to Google Drive.
- **Authorization:** The user provides authorization for Takeout Processor to access their Google Drive account, then selects the generated archive for processing.
- **Acquisition, Extraction:** (Background task) The archive is copied from Google

[3]`https://moves-app.com/`

[4]`https://www.rescuetime.com`

Drive to a Google-hosted server, where it is decompressed and extracted into a folder hierarchy.

- **Import:** (Background task) A task is launched for each data mode (currently search, browser history, and location) that prepares the relevant files for insertion into the database and then inserts them. Indexing is performed as data is inserted.

Additionally, there are two optional phases that can be conducted in any order:

- **Filtering:** The user interactively specifies filtering templates, which are applied to their data prior to its release to researchers and to the analysis tasks.
- **Analysis:** The user can launch visualization tasks via the interface and view the results inline. Researchers can query a user's filtered data via a secure connection to the PostgreSQL database and perform further analysis.

5.3.1 Client-Side App

The user-facing part of the app is primarily developed in React, a popular Javascript framework. In order to make the site feel more responsive each page is implemented as a root React component, which can be bound to changing data on the remote server and seamlessly update the page without requiring a refresh. React's component-based model makes separating concerns and testing portions of the UI much easier than non-component-based frameworks such as Ember.js and Angular. The app also makes use of a number of third-party React-compatible components, specifically **react-bootstrap** for themeing, **react-datetime-picker** for date and time pickers, **react-googlemaps** as a map viewer/editor for creating location filters, and **react-target-time**'s day-of-week/hour picker for time filtering.

Due to the fact that OAuth2 token-based authentication requires a full server round trip, the OAuth2 authorization flow includes at least two full page refreshes, one for redirecting to Google for authentication, and one back to the Takeout Processor for receiving the token and displaying the results. Otherwise, the site is implemented as a single-page Javascript application. We opted for this flow over a fully client-side OAuth2 flow because we require the server to fetch data on behalf of the user.

5.3.2 HTTPS REST API

Supporting the client-side app is a server-side REST/RPC API, implemented in Flask (a “micro" web framework for Python). The API handles authentication of incoming requests and communicating with both the database and the distributed task queue. It RESTfully serves entities from the database encoded in JSON. While the task queue management endpoints are not strictly RESTful (since they do not reflect entities, but processes), requests to download or process an archive or run an analysis task are exposed through the same API as the database entity requests.

5.3.3 Distributed Task Queue

The distributed task queue asynchronously handles tasks that cannot be completed within a single request-response cycle in Flask. These include downloading a Takeout archive from Google Drive and performing the pre-processing, filtering, and analysis tasks that eventually result in the Takeout archive's contents being entered into the database. All task progress is incrementally written into the database for visualization on the front-end.

The task queue is implemented in Celery (a popular Python task queue that is often used with Flask), and uses Redis (an in-memory database) as a message broker for synchronizing the queues. At the moment we run a pool of three workers on the same machine on which the rest of the pipeline runs, but it could be scaled up to a larger number of worker nodes conceivably running on separate server instances.

5.3.4 Database

As mentioned in the previous section, the contents of the user's Takeout archive are inserted into a database to ease analysis of the data. We chose PostgreSQL for its maturity, performance, and extensive library of supported data types. We specifically use the PostGIS extension to store location traces and to perform geographic queries on those traces (for filtering out regions that the user wishes to remove from study analysis, for instance.)

The database currently stores metadata for each archive, and has one table keyed to the archive for each data modality that we support. We currently support search/browser history and location, with plans to expand to other modes as necessary. Additionally, the database holds filtering metadata on a per-archive basis, allowing the user fine-grained control over what parts of their data are included in their visualizations. This includes location- and search-term-based filtering, with the interface for those functions described in the filtering section below, 5.4.2. The filters are currently applied via a view over the raw data tables, and analysis modules can access only these filtered views.

5.3.5 Visualization Generation

In addition to the copying, extraction, and database importing tasks mentioned above, the pipeline provides additional tasks that access specific filtered data modes and produce visualizations inline in the user interface. These visualization tasks are run at will by the user, since they should occur after the user has had an opportunity to specify filters for their data. The available visualizations are described in the “Visualization" section5.4.3.

5.4 Usage

The following sections describe the end-user experience, specifically in acquiring, filtering, and viewing visualizations of their data.

5.4.1 Archive Acquisition

The user’s interaction with Takeout Processor begins from the Takeout Processor website[5]. Before interacting with the site, the user must consent to an agreement that details how and for what purpose their data will be used. Once they have agreed, the user is presented with

[5] `https://top.smalldata.io`

a series of step-by-step directions for creating their archive and importing it into Takeout Processor.

Google's Takeout service requires that the user interactively requests their Takeout archive – it is currently not possible to have the archive automatically generated. Granted that fact, part of the usage flow involves the Takeout website, where the user selects the data modes that they would like to be included in their archive, as well as where the archive will be eventually stored. Takeout storage options include Google Drive, Dropbox, and downloading the archive to the user's local machine. Our system currently supports acquiring from Google Drive, but we intend to implement acquiring from other sources (especially user upload) in the future.

Once the user has exported their archive to Drive, they return to the Takeout Processor interface. They must then authorize Takeout Processor to access Drive, and then specify the location of their archive; this initiates a series of background tasks which the user can elect to monitor in the interface if they choose. The user is free to close the browser at this point, and only needs to return if they wish to perform filtering or analysis, both of which are optional steps.

5.4.2 Filtering and Review

Due to the personal and potentially sensitive nature of the data contained in the user's archive, we provide fine-grained filtering options to allow users to redact information that they may be uncomfortable sharing with researchers or simply wish to exclude from their own reflection analyses. Supporting this filtering process, we also include mode-specific components for reviewing the unfiltered data; this allows users to make informed choices about what to filter, and to review the results of their filtering. The filtering interface is designed to be easily extensible; we anticipate creating more filtering and review options as we include more data modes from the Takeout archive.

Filtering and review components are described in detail in Chapter 6, Subsections 6.5.2 for filtering components and 6.5.3 for review components.

5.4.3 Visualization

In addition to supporting research studies (which we anticipate will be conducted against the filtered database directly with specialized tools), we also support analyses by end-users via the interface. We currently support the following visualization modules:

- **Places Visited:** a line graph of the number of distinct places visited over time.
- **Enter/Leave a place:** a line graph of the time entering and leaving a pre-defined location (e.g. workplace) over time.
- **Search Term Count:** a line graph of the occurrence of a user-provided search term over time.

Visualizations are executed by the user with an optional set of parameters; their progress can be monitored by the user once the task has been started through the interface. We currently use plot.ly, a plotting library with broad language support, to produce the visualizations. For the supported visualization modules, the implementation details are as follows:

Places Visited is derived by performing DBSCAN (Density-Based Spatial Clustering of Applications with Noise) per day on the user's location trace. It recognizes high density samples and expands clusters from them, which in our case differentiates distinct places a user visited (such as home, workplace, restaurants) for that day from other noise samples collected, e.g., during transportation. The clustering algorithm requires two parameters: *eps* and *min_samples*. *eps* defines the maximum distance between two samples in the same neighborhood. *min_samples* defines the minimum number of neighbors a point needs to have to be included in a cluster. Since we use the Haversine formula to calculate the distance in kilometers between two locations as the distance metric for clustering, we choose *eps*=0.2 so that the maximum neighborhood radius is 200 meters. We set *min_samples* = 10.

Enter/Leave a Place is detected by looking at the location transitions of the user. Given a place, if we identify that the user arrives at the place and stays for a certain period of time, we define it as a ‘enter’ event. Similarly, we define a ‘leave’ event as the user moving from and remaining away from the place for a length of time. If the user is absent from the place, no enter or leave events are recorded.

Search Term Count is calculated by counting the occurrence of the user-provided search term in the search query database on a daily basis. The presence of any word from the user search term in the query is considered a match, and added to the count.

Much like the filtering interface, the visualization interface is intended to be extensible; modules can be added by implementing a Celery task. Tasks have access only to the filtered data for the selected archive, but can integrate across any of the available modes. The task must eventually produce an HTML file, which is displayed in an iframe. The HTML file can reference additional local artifacts, such as images, and can include Javascript code to allow for interactive visualizations.

5.5 Security and Privacy

As mentioned in section 5.4.2, controlled release of user data is a core part of the system.

To reduce the vulnerability footprint of the system, all server-side parts of the architecture run on Google’s cloud infrastructure. The user-facing portions are secured via HTTPS.

5.6 Illustrative Use Case and Future Work

5.6.1 Preliminary Experiment

To illustrate the use of our system, we conducted a small exploratory study in conjunction with our lab’s relocation from downtown Manhattan to Roosevelt Island. We recruited six users from Cornell Tech who were sited on both campuses, before and after the move; each user granted access to their location data through Takeout Processor. We used the natural experiment of the move to validate if changes in daily patterns can be detected by processing of Takeout data alone: in this case, if we could identify the campus move solely from summarizing statistics derived from their location trace. In this section, we will explain what we found in the data and illustrate metrics that we can use in future studies to characterize users' behavior change over time.

We explored several daily metrics to measure the shift in location trace: places visited, maximum intra-day distance, and time spent in the workplace.

- **num_place** is the number of places visited on a specific day. We use the implementation introduced in Section 5.4.3. The minimal value is 1, which indicates staying in the same place for the entire day.
- **max_dist** is the diameter in kilometers of the smallest circle that would cover all the location traces of the day, i.e. the distance between the furthest two points the user has visited in a day. We first compute the convex hull of all location points and then find the maximum distance between points on the hull.
- **time_stayed** is the time in minutes the user spent at their workplace. It is calculated as the difference between the first arrival to and the last departure from the workplace in a day.

We considered one month worth of data before and after the move, respectively, for each user. We excluded those days when the user is on travel by applying a threshold of *max_dist* >30km. This yielded 353 days of data across all users.

As a first exploration of whether the pipeline delivers credible user location patterns, we examined weekdays and weekends as separate conditions, expecting to see consistent differences in location patterns. Fig. 5.5 illustrates the distribution of *num_place* and *time_stayed* in the two conditions. As expected, we see that users are more likely to stay at the same place and less likely to go to the workplace on weekends compared to weekdays. We average each measure for each user in two conditions separately. A paired t-test shows significant difference in *time_stayed* for weekday (M=400.2, SD=119.57) and weekend conditions (M=138.9, SD=148.82) with $t(5)$=4.92, p=0.004. No significant effect was found on *num_ place* (Weekday: M=2.9, SD=0.78; Weekend: M=2.7, SD=0.40) and *max_ dist* (Weekday:M=5.8, SD=1.60; Weekend:M=5.0, SD=1.38).

We then examined the effect of the move on these measures by comparing the data before and after the move. We filter the days when the user stayed in the workplace for less than 5 hours. After enforcing this criterion, one user contributed less than 3 data points before the move, so we excluded that user for this part of the analysis. Overall, we considered 164 days of data across 5 users. Using a paired t-test, we found significant difference in *time_ stayed* on campus before (M=495.2, SD=67.10) and after the move (M=569.1, SD=85.41) with $t(4)$=-3.70, p=0.02. By looking into the enter and leave time respectively, we found a significant difference in enter time ($t(4)$=3.44, p=0.03) but not in leave time. This indicates the user entered the workplace earlier after the move and left at a similar time as before, resulting in a longer time stayed in the workplace. A less significant difference is found in *num_ place* (Before: M=3.0, SD=0.36; After: M=2.2, SD=0.62; $t(4)$=2.55, p=0.06). Users seems to visit fewer places after the move.

Through this simple use case, we validate the functionality of the pipeline and the ability to extract relevant metrics about daily patterns from Google Takeout. Going beyond the study, we anticipate leveraging these metrics as building blocks to model user behavior patterns over time through more purpose-driven and likely health-related studies enabled by our system.

5.7 Conclusion

In this work, we described the creation of an extensible processing pipeline, user interface for filtered data release, and query interface for research studies, built on top of Google Takeout. We demonstrate its use in an example study on inferring changes (i.e. the effect of our campus move on our residents' movement patterns) from the data made available through the service.

In the future, we anticipate extending this pipeline to meet the needs of health-related studies that use retrospective data collected from Google and possibly other services. We intend to explore, among other topics, how retrospective data learning can be used to develop novel classifiers for patient-symptom and treatment-outcome measures, and how retrospective data can be used as a prior for prospective interventions and studies.

