THE UNIVERSITY OF CHICAGO

ARTICULATING AND ENFORCING DATAFLOW PREFERENCES

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
ZHIRU ZHU

CHICAGO, ILLINOIS
JUNE 2026

Dedicated to my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Raul Castro Fernandez, for his support and guidance throughout my PhD. I also thank my committee members, Prof. Marshini Chetty and Prof. Aloni Cohen, and collaborators for their time and feedback on my research.

I also extend my thanks to my previous advisors at New York University and Franklin & Marshall College for their early mentorship, and to my former colleagues at ZILLIZ.

Thank you to my friends and extended family for being there for me.

Finally, and most importantly, I want to thank my parents, Wenlian Wu (伍文莲) and Xiwu Zhu (竺锡武). None of this would have been possible without your unwavering love and constant belief in me. Thank you for everything.

ABSTRACT

The modern digital economy relies on continuous data exchange, but existing foundational mechanisms for data protection primarily govern data at rest. Once an authorized agent retrieves data, the semantic context of its intended purpose is lost, making this data-centric paradigm insufficient for articulating and addressing today's complex data problems.

This dissertation argues for a shift toward governing data in motion through a dataflow-centric paradigm. We introduce the dataflow preferences framework that enables agents to articulate enforceable preferences over dataflows, which are abstractions over the movement of data among agents.

To enforce these preferences within the application ecosystem, we propose a bolt-on data escrow architecture. Rather than pulling raw data to centralized platforms, this model leverages delegated computation, requiring applications to send their computational tasks to a trustworthy intermediary operating within the individual's trust zone.

For statistical computations, Differential Privacy (DP) serves as a rigorous algorithmic intervention to enforce preferences on the computation function of a dataflow. To address the practical difficulty data controllers face when choosing the privacy parameter ϵ in DP, we propose solutions to help controllers make more informed choices of ϵ, thus enabling practical deployment of DP intervention.

Together, the dataflow preferences framework and the proposed technical interventions provide the tool to articulate data problems and concrete mechanisms to incentivize, block, or modify dataflows, ensuring that agents' dataflow preferences are effectively enforced.

CHAPTER 1

INTRODUCTION

1.1 The Data-Centric Paradigm

The modern digital economy is powered by the continuous exchange of data. Individuals routinely provide personal data in return for services, and organizations increasingly pool data to extract collaborative insights. Data affects every aspect of our lives; however, we lack adequate tools to articulate the complex data problems that arise from this continuous exchange.

Historically, we have attempted to address these problems by exerting control over the data itself. Foundational mechanisms for data protection, such as access control, are designed to govern the initial release of data by evaluating an agent's credentials against static policies before granting access. This *data-centric* paradigm treats data as a static asset. It focuses on controlling the characteristics of the data—such as whether it contains Personally Identifiable Information (PII)—and gating the initial access to it.

However, the complex nature of today's data problems has fundamentally outpaced this approach. Once an authorized agent authenticates and retrieves the data, the semantic context of why the data was shared and the purpose for which it was used is lost. Because existing tools strictly control the data itself, they lack the vocabulary to assert preferences over the subsequent computation on the data or the downstream flow of any derived data. Consequently, data can be covertly copied, fed into proprietary machine learning models, or transmitted