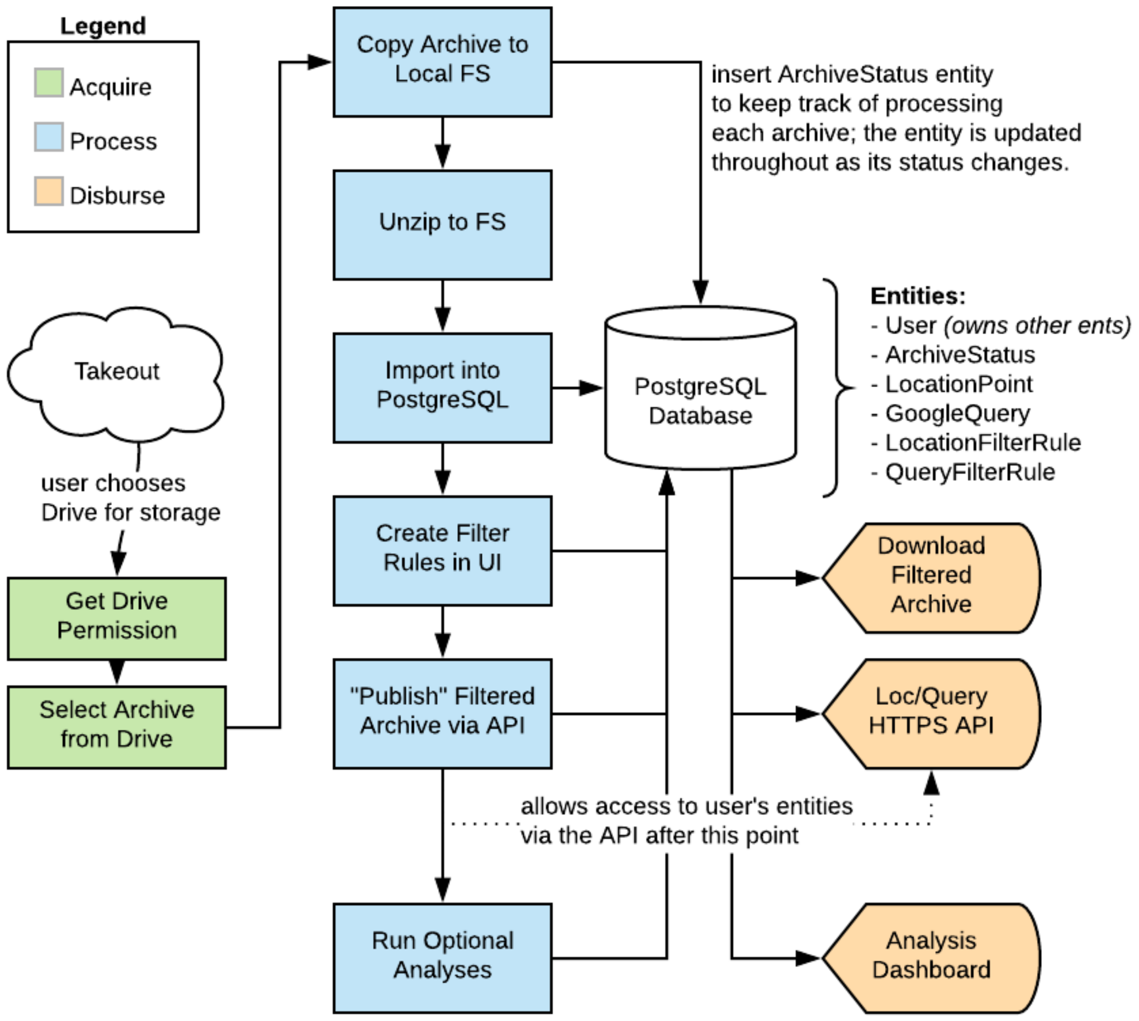
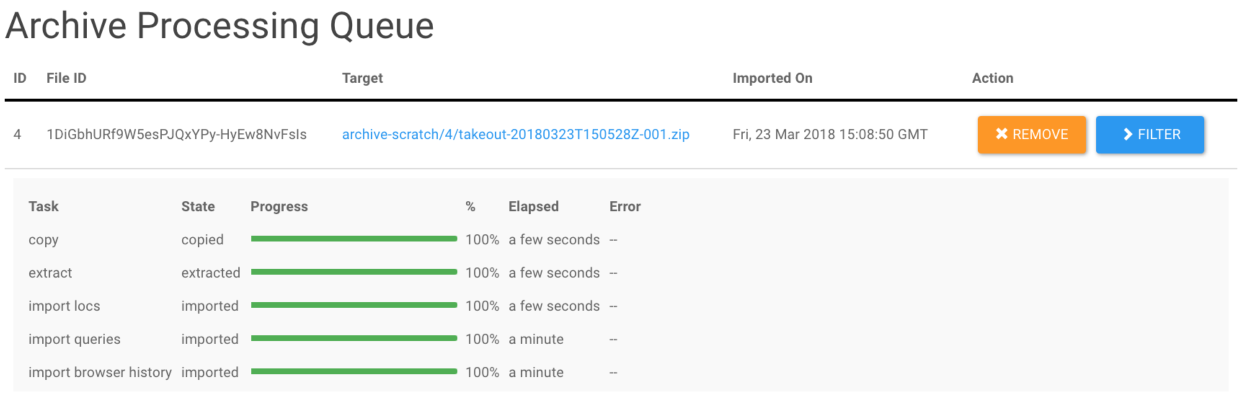

Legend
Acquire
Process
Disburse
Takeout
user chooses Drive for storage
Get Drive Permission
Select Archive from Drive
Copy Archive to Local FS
Unzip to FS
Import into PostgreSQL
Create Filter Rules in UI
"Publish" Filtered Archive via API
Run Optional Analyses
insert ArchiveStatus entity to keep track of processing each archive; the entity is updated throughout as its status changes.
PostgreSQL Database
Entities:
- User (owns other ents)
- ArchiveStatus
- LocationPoint
- GoogleQuery
- LocationFilterRule
- QueryFilterRule
Download Filtered Archive
Loc/Query HTTPS API
allows access to user's entities via the API after this point
Analysis Dashboard

Figure 5.1: Combined block and sequence diagram showing the components involved in processing and releasing archive data


Archive Processing Queue

ID	File ID	Target	Imported On	Action
4	1DiGbhURf9W5esPJQxYPy-HyEw8NvFsIs	archive-scratch/4/takeout-20180323T150528Z-001.zip	Fri, 23 Mar 2018 15:08:50 GMT	REMOVE FILTER

Task	State	Progress	%	Elapsed	Error
copy	copied		100%	a few seconds	--
extract	extracted		100%	a few seconds	--
import locs	imported		100%	a few seconds	--
import queries	imported		100%	a minute	--
import browser history	imported		100%	a minute	--

Figure 5.2: Archive acquisition status display, showing the status of each subtask associated with importing a Takeout archive.

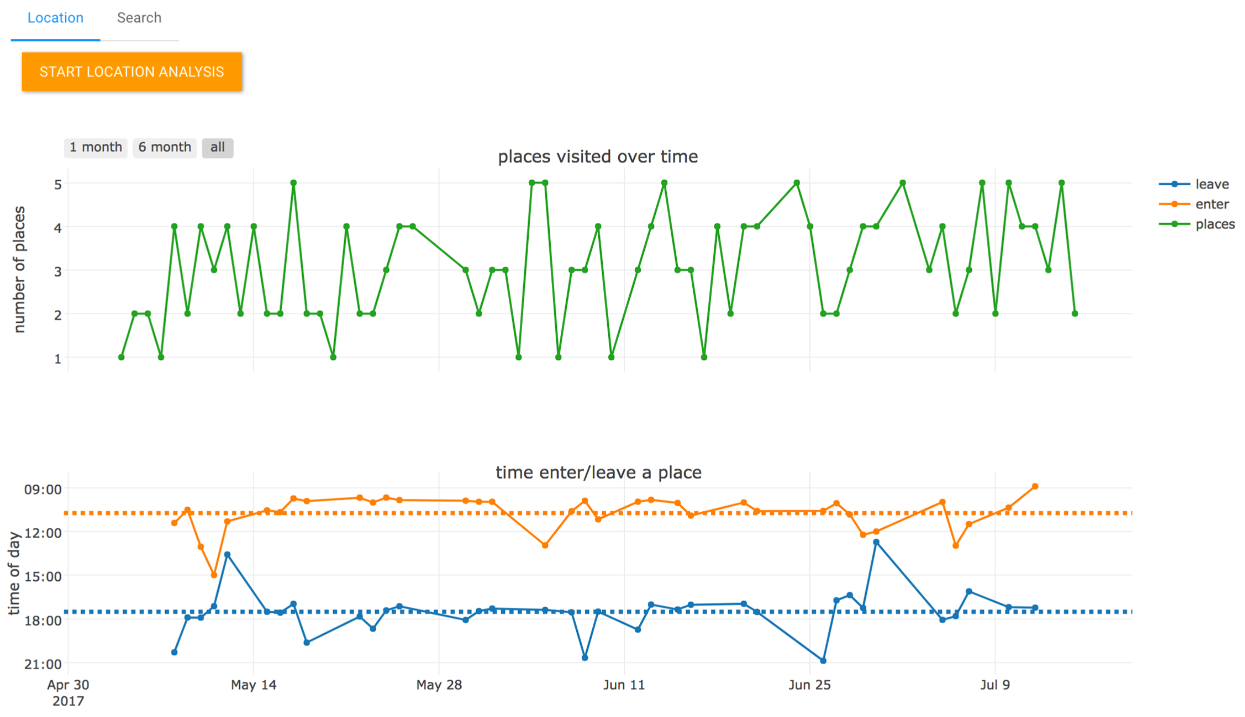


Figure 5.3: Location analysis, showing times entered and left Cornell Tech's campus and the number of distinct places visited, both over time.

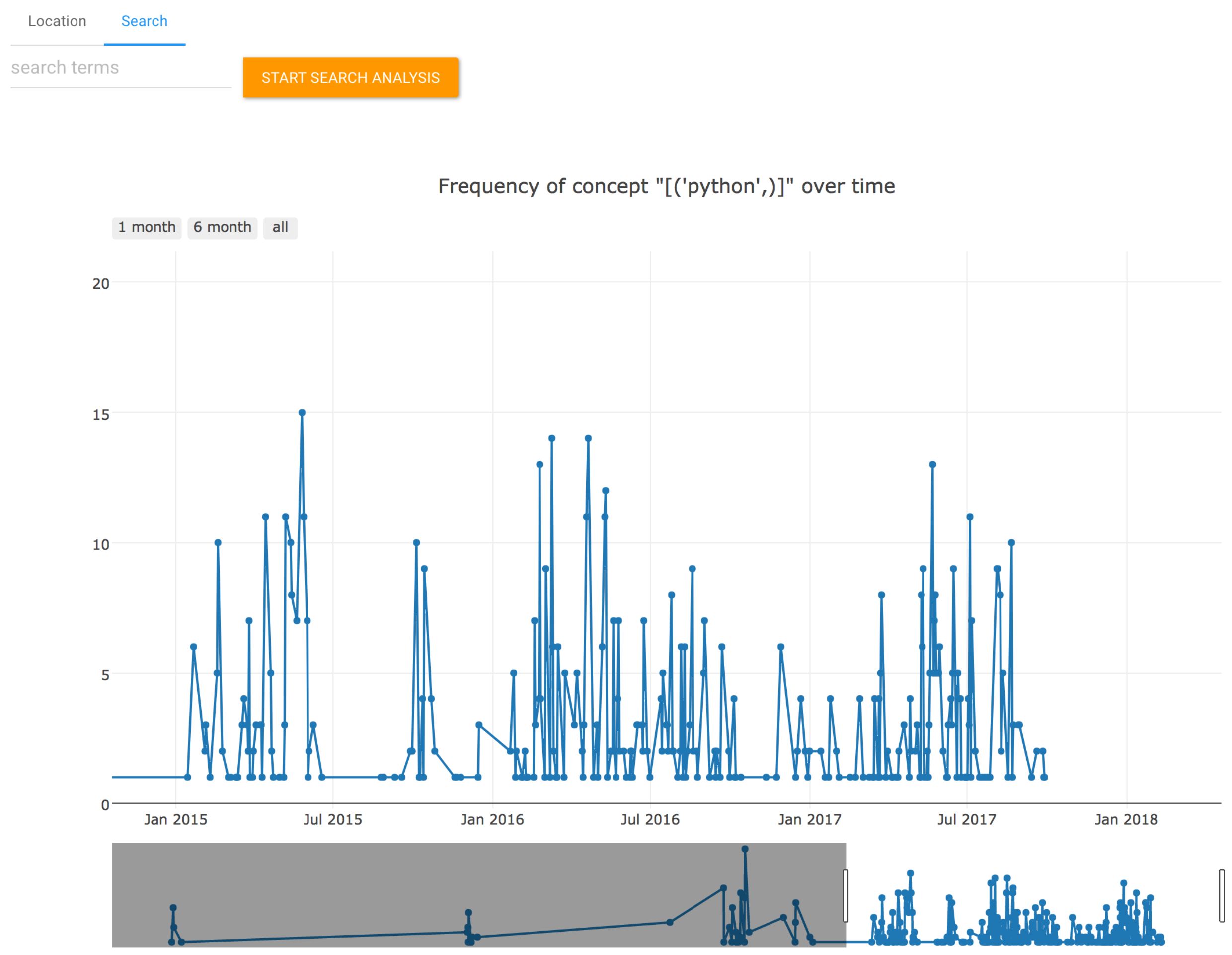
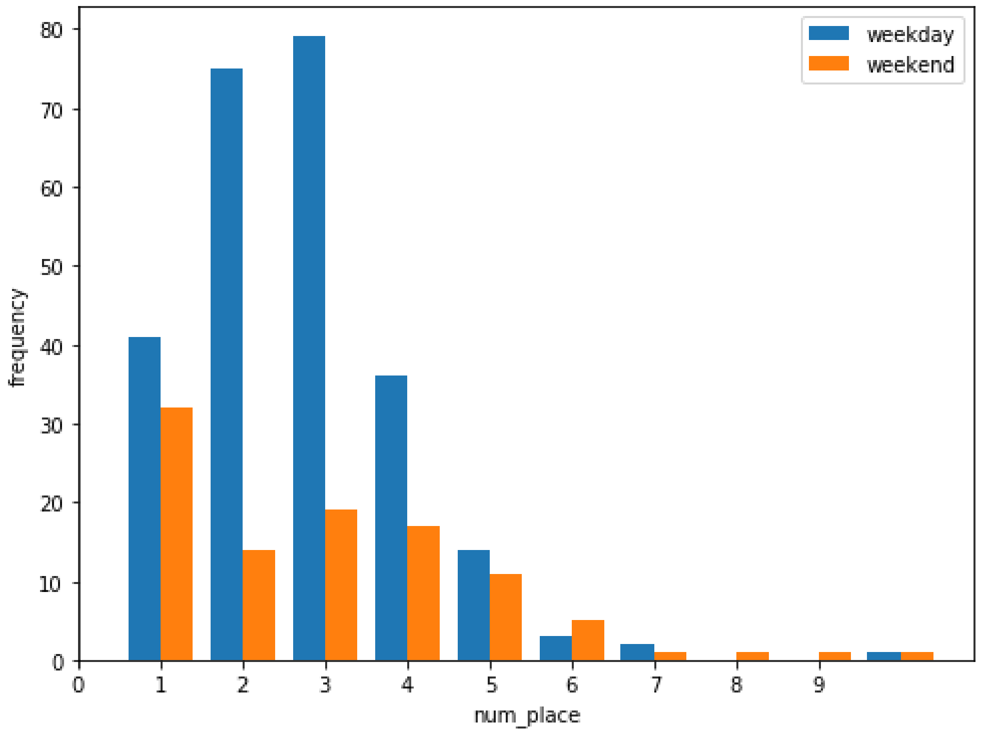


Figure 5.4: Search analysis, showing occurrences of searches for the term “python" over time.



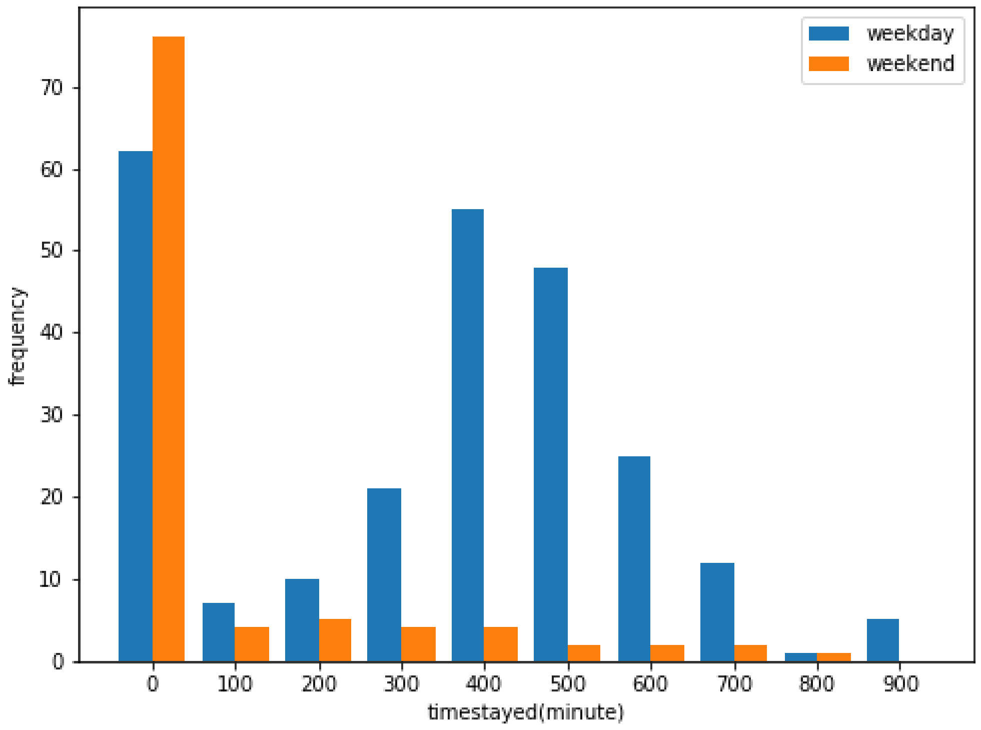


Figure 5.5: *num_ place* and *time_ stayed* across users on weekdays and weekends. While the difference in time stayed at work is unsurprising, the difference in the number of places visited deserves further investigation.

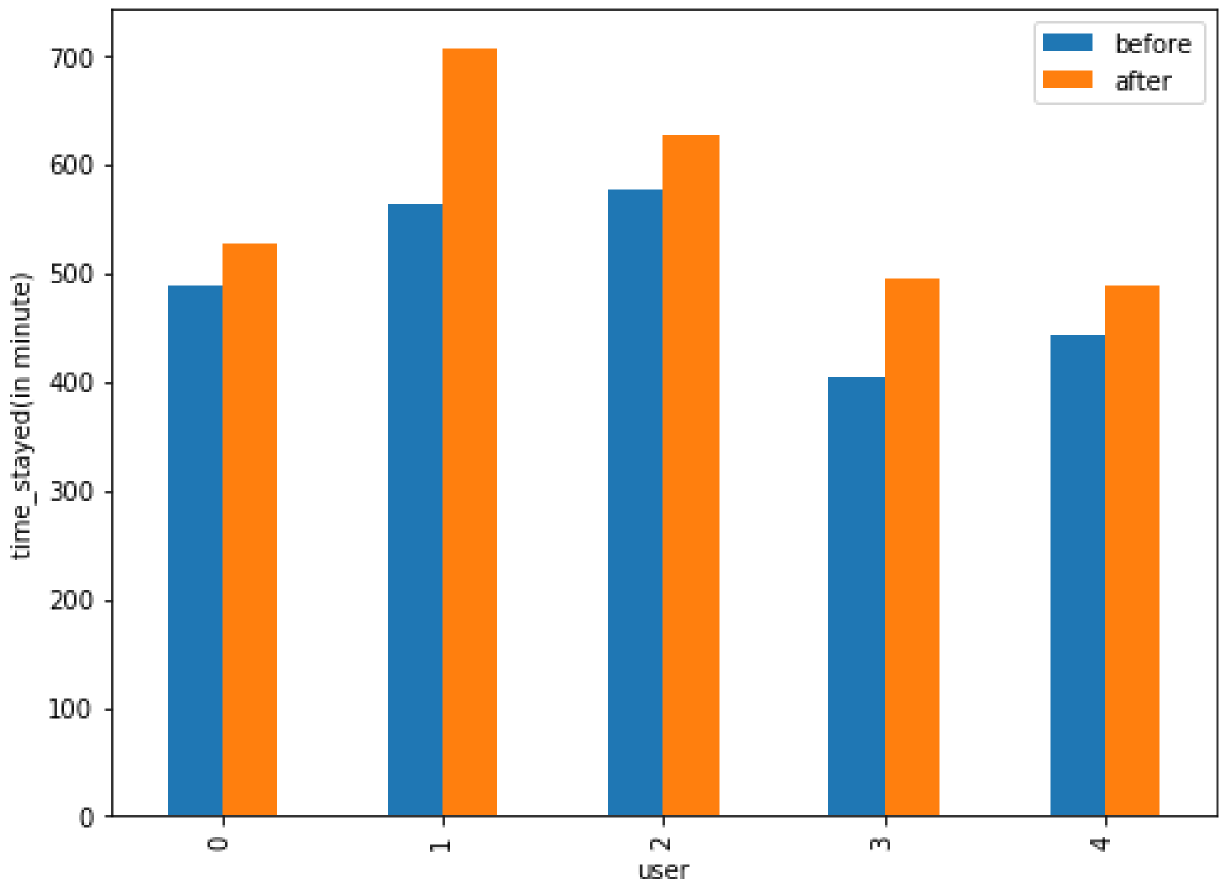


Figure 5.6: Individual level *time_ stayed* before and after the move. After the move users spent more time at campus, perhaps due to it being further from their residence than before.

CHAPTER 6

TOP Validation: General User Study

Submitted to NordiCHI 2018 as "Evaluating the Feasibility of a Personal Data Filtering Interface" Alquaddoomi, F., Tseng, E., & Estrin, D. Included here with typographical modifications.

6.1 Introduction

Individuals routinely and increasingly interact with online services through a variety of devices, producing personal data that provides a valuable – but acutely sensitive – window into their lives. The creation of data federation mechanisms over the last few years, such as Google Takeout [Fit11] and Facebook's export data feature [Inc18], enables a different kind of research methodology that we refer to as *retrospective data learning*, learning on personal data that has already been collected. This is in contrast to *prospective* methodologies that first recruit users, then either instrument their devices or ask that they regularly submit data to the study. Big Data, while potentially retrospective, lacks this individual focus, and thus the opportunity to put the user "in the loop" of their data's usage and interpretation. We anticipate that the availability of such federated personal data will only increase, as data federation mechanisms are now mandated by newly applied legislation in the European Union, the General Data Protection Regulation (GDPR) [Eur16], specifically Article 20, "Right to data portability".

With this opportunity in mind as well as its inherent privacy complications, we have developed a web-based filtering interface for Google Takeout archives, named Takeout Processor ("TOP" for short), which is intended for Internal Review Board-approved studies in which participants submit filtered versions of their personal data with informed consent. The system guides research participants through the process of exporting their study-relevant data from Google via Takeout, importing it into TOP, and then constructing fine-grained filters to exclude data from consideration in the study. The researchers access the filtered data through a PostgreSQL-based interface, which alleviates the burden of parsing and normalizing Takeout data.

Prior to this work, we have performed a few technical and limited pilot studies with the system to evaluate its application, which have all involved direct and sustained in-person coaching on how to use the interface. This work presents the results of an experiment on evaluating the feasibility of the interface with a more general audience, specifically Amazon Mechanical Turk (MTurk) users. The participants were given brief guidelines and a set of tasks to perform on a sample dataset, with the intent of evaluating how well they can perform these tasks *without* intensive guidance. While the results were mixed, they do suggest new directions in which to improve the guidelines and interface.

The remainder of the chapter is structured as follows. Section 6.2 briefly covers related work using retrospective data sources. We describe our study methodology in 6.3 and present the study results in Section 6.4. Section 6.5 provides an in-depth description of the functionality presented in the interface for both filtering and reviewing user's Takeout data. We close the chapter by discussing the results in Section 6.6 and with some final remarks in Section 6.7.

6.2 Related Work

While our system is, to our knowledge, the first offering to the research community of its kind, the concept of employing federated data, and specifically Takeout data, has existed for some time. A4Cloud, a policy institute focused on privacy and security issues around cloud services, observed that Takeout archives could be used as input for user-facing tools that assess disclosure of personal information [ABE15]. The issue of data portability, and indirectly service providers that support it, has been examined from a legal perspective as well, evoking both praise for stimulating market competition [GVV13] and concern for consumer welfare in the case of data breaches [SL12]. Some are concerned that providers like Google and Facebook do not disclose enough of the data that they collect and that Takeout constitutes a token effort, albeit one in the right direction [VK15].

In terms of research projects that actually use Takeout, a project investigating walksheds (the walkable area around an individual's work and home) employed the location trace extracted from Takeout archives as a form of, in their words, "Voluntary Citizen Geospatial Data" [LG16]. The data were collected manually in a sit-down session with the participants.

A few commercial offerings exist that ingest Takeout archives; for example, "Location History Visualizer Pro" provides a service by which users can obtain in-depth analysis of their Takeout-derived location trace for a fee [loc16]. Individuals also make use of Takeout to migrate their Google account data between different accounts or systems, evidenced by tutorials on a number of blogs [Cha18] and institutional websites [Uni18, She18].

6.3 Method

Each participant is given a sample dataset and instructed to pretend that it is either their own data or the data of a family member, depending on the study arm into which they have been assigned. Each participant must perform a set of filtering tasks on the dataset using the filtering interface.

Our evaluation is divided into three parts: a quantitative evaluation in which we compare the participants' filtered datasets against a reference filtered dataset, an investigation into the filters the users created to address the tasks, and a qualitative survey where we collect their impressions of the system and the tasks.

6.3.1 Recruitment and On-boarding

We recruited 20 U.S.-based English-speaking individuals from Amazon's Mechanical Turk ("MTurk") service, with the additional constraint that they are between the ages of 25 and

55. Each participant was compensated $15 USD for their involvement, regardless of ability to accurately complete the tasks. According to our exit survey, 13 (65%) are Android users, and the remaining 7 (35%) use iOS.

Participants were provided with a short guideline document containing instructions on how to authenticate to the site, how to navigate, and what tasks to perform. Brief appendices in that document detailed the filtering and review components they would be using.

6.3.2 Procedure

6.3.2.1 Dataset

The sample dataset was created from one of the researcher's (heavily-filtered) location and search history archive. Personal information, including the current home and work location of the researcher, was redacted from the sample dataset, although the prior home and work locations of the individual were retained for realism. The dataset spans from April 13th, 2016 to November 6th, 2016, and includes $146,068$ location points across multiple continents and $5,403$ Google searches.

6.3.2.2 Filtering Tasks

The study was partitioned into two arms, "For Myself" and "For a Family Member"; participants were randomly assigned to an arm in a round-robin fashion. Depending on the arm, the user was prompted at the start of the exercise to pretend that the data belonged either to themselves or to a family member, and that they would be filtering the data for that individual prior to releasing it to a life coach or therapist. Individuals who were in the "For Myself" arm of the study were also provided with the precise home and work locations for the sample dataset.

The participants were asked to complete the following tasks (verbatim from the instructions) to the best of their ability:

1. "Find the user's home location via the 'review' page's 'location' tab. Remove it using

the location filters in the 'filter' page. (It's in Manhattan, NY, USA.)"

2. "Find the user's work location in the same way and remove it as well. (It's also in Manhattan.)"
3. "Find when there was a vacation in Iceland. Remove all location and searches during that time period."
4. "Remove all searches on Saturdays."
5. "Remove searches that include the words 'jersey' or 'nj'. (Note that you should remove just those words, not words that contain 'nj', e.g., 'ninja'.)"

The result of this filtering exercise was compared against a reference dataset we built by following the filtering tasks on the sample dataset. Specifically, we compared the overlap between the data that remained in each users' filtered output to the reference dataset to determine how well they were able to accomplish each task. We also collected the filter definitions that the users created, whether they applied to the tasks or not, which are explored in Section 6.4.2.

6.3.2.3 Exit Survey

Participants were required to fill out an exit survey consisting of questions about the interface and the study procedure itself. The exit survey contained a demographic-related question, a few ranked impression questions, and free response questions. Ranked-impression results are summarized in Table 6.1; the questions are phrased as statements and responses were ranked on a Likert scale [Lik32], ranging from 1, "Strongly Disagree", to 5, "Strongly Agree". The free-response prompts are listed in Table 6.3. The responses to these prompts and how they color our interpetation of the study are discussed in Section 6.6. Participants were also asked two quality-control "yes"/"no" questions, listed in 6.2, to assess if technical issues had impacted their performance.

Survey response statistics are listed and discussed in Section 6.4.3.

ID	*Prompt*
Task-related	
Q1	"The tasks I was asked to do were easy to follow compared to other tasks."
Q2	"The tasks I was asked to do had a single obvious way to complete them."
Q3	"I was confident that I completed the task correctly."
Q4	"The tasks that I was asked to do are tasks that I would want to perform on my own data."
Q5	"The filtering options were sufficient for the tasks."
Interface-related	
Q6	"The login process was simple."
Q7	"It was easy to navigate the site."
Q8	"The website loaded quickly."
Q9	"Interacting with the site felt fluid."
Q10	"There was sufficient help text on the site to explain how it worked."

Table 6.1: Impression Questions

Prompt	*Yes*	*No*	*Percent Y/N*
Did you encounter any bugs while using the site?	4	16	20% / 80%
Did you notice any visual issues while using the site?	2	18	10% / 90%

Table 6.2: Site Functionality Questions

6.4 Results

6.4.1 Filtering Overlap

We use the Jaccard index to compare the overlap between the reference dataset and each participants' filtered dataset, defined as the intersection of the points from the two sets divided by their union. We chose this statistic in order penalize both under- and over-filtering relative to the reference. The Jaccard index ranges from 0, the inverse of the reference set, to 1, perfectly overlapping the reference. The Jaccard index of the sample dataset and the referenced filtered dataset is 0.939 for search, and 0.276 for location, indicating that most

Prompt	*# of Responses*
Briefly describe other desired filtering tasks, if any.	11
Which filtering tasks, if any, were unnecessary?	9
Include any other comments about the site, e.g., specific issues you may have encountered, if any.	13

Table 6.3: Optional Free Response Questions

of the searches were retained whereas most of the locations were removed. The scores are included in Table 6.4. Entries that match the sampe dataset's Jaccard index are in bold, indicating that the user didn't create any filters. Section 6.6.1 contains comments on this scoring choice.

User ID	*Location*	*Search*
user-1	**0.939**	**0.276**
user-2	0.958	0.83
user-4	0.968	0.275
user-5	0.927	0.616
user-6	0.967	0.819
user-7	0.968	0.683
user-8	0.939	0.62
user-9	**0.939**	0.276
user-10	1	0.561
user-11	0.955	0.778
user-12	**0.939**	**0.276**
user-13	0.872	0.284
user-14	0.968	0.684
user-15	0.954	0.495
user-16	1	0.561
user-17	0.966	0.823
user-18	0.968	0.684
user-19	**0.939**	**0.276**
user-20	0.962	0.684
user-21	0.968	0.244

Table 6.4: Per-User Scores vs. Reference

We investigated which survey factors corresponded to these scores and summarized the most relevant results in a heatmap. Figure 6.1 shows a centered and scaled version of the scores on a per-user basis; each column was centered around its mean and divided by its variance in order to make the scores comparable to each other and to the ranked responses. Rows and columns were arranged via clustering by Ward distance. (Specifically, the value "`Ward.D2`" was used for the `clustering_method` parameter to R's `pheatmap` function.)

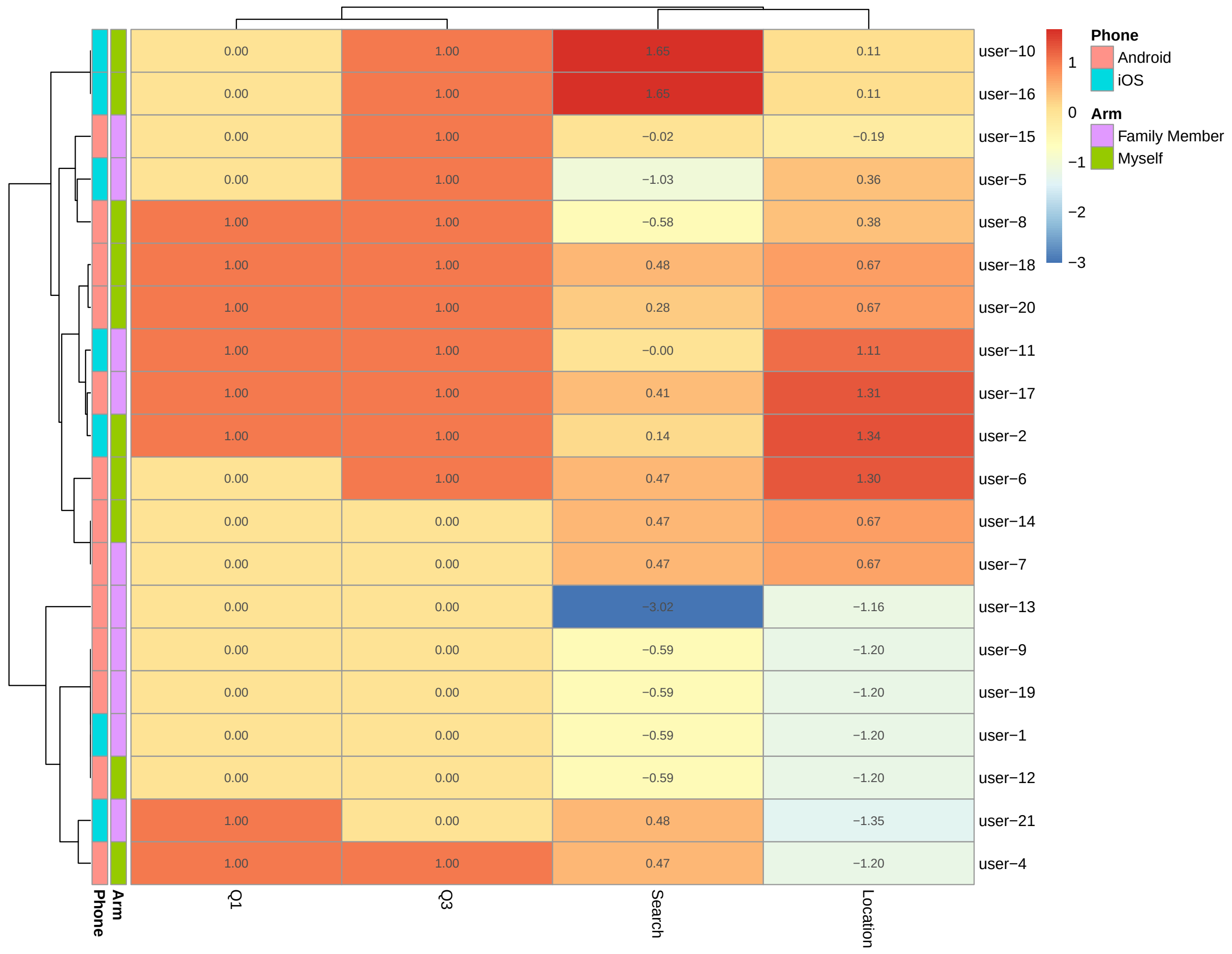


Figure 6.1: Heatmap of Search and Location Scores, Prompts

Q1 is a question from the exit survey on how easy the tasks were compared to others they had encountered on MTurk, and Q3 is about how confident the participant was in their accuracy. These have been thresholded to 0 for responses which were lower than the average for that question and 1 for responses that were higher.

There is a visible correspondence between confidence and performance; users who reported higher confidence tended to perform better. Surprisingly, the study arm and performance on the location-related tasks were not strongly correlated, despite participants being given the home and work location of the user. The notably negative search outlier, user-13, is due to the participant flagging every query containing “nj” *except* the one which contained it as a word. We examine cases like user-13’s more closely in the following section.

6.4.2 Individual Task Performance

The Jaccard index gives an overall estimation of the user’s performance on the tasks, but it does not communicate fine-grained information about which tasks they failed on or why. To that end, we examined the filter definitions that the users had created, and discovered that users had either misunderstood parts of the interface or otherwise declined (perhaps after trying and failing) to address the tasks at all.

For tasks 1 and 2, identifying home and work locations, we queried for participants’ location filters that were within 2.5km of the target (roughly half the distance between the sample user’s actual work and home). For task 3, removing searches during the time that the user was away on vacation, we queried for time filters that weren’t weekly-recurrent (e.g. either single dates or ranges). To identify spurious location filters created for task 3, assumedly under the misunderstanding that it was a location and not a search filtering task, we queried for location filters that were outside of a 2.5km radius of the sample user’s home or work location. For task 4, removing Saturday searches, we queried for weekly recurrence time filters. Finally, for task 5, removing the words “jersey” and “nj” while retaining words that contain “nj”, we simply queried for any search filter, since that was the only solely search-related task.

Tasks 1 and 2 were mostly successfully accomplished by the users who attempted them. One user created multiple redundant filters on the same location. All but two of the users, regardless of study arm, were able to find and filter the home and work location to within 300 meters. One user who had been told the home and work locations placed the work marker 562 meters away, a distance of several blocks. A user who had not been told the home and work locations placed a marker on the west side of Central Park, but left it unclear if it was intended to be a home or a work filter. The filter is entitled "Manhattan, NY, USA", which likely non-incidentally is the precise location pinpointed when searching for that string on the map.

Task 3 proved to be a major stumbling block. The phrasing of the task was that users should remove location and search history in the *time period*, yet five users attempted to filter out arbitrary locations in Iceland, four around Reykjavik (which was, in fact, where the sample user had dwelled for most of the trip), and one in the center of Iceland. User-20 went so far as to search for any terms containing "Iceland" and removed them from the search history. All the individuals who were able to identify that it was a time range also selected the correct modes (location, search) for the filter.

Task 4, removing search history for Saturdays, also produced a lot of erroneous results. While the 15 participants who attempted it all were able to filter Saturday, only 4 succeeded in specifying only the "search" mode as requested. 10 specified all the modes, while two specified solely "browser history".

Finally, of the 15 participants who attempted task 5, 12 were successful in removing the single matching search entry. One user had two filters defined named "jersey" and "nj", but neglected to exclude anything. User-13 excluded everything that matched "nj" *except* the target search entry, resulting in their low search score. User-20 also removed all terms containing "nj", including the target. User-20 also searched for and excluded all searches that contained the terms "Manhattan" or "Iceland".

The interpretation of these issues in terms of the interface and how they can inform future development is discussed in Section 6.6.2.

6.4.3 Exit Survey

Figures 6.2 and 6.3 show the distribution of responses for prompts relating to the tasks and prompts relating to the interface, respectively. The boxplot shows the average response as a solid line in the center of the box, with the extents of the box containing 50% of the responses. The lines extending from the boxes capture the remaining 90%; points that lie outside of the lines are outliers.

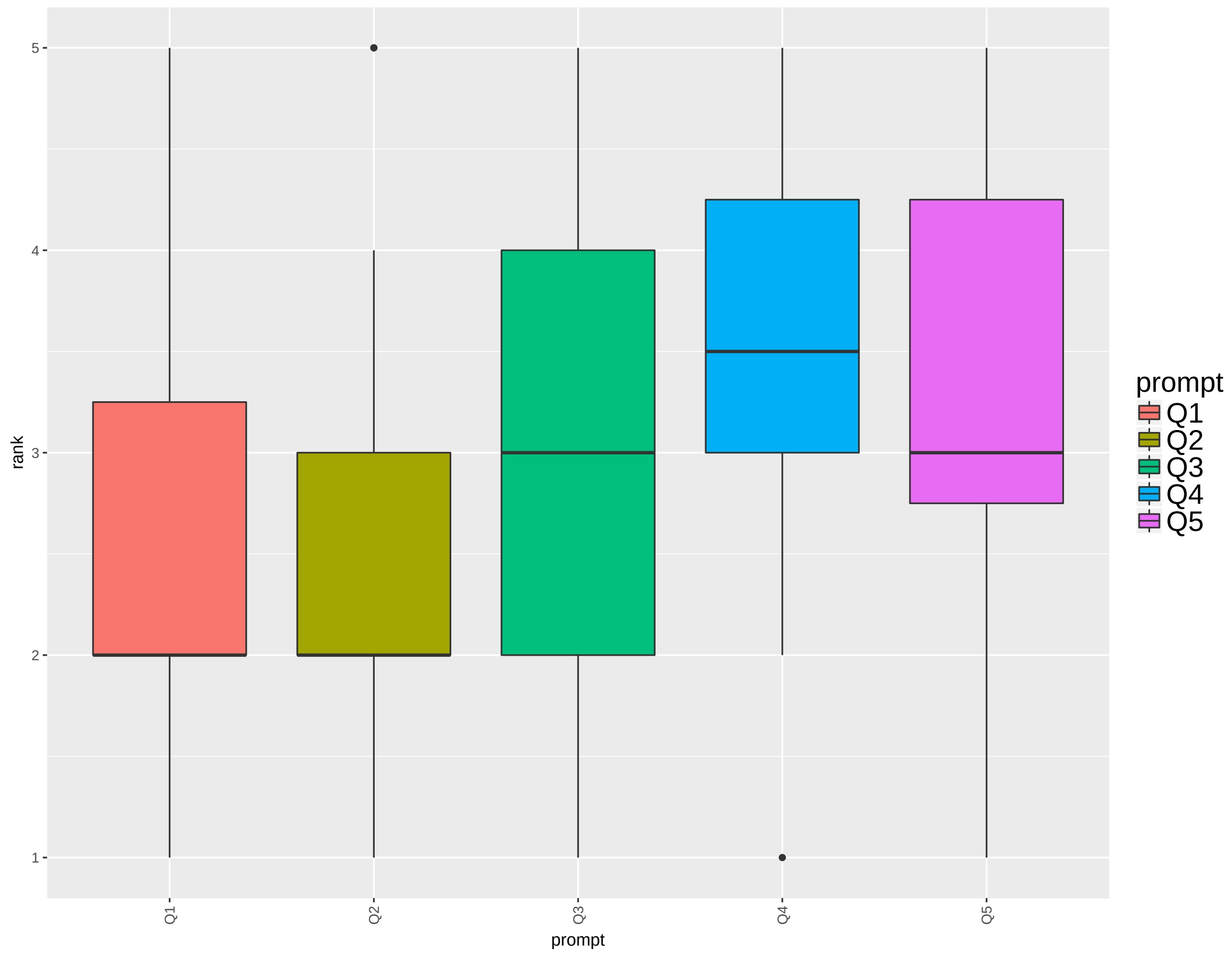


Figure 6.2: Task-related Prompt Responses

All but one user found the login process simple, which is not surprising since we skipped the archive acquisition step in this study. The one user who reported less than “Strongly Agree” on Q6 mentioned the following in their free comments response: “I found the Google data import process difficult to understand. Also when analyzing data at the end it never completed.” Upon inspection the user had indeed attempted to investigate parts of the site not mentioned in the guidelines.

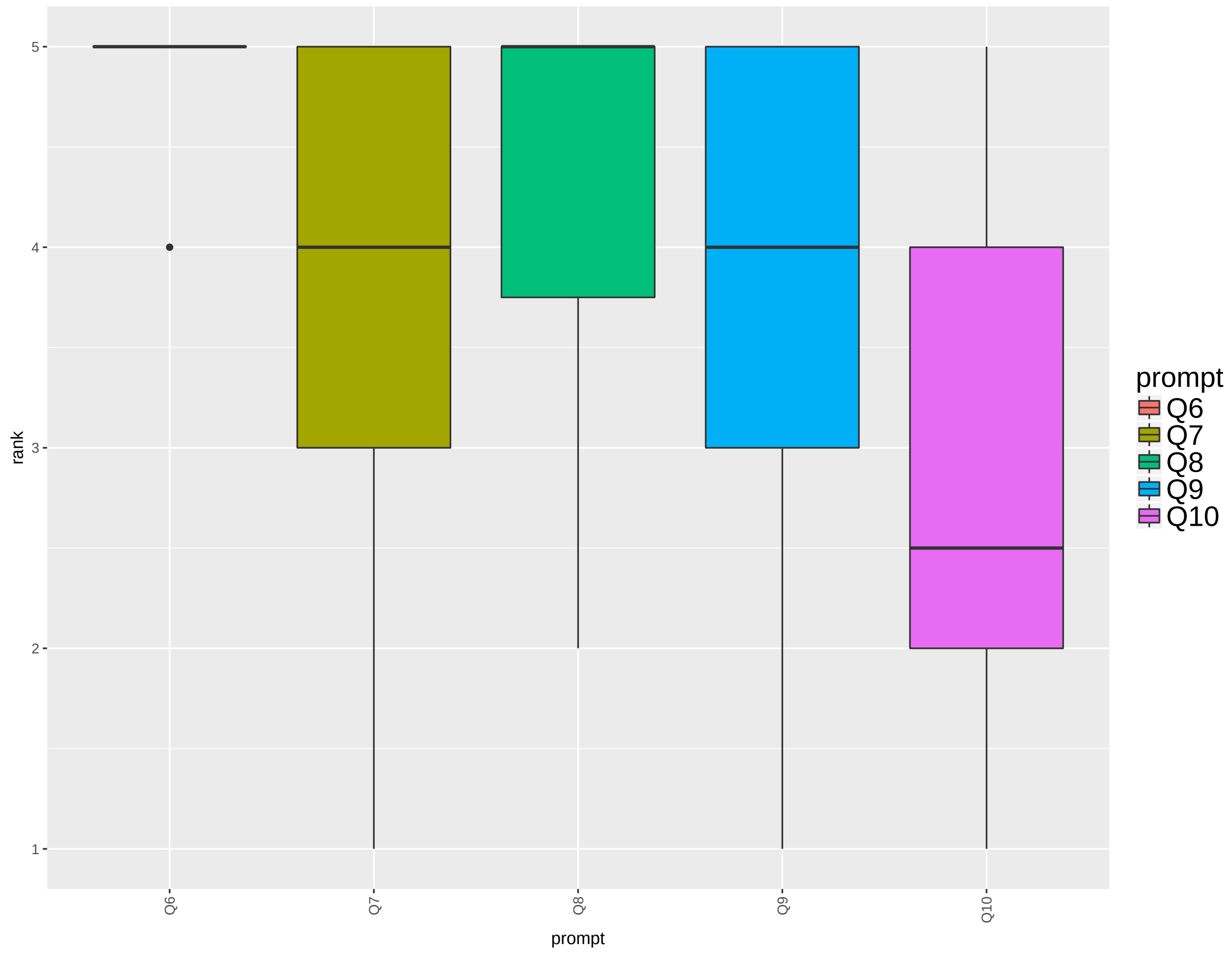


Figure 6.3: Interface-related Prompt Responses

6.4.3.1 Quality Control Prompts

Of the 20 users in the study, four reported that they encountered a bug, and two reported visual issues. One early participant encountered an insurmountable issue with the site that required a hotfix; they were able to complete the study after the fix had been implemented.

6.5 Interface Description

The interface currently allows importing three data modes: location data (collected via Google's location services on Android devices, and with lower resolution on iOS devices), Google Search history, and Google Chrome browser history.

The user-facing portion of the pipeline is composed of three stages:

- **Acquisition**, in which the user provides their Takeout archive to the interface.
- **Filtering**, in which the user establishes filtering rules to exclude data from the result.
- **Review**, in which the user reviews the results of their filtering rules, as well as determines what data needs to be filtered.

Ordinarily a user would proceed through all three stages. In this study, we have skipped the acquisition stage by providing a sample dataset to the study participants, which allows us to compare their performances on the same dataset for their tasks.

6.5.1 Stage 1: Acquisition

Acquisition consists of the user first visiting Google Takeout to export their archive and selecting Google Drive as the storage location. The user returns to the interface, authenticates themselves to Google Drive via OAuth2, and then selects their archive from their Drive account. This kicks off the process of parsing and importing their archive into the interface, allowing them to proceed to the next two stages, Filtering and Review.

Filtering and Review can be performed in either order. Ideally, the user would switch between filtering and reviewing repeatedly to determine if their data has been adequately filtered.

6.5.2 Stage 2: Filtering

In the Filtering stage, the user creates exclusions over their dataset. Like the Review stage, these are organized by data mode, although a fourth filter type, "Time", applies to all of the modes. Users can create as many filters of each type as they like, with the filters 'or'-ed together to remove the union of the matching results from the dataset.

The **location filter**, shown in Figure 6.4, allows users to interactively specify locations on a map to filter out. Users can either click on the map to place the circle, or use the search box to enter an approximate or exact address. A 100 meter circle around the selected area is excluded from the results. Each filtered location appears as a separate "tile", i.e., a small map with the filtered area shown as a red 100m circle around a marker at the center.

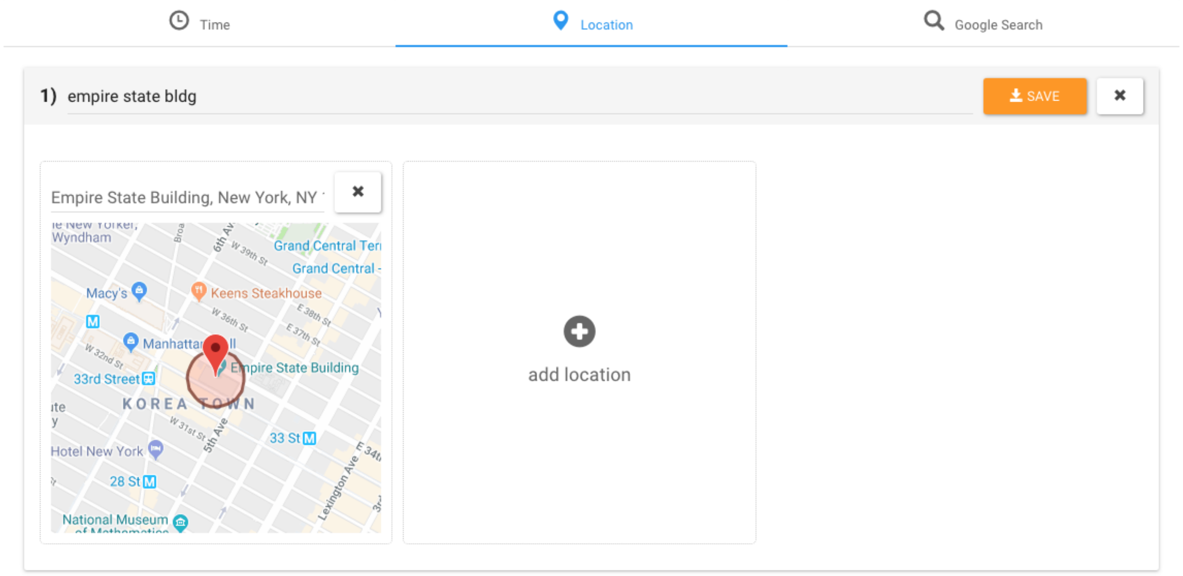


Figure 6.4: Location Filter

The **search and browser history filters**, presented separately in the interface but described together here due to their similarities, allow users to filter out individual searches and browser history entries, respectively. The filter consists of a search box and a table which initially lists all results, but can be filtered by a search query. Checkboxes alongside each entry indicate whether or not that specific item has been filtered out; the checkbox in the table header toggles the selection status of everything in the table. Once the user has selected the entries they wish to exclude, pressing the "Exclude Items" button adds an entry to the list of exclusions on the right side. The entries are labeled according to the term that was searched. Users can edit an entry by selecting it, changing their checkbox selections, and then pressing the "Exclude Items" button again to update the entry. Figure 6.5 shows the search filter for a sample query.

Finally, the **time filter** cuts across the rest of the modes, allowing the user to remove slices of time from any of the other data modes. A set of checkboxes indicates the modes to

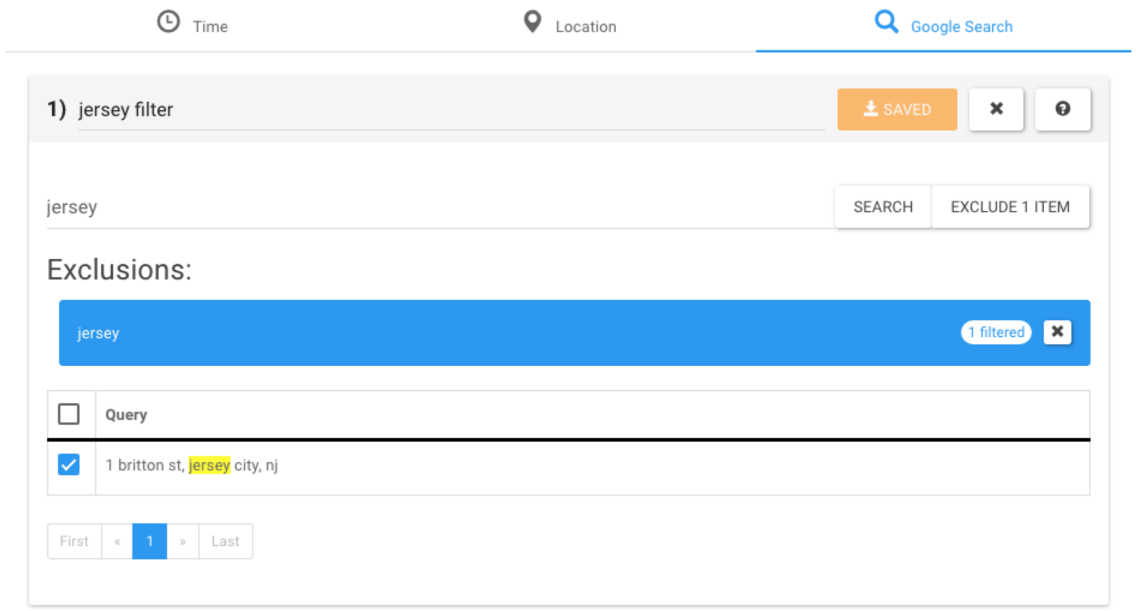


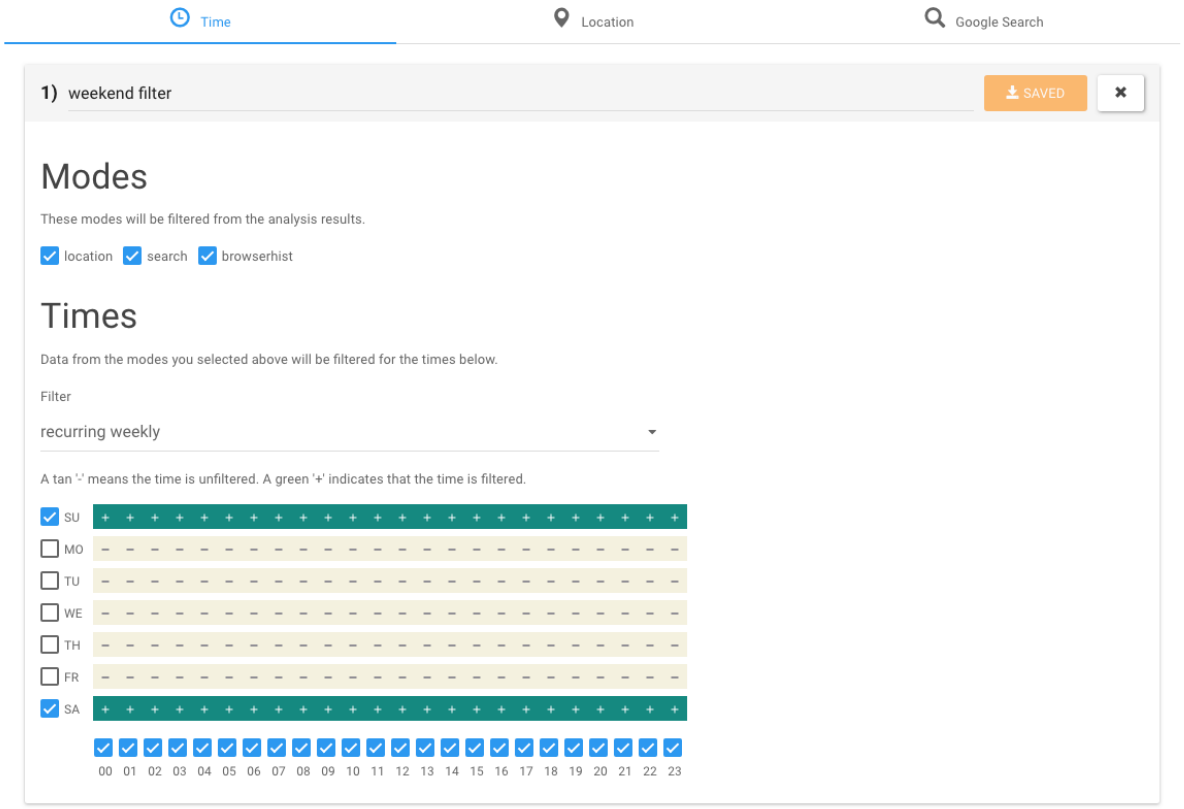
Figure 6.5: Search Filter with Sample Query "jersey"

which the time filter will be applied. Intervals can be specified on three bases: 1) a single day, 2) a range of days, or 3) on a recurring day-of-week/hour-of-day basis. The weekly recurrence option is depicted in Figure 6.6; clicking an entry in the matrix will remove entries for that day-of-week/hour-of-day pair across the entire dataset. Users can easily toggle removal of an entire day of the week or hour of the day by clicking the corresponding checkbox in the margin.

6.5.3 Stage 3: Review

In the Review stage, the user can view their data via mode-specific views. The data displayed in these views is the *filtered* data, although initially all the data are visible. In brief, the location view visualizes the user's location history, search shows their Google searches, and browser history displays the URLs they have accessed via Chrome. During Review, the user can both identify data they wish to filter as well as determine that their filters, created in the Filter stage, are being adequately applied.

Specifically, the **location view** displays a heat map of the user's visited locations along



Time | Location | Google Search

1) weekend filter SAVED ✖

Modes

These modes will be filtered from the analysis results.

✓ location ✓ search ✓ browserhist

Times

Data from the modes you selected above will be filtered for the times below.

Filter

recurring weekly

A tan '-' means the time is unfiltered. A green '+' indicates that the time is filtered.

✓	SU	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
☐	MO	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
☐	TU	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
☐	WE	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
☐	TH	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
☐	FR	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
✓	SA	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Figure 6.6: Time Filter, Weekly Recurrence

with a time series showing the amount of recorded location points over time, shown in Figure 6.7. Selecting a span of the time series constrains the data displayed on the map; selecting a bounding rectangle on the map similarly constrains the time series to data for just the selected region. By using these two representations of the location trace in combination, a user can determine when they were at a specific location, as well as where they were at some given point in time.

The **search and browser history** are very similar to each other, consisting of a search box, a time series, and a table which initially shows the search text or visited URL (depending on the mode) sorted chronologically. Figure 6.8 depicts the Google search history. A list to the side of the table displays the top ten most frequent entities for that mode; in the search history case, these are words, while domains (e.g., "github.com") are listed in the browser history case. A separate "recurrence" tab switches the list from overall frequency to a count

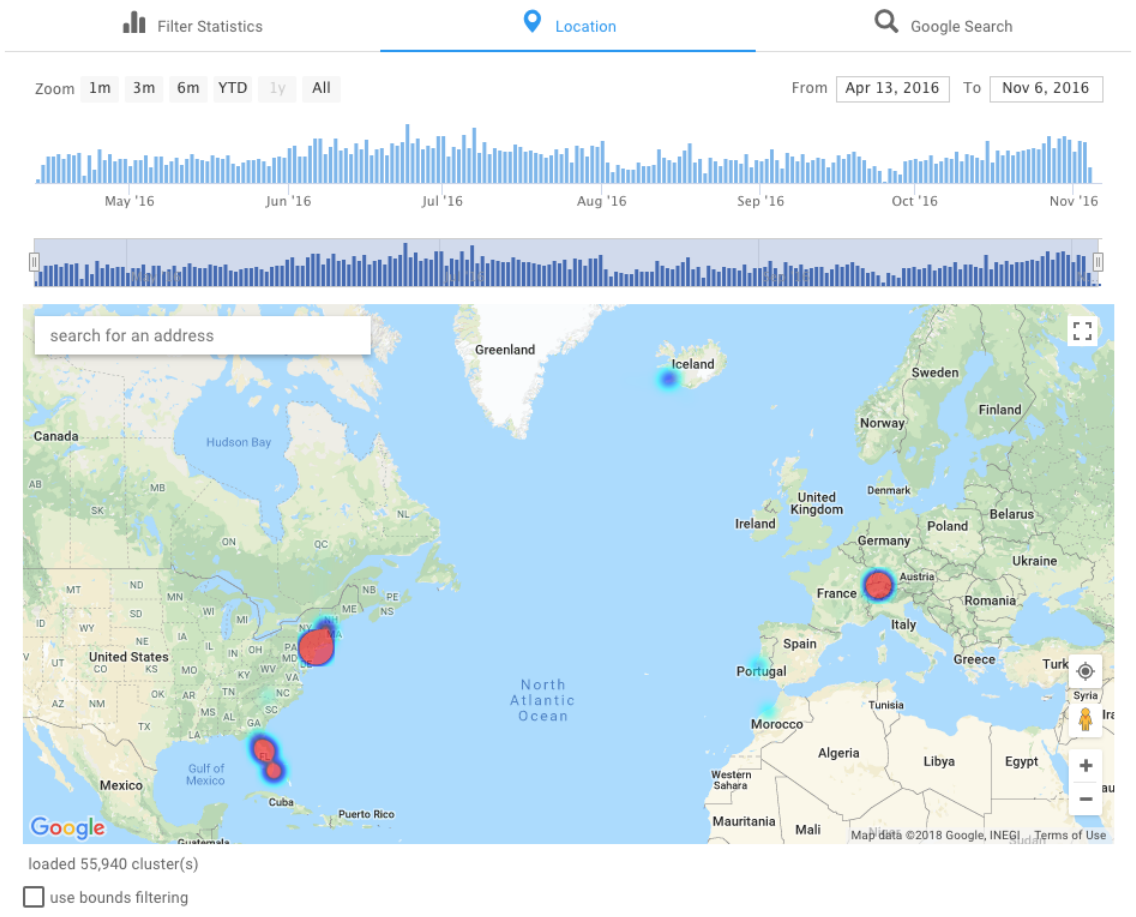
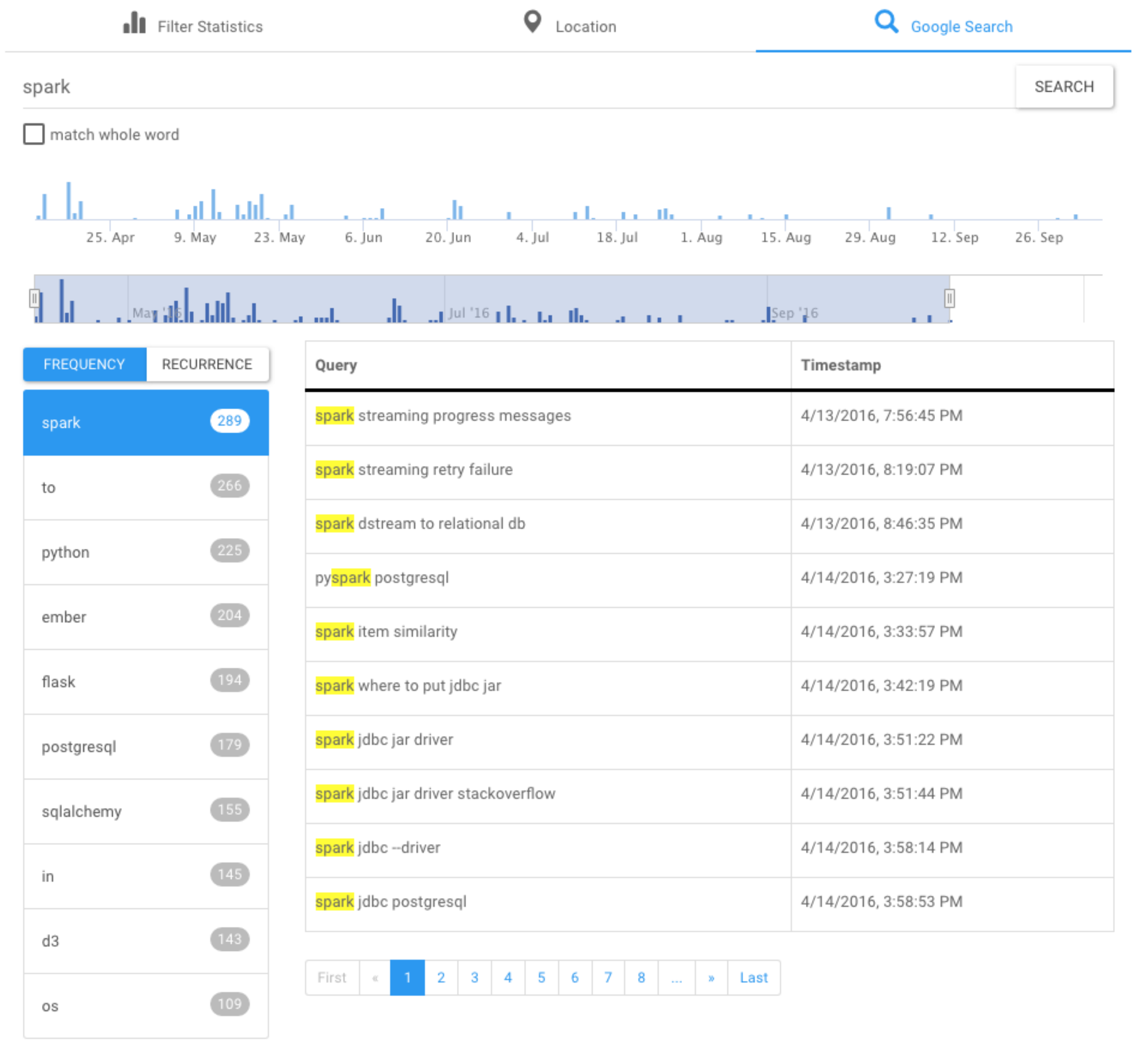


Figure 6.7: Location Heat Map with Time Series

of the times that the entity occurred within each weekly interval, indicating entities which consistently occur, even at low frequency. Entering queries in the search box will filter the table to just entries that contain the term. Similar to the location view, selecting spans of time in the time series will restrict the results (both the table and frequent/recurrent entity lists) to just entries that fall within that time-span.

Finally, the Review stage includes a fourth view, the **filtering summary** view, shown in Figure 6.9, which displays information on how much of the data one has chosen to filter out. This is displayed both in raw counts of remaining data points, as well as in a bar chart showing the proportion of total to unfiltered data remaining.



spark

☐ match whole word

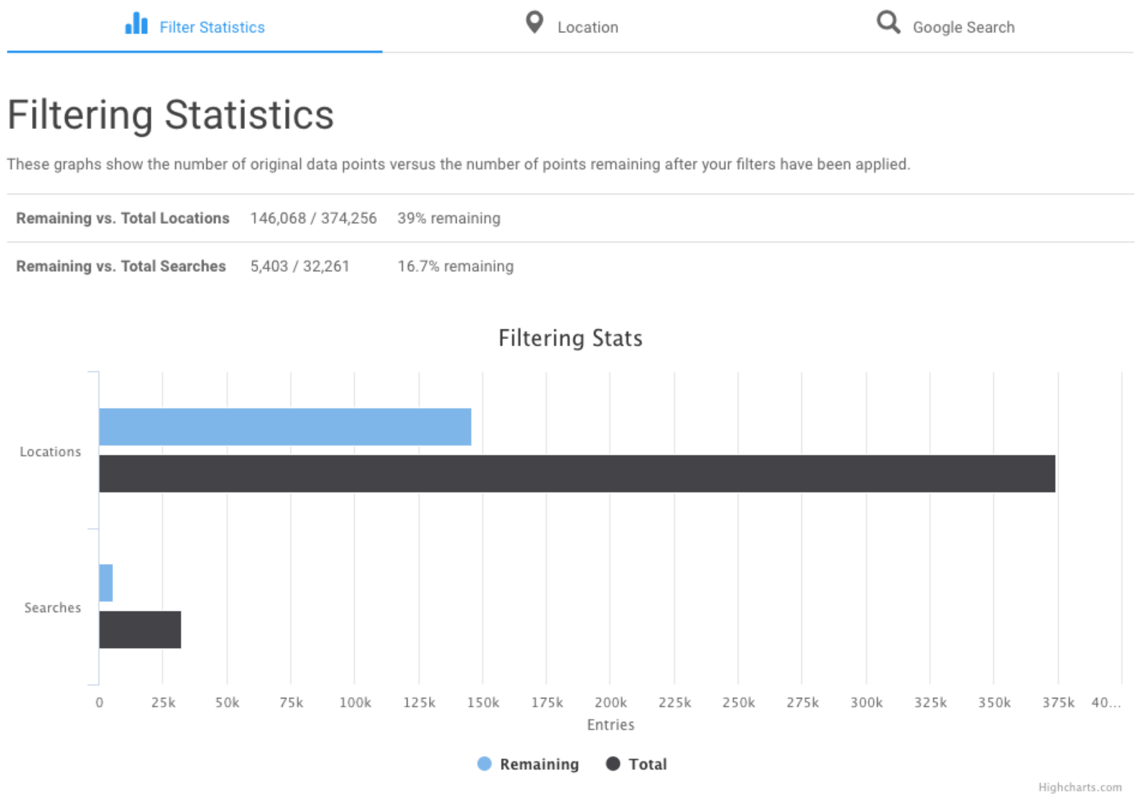
spark	289
to	266
python	225
ember	204
flask	194
postgresql	179
sqlalchemy	155
in	145
d3	143
os	109

Query	Timestamp
spark streaming progress messages	4/13/2016, 7:56:45 PM
spark streaming retry failure	4/13/2016, 8:19:07 PM
spark dstream to relational db	4/13/2016, 8:46:35 PM
pyspark postgresql	4/14/2016, 3:27:19 PM
spark item similarity	4/14/2016, 3:33:57 PM
spark where to put jdbc jar	4/14/2016, 3:42:19 PM
spark jdbc jar driver	4/14/2016, 3:51:22 PM
spark jdbc jar driver stackoverflow	4/14/2016, 3:51:44 PM
spark jdbc --driver	4/14/2016, 3:58:14 PM
spark jdbc postgresql	4/14/2016, 3:58:53 PM

Figure 6.8: Google Search History

6.6 Discussion and Future Work

While the quantitative portion of the study verified the fact that participants were having difficulty with the tasks, it was mostly the individual task breakdown and exit survey that served to pinpoint and contextualize the issues. Users had difficulty knowing which parts of the high-level tasks corresponded to what parts of the interface, and many voiced concerns over being unable to see their filtering changes reflected in the Review pages. We believe these issues centered around a lack of inline documentation, which in the absence of direct supervision is, as we have learned, essential.



Filtering Statistics

These graphs show the number of original data points versus the number of points remaining after your filters have been applied.

Remaining vs. Total Locations	146,068 / 374,256	39% remaining
Remaining vs. Total Searches	5,403 / 32,261	16.7% remaining

Figure 6.9: Filtering Summary

It should also be noted that the filtering tasks contained ambiguities; for instance, in task 3 users were expected to filter out the time period between when the sample user left from home to Iceland and returned from Iceland back home, but as one astute tester pointed out the sample user did not remain in Iceland during the entire period – there was a brief trip to mainland Europe in the middle of it. Task 3 also requested that the user filter location and search data from the time interval, which could validly be interpreted as filtering out the location data in Iceland and the search history as separate tasks (which some participants did attempt to do via location filters in Iceland).

The tasks involving locating the users' home and work locations could have been more loosely evaluated, since filtering out a larger area than precisely 100 meters around the exact location would not be a problem in practice. In terms of task coverage, search was unfortunately given less attention than it could have been, especially the Google search history view, which was not required at all to solve the given tasks. It would be beneficial to have tasks in future studies which require the use of the search history view, e.g., a task in which the user explores frequent terms and filters out highly recurrent ones of a particular category.

6.6.1 Filtered Data Overlap Scoring

The choice of the Jaccard index as a score for task accuracy came with a few consequences that were realized after the study. One is that, in a real-world scenario, it is more prudent for privacy's sake to remove more of the points than less; the Jaccard index being symmetric does not reflect that concern. In the location case, filtering areas that overlap with higher densities of the location trace has more of an impact on the score, which biases the score toward tasks that relate to these high-density areas (e.g., work and home). A similar phenomenon can occur in the search case when searching for a short sequence; filtering all of these occurrences would have a larger impact than filtering longer and thus more rare sequences. This can be seen in user-13's response to task 5, when filtering the common sequence "nj" without concern for its overlap with unrelated entries produced a dramatic effect on their score, in conjunction with their possible confusion of exclusion versus inclusion in the interface. Similar errors, for instance user-20's filtering of "Manhattan" and "Iceland" did not affect their score in the same way.

6.6.2 Task-level Analysis

Granted that most of the participants who attempted to filter the home and work locations were able to identify them, it seems likely that the location history view is sufficient for that task. Users encountered significant issues with establishing a correlation between time and location using the "bounds filtering" feature, which was required for the Iceland task; the feature could be made more apparent or, as one user suggested, the current zoomed-in area could serve as the bounds filter.

Users appeared to have difficulty removing just the data modes specified in the tasks with the time filter, evidenced in both tasks 3 and 4. Interestingly the few users who were attentive to removing "browser history" in task 3 were not always attentive to the modes in other tasks; half of those users failed to uncheck "location" in task 4. All users who attempted task 4 were able to filter Saturdays out without issue, leading us to believe that the day-of-week/hour-of-day matrix for weekly recurrent filtering is comprehensible after all.

The only stricly search-related task, task 5, consisted of a redundant request to remove both "jersey" and "nj" – both words were found in the same search entry. Nevertheless, of the 15 who attempted the task, only 4 used a single exclusion to filter the entry. The remaining 11 used at least two filters, typically "jersey" and "nj". (Entries that are previously covered by another filter are displayed as initially checked, but perhaps this is not immediately apparent.) The fact that user-13 filtered out every entry but the relevant one seems to indicate that they thought the checkboxes indicated inclusion, not exclusion. Some users created empty filters, indicating perhaps that they had difficulty understanding it immediately and were experimenting.

6.6.3 Exit Survey Comments

To our welcome surprise, the majority of users elected to leave final comments despite the prompt being optional. Some of the users used the opportunity to express their frustration or apology for their perceived inability to do the tasks, while others pointed out ambiguities in the tasks and explained their choices.

For instance, a user cites the lack of inline help and insufficiency of the guidelines as a confounder:

> *there is no help on the site what so ever, the instructions are incomplete, it is impossible to blindly figure this out at all. Additional help is nothing more then definitions and does not instruct on what needs to be done in order to get a desired result.*

Another user mentioned the lack of inline help, but found the guidelines given with the task to be sufficient:

> *I did not have any issues with the site itself. Sufficient help text was provided on the HIT itself rather than on the site. As such, I indicated that there insufficient help text on the site as it wasn't on the site but on the HIT.*

Of course, bugs and oversights in the design also hampered the evaluation. One participant was unable to visibly verify their filtering changes in the review stage:

> *When you go to the review page the graph changes...but it's not obvious *what* changes were made. Or, if it was even the right one. The map itself doesn't change either. The only graphical response I had was the overall look of the line graph placement. I had no idea if the changes I made actually did what they were supposed to.*

These comments illustrate that inline documentation and clear presentation are just as important to the usability of a system as its actual features. The ambiguities in the task definition further drive home the message that real data are messy; one can easily be left unaware of the caveats in their assumptions, for instance traveling incidentally to other locations during what they considered a vacation to a single place.

Many users mentioned the interesting possibility of providing some kind of automated guidance on creating filters, for instance: "I'd like to see something automatic or prompted. Meaning I could set it to automatically filter if I went to Vegas (because what happens there stays there, etc.)" Another user commented "favorite places like stores/shops", assumedly meaning that they would like to see a list of frequent places perhaps similar to the frequent/recurrent terms in the search history. The Review stage was intended to enable informed filtering, but it would be interesting to extend it into intelligent prompting and automated filtering.

6.7 Conclusion

This work presents an evaluation of the feasibility of a Takeout archive filtering interface for a general audience, specifically a pool of participants from Amazon Mechanical Turk. We performed a quantitative analysis of the overlap of their filtered data with a reference set, and conducted an exit survey to help contextualize the quantitative results and obtain other insights about the site.

Even without in-person coaching, a few individuals were able to complete the tasks to some degree of success, and while much of the free-response feedback was critical, it was helpful to be told which areas of the documentation and interface needed attention. The tester's "pain points" in using the site, both the ones we determined from inspecting their performance and the ones they self-reported, were surprisingly valuable, not to mention their ideas on what they would like to see in the project. We intend to conduct further evaluations in the same vein with larger sample sizes, and potentially integrate general-audience testing – alongside our existing internal and supervised tests – as a regular part of our development cycle.

CHAPTER 7

Conclusion, Future Work

This thesis introduces the small data context and its opportunities and challenges. It details the components that comprise the small data ecosystem and several projects which have implemented portions of that ecosystem. The first project involve feature extraction on a particularly challenging form of small data, natural-language computer-mediated communication, and implemented a mechanism to grant access to that derived data to authorized downstream components. Cross-cutting concerns in developing small data applications are explored, and a middle-layer system is proposed and implemented which handles these concerns. Finally, the work reiterates the importance of incorporating the user as a mediator of access to their data and an active participant in filtering their data, so that it can be ethically employed in user-facing tools and research studies.

There is still a vast amount of work to be done in realizing the small data ecosystem. For instance, this work only scratches the surface of the security implications inherent in collecting personal data, and the policies that must be in place, both technological and legal, before such systems can be trusted at a larger scale. While the architecture outlines where components that deal with rich data (such as natural-language parsers or image classifiers) should go, the implementations of many of these components are still works in progress. In some sense the small data ecosystem will never be fully complete, since new data modes and means of comprehending and presenting that data are continuously being invented. Nevertheless, the work included in this thesis provides a starting place for acquiring and filtering the data that will drive these applications, and includes a variety of applications that can be built on top of the ecosystem as it stands today.

REFERENCES

[ABE15] Julio Angulo, Stefan Berthold, Kaoutar Elkhiyaoui, M Carmen Fernandez Gago, Simone Fischer-Hübner, Nunez David, Tobias Pulls, Jenni Reuben, Cédric Van Rompay, Anderson Santana de Oliveira, et al. "D: D-5.3 User-Centric Transparency Tools V2.", 2015.

[AKE14] Faisal Alquaddoomi, Cameron Ketcham, and Deborah Estrin. "The Email Analysis Framework: Aiding the Analysis of Personal Natural Language Texts." In Federica Cena, Altigran Soares da Silva, and Christoph Trattner, editors, *Hypertext 2014 Extended Proceedings*, volume 1210 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

[And18] "Intents and Intent Filters." `https://developer.android.com/guide/components/intents-filters`, 2018. Accessed: 2018-07-26.

[App] Apple Inc. "HealthKit." Retrieved on 2018.03.19 from `https://developer.apple.com/healthkit/`.

[BBB14] Dror Ben-Zeev, Christopher J Brenner, Mark Begale, Jennifer Duffecy, David C Mohr, and Kim T Mueser. "Feasibility, acceptability, and preliminary efficacy of a smartphone intervention for schizophrenia." *Schizophrenia bulletin*, **40**(6):1244–1253, 2014.

[Bir06] Steven Bird. "NLTK: the natural language toolkit." In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pp. 69–72. Association for Computational Linguistics, 2006.

[Bos09] Jan Bosch. "From software product lines to software ecosystems." In *Proceedings of the 13th international software product line conference*, pp. 111–119. Carnegie Mellon University, 2009.

[CAJ16] Jean Costa, Alexander T Adams, Malte F Jung, François Guimbetiere, and Tanzeem Choudhury. "EmotionCheck: leveraging bodily signals and false feedback to regulate our emotions." In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 758–769. ACM, 2016.

[CDH05] Yuhan Cai, Xin Luna Dong, Alon Halevy, Jing Michelle Liu, and Jayant Madhavan. "Personal Information Management with SEMEX." In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pp. 921–923, New York, NY, USA, 2005. ACM.

[CFT07] Dan Cosley, Dan Frankowski, Loren Terveen, and John Riedl. "SuggestBot: Using Intelligent Task Routing to Help People Find Work in Wikipedia." In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, IUI '07, pp. 32–41, New York, NY, USA, 2007. ACM.

[Cha18] Louisa Chan. "How to Use Google Takeout for Archiving and eMails Import.", 2018. Retrieved April 13, 2018 from `http://www.louisachan.com/google-takeout-emails-import/`.

[CHS12] Connie Chen, David Haddad, Joshua Selsky, Julia E Hoffman, Richard L Kravitz, Deborah E Estrin, and Ida Sim. "Making sense of mobile health data: an open architecture to improve individual-and population-level health." *Journal of medical Internet research*, **14**(4), 2012.

[CI97] Pei Cao and Sandy Irani. "Cost-Aware WWW Proxy Caching Algorithms." In *Proceedings of the USENIX Symposium on Internet Technologies and SystemsMonterey, California, December 1997*, pp. 193–206, 1997.

[CMB02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. "GATE: an architecture for development of robust HLT applications." In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 168–175. Association for Computational Linguistics, 2002.

[CMT08] Sunny Consolvo, David W. McDonald, Tammy Toscos, Mike Y. Chen, Jon Froehlich, Beverly Harrison, Predrag Klasnja, Anthony LaMarca, Louis LeGrand, Ryan Libby, Ian Smith, and James A. Landay. "Activity Sensing in the Wild: A Field Trial of Ubifit Garden." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pp. 1797–1806, New York, NY, USA, 2008. ACM.

[ELK11] Michael D Ekstrand, Michael Ludwig, Jack Kolb, and John T Riedl. "LensKit: a modular recommender framework." In *Proceedings of the fifth ACM conference on Recommender systems*, pp. 349–350. ACM, 2011.

[Epi] Epic System Corp. "MyChart." Retrieved on 2018.03.19 from `https://mychart.deancare.com/mychart/`.

[Eur16] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)." *Official Journal of the European Union*, **L119**:1–88, May 2016.

[Fia14] Eva Fialová. "Data portability and informational self-determination." *Masaryk UJL & Tech.*, **8**:45, 2014.

[Fit11] Brian Fitzpatrick. "The Data Liberation Front Delivers Google Takeout.", 2011. Retrieved April 13, 2018 from `http://dataliberation.blogspot.ch/2011/06/data-liberation-front-delivers-google.html`.

[Fit14] FitnessKeeper, Inc. "Health Graph API.", 2014. Retrieved on March 19th, 2018 from `http://developer.runkeeper.com/healthgraph/`.

[FL04] David Ferrucci and Adam Lally. "UIMA: an architectural approach to unstructured information processing in the corporate research environment." *Natural Language Engineering*, **10**(3-4):327–348, 2004.

[Fou] "Foursquare." Retrieved on 2018.03.19 from `https://foursquare.com/`.

[go] "Google Fit." Retrieved on March 3rd, 2018 from `https://fit.google.com/`.

[goo17] "Google Feed: feed your need to know." `https://www.blog.google/products/search/feed-your-need-know/`, 2017. Accessed: 2018-07-26.

[GRF11] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. "MyMediaLite: A free recommender system library." In *Proceedings of the fifth ACM conference on Recommender systems*, pp. 305–308. ACM, 2011.

[GSP14] Trinabh Gupta, Rayman Preet Singh, Amar Phanishayee, Jaeyeon Jung, and Ratul Mahajan. "Bolt: Data Management for Connected Homes." In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pp. 243–256, Berkeley, CA, USA, 2014. USENIX Association.

[GVV13] Inge Graef, Jeroen Verschakelen, and Peggy Valcke. "Putting the right to data portability into a competition law perspective." *Law: The Journal of the Higher School of Economics, Annual Review*, pp. 53–63, 2013.

[GZZ13] M. Grabchak, Z. Zhang, and D. T. Zhang. "Authorship Attribution Using Entropy." *Journal of Quantitative Linguistics*, **20**(4):301–313, November 2013.

[Hal14] Harry Halpin. "Social Web Working Group Charter.", 2014. (accessed on 2018.03.19).

[HH11] DE Hammer-Lahav and D Hardt. "The OAuth2. 0 Authorization Protocol. 2011." Technical report, IETF Internet Draft, 2011.

[IFT14] IFTTT. "IFTTT: Put the internet to work for you.", 2014. (accessed on 2018.03.19).

[Inc18] Facebook Inc. "Downloading Your Info | Facebook Help Center.", 2018. Retrieved April 13, 2018 from `https://www.facebook.com/help/131112897028467`.

[ISE11] Molly E Ireland, Richard B Slatcher, Paul W Eastwick, Lauren E Scissors, Eli J Finkel, and James W Pennebaker. "Language style matching predicts relationship initiation and stability." *Psychological Science*, **22**(1):39–44, 2011.

[JC13] Slinger Jansen and Michael A Cusumano. "Defining software ecosystems: a survey of software platforms and business network governance." *Software ecosystems: analyzing and managing business networks in the software industry*, **13**, 2013.

[Jon07] William Jones. "Personal information management." *Annual review of information science and technology*, **41**(1):453–504, 2007.

[LG] Mark Linquist and Paul Galpern. “Crowdsourcing (in) Voluntary Citizen Geospatial Data from Google Android Smartphones.”.

[LG16] Mark Linquist and Paul Galpern. “Crowdsourcing (in) Voluntary Citizen Geospatial Data from Google Android Smartphones.” *Journal of Digital Landscape Architecture, 1-2016*, pp. 263–272, 2016.

[Lik32] Rensis Likert. “A technique for the measurement of attitudes.” *Archives of psychology*, 1932.

[LLM12] Mu Lin, Nicholas D. Lane, Mashfiqui Mohammod, Xiaochao Yang, Hong Lu, Giuseppe Cardone, Shahid Ali, Afsaneh Doryab, Ethan Berke, Andrew T. Campbell, and Tanzeem Choudhury. “BeWell+: Multi-dimensional Wellbeing Monitoring with Community-guided User Feedback and Energy Optimization.” In *Proceedings of the Conference on Wireless Health*, WH ’12, pp. 10:1–10:8, New York, NY, USA, 2012. ACM.

[loc16] “Location History Visualizer Pro.”, 2016. Retrieved April 13, 2018 from `https://locationhistoryvisualizer.com/`.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. “Amazon.Com Recommendations: Item-to-Item Collaborative Filtering.” *IEEE Internet Computing*, **7**(1):76–80, January 2003.

[Mal10] A Mallet. “Java-based packed for statistical NLP toolkit.” *Available at (accessed 26.01. 10)*, 2010.

[mara] “Net Market Share: Desktop Browsers.” `https://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=0`. Accessed: 2017-09-29.

[marb] “Net Market Share: Desktop Search Engine Usage.” `https://www.netmarketshare.com/search-engine-market-share.aspx?qprid=4&qpcustomd=0`. Accessed: 2017-09-29.

[marc] “Net Market Share: Mobile Browsers.” `https://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=1`. Accessed: 2017-09-29.

[mard] “Net Market Share: Mobile Search Engine Usage.” `https://www.netmarketshare.com/search-engine-market-share.aspx?qprid=4&qpcustomd=1`. Accessed: 2017-09-29.

[MHM16] Abhinav Mehrotra, Robert Hendley, and Mirco Musolesi. “PrefMiner: mining user’s preferences for intelligent mobile notification management.” In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 1223–1234. ACM, 2016.

[Mic] Microsoft. “HealthVault.” Retrieved on 2018.03.19 from `https://www.healthvault.com/`.

[mit] "MIT Media Lab's Immersion Project." `https://immersion.media.mit.edu/`. Accessed: 2014-02-04.

[Mov] "Moves API." (accessed on 2018.03.19).

[MS05] David G Messerschmitt, Clemens Szyperski, et al. "Software ecosystem: understanding an indispensable technology and industry." *MIT Press Books*, **1**, 2005.

[MSW14] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. "openpds: Protecting the privacy of metadata through safeanswers." *PloS one*, **9**(7), 2014.

[MUA10] Michael Martin, Jörg Unbehauen, and Sören Auer. "Improving the Performance of Semantic Web Applications with SPARQL Query Caching." In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *The Semantic Web: Research and Applications*, number 6089 in Lecture Notes in Computer Science, pp. 304–318. Springer Berlin Heidelberg, January 2010.

[NP02a] Kate G Niederhoffer and James W Pennebaker. "Linguistic style matching in social interaction." *Journal of Language and Social Psychology*, **21**(4):337–360, 2002.

[NP02b] Kate G. Niederhoffer and James W. Pennebaker. "Linguistic Style Matching in Social Interaction." *Journal of Language and Social Psychology*, **21**(4):337–360, December 2002.

[omh] "Ohmage DSU 1.0 Specification." `https://github.com/openmhealth/developer/wiki/DSU-1.0-Specification`. Accessed: 2014-02-04.

[ope18] "Open mHealth Developer Wiki.", April 2014, accessed on 2018-03-18. Retrieved on 2018.03.19 from `https://github.com/openmhealth/developer/wiki`.

[Per10] Sarah Perez. "Mobile Application Stores State of Play.", 2010. (accessed on 2018.03.19).

[PMN03] James W Pennebaker, Matthias R Mehl, and Kate G Niederhoffer. "Psychological aspects of natural language use: Our words, our selves." *Annual review of psychology*, **54**(1):547–577, 2003.

[PSK03] Fuchim Peng, Dale Schuurmans, Vlado Keselj, and Shaojun Wang. "Automated authorship attribution with character level language models." In *10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, pp. 267–274, 2003.

[RP04] Ellen Riloff and William Phillips. "An introduction to the sundance and autoslog systems." Technical report, Technical Report UUCS-04-015, School of Computing, University of Utah, 2004.

[Rus] Rustici Software. “xAPI.” Retrieved on 2018.03.19 from `https://xapi.com/`.

[Sch18] Schema.org Community Group. “Schema.org core schema.”, 2018. (accessed on 2018.03.19).

[SGK06] Leo Sauermann, Gunnar Aastrand Grimnes, Malte Kiesel, Christiaan Fluit, Heiko Maus, Dominik Heim, Danish Nadeem, Benjamin Horak, and Andreas Dengel. “Semantic Desktop 2.0: The Gnowsis Experience.” In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora M. Aroyo, editors, *The Semantic Web - ISWC 2006*, number 4273 in Lecture Notes in Computer Science, pp. 887–900. Springer Berlin Heidelberg, January 2006.

[She18] University Sheffield. “Google Takeout - IT services for students - CiCS - The University of Sheffield.”, 2018. Retrieved April 13, 2018 from `https://www.sheffield.ac.uk/cics/students/google-takeout`.

[SL12] Peter Swire and Yianni Lagos. “Why the right to data portability likely reduces consumer welfare: antitrust and privacy critique.” *Maryland Law Review*, **72**:335, 2012.

[SLC11] Amre Shakimov, Harold Lim, Ramón Cáceres, Landon P Cox, Kevin Li, Dongtao Liu, and Alexander Varshavsky. “Vis-a-vis: Privacy-preserving online social networking via virtual individual servers.” In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pp. 1–10. IEEE, 2011.

[Sne15] James M. Snell. “Activity Streams 2.0.”, 2015. (accessed on 2018.03.19).

[SSC09] Brandon Salmon, Steven W. Schlosser, Lorrie Faith Cranor, and Gregory R. Ganger. “Perspective: Semantic Data Management for the Home.” In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST ’09, pp. 167–182, Berkeley, CA, USA, 2009. USENIX Association.

[Swa13a] Melanie Swan. “The quantified self: Fundamental disruption in big data science and biological discovery.” *Big Data*, **1**(2):85–99, 2013.

[Swa13b] Melanie Swan. “The quantified self: Fundamental disruption in big data science and biological discovery.” *Big Data*, **1**(2):85–99, 2013.

[Swa13c] Melanie Swan. “The Quantified Self: Fundamental Disruption in Big Data Science and Biological Discovery.” *Big Data*, **1**(2):85–99, June 2013.

[tak] “Remember where you’ve been and what you’ve done with Your Timeline on iOS.” `https://www.blog.google/products/maps/remember-where-youve-been-and-what-youve-done-your-timeline-ios/`. Accessed: 2017-09-29.

[The14] The Apache Software Foundation. “Apache Jena.”, 2014. (accessed on 2018.03.19).

[Uni18] Indiana University. “Import and export Google data.”, 2018. Retrieved April 13, 2018 from `https://kb.iu.edu/d/aitl`.

[USD] USDA. “National Nutrient Database for Standard Reference.” (accessed on 2018.03.19).

[VK15] Konstantina Vemou and Maria Karyda. “Evaluating Privacy Practices in Web 2.0 Services.” volume 7, p. 7, 2015.

[VMK10] Max Van Kleek, Brennan Moore, David R Karger, Paul André, et al. “Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web.” In *Proceedings of the 19th international conference on World wide web*, pp. 951–960. ACM, 2010.

[ZZV06] Ying Zhao, Justin Zobel, and Phil Vines. “Using relative entropy for authorship attribution.” In *Information Retrieval Technology*, pp. 92–105. Springer, 2006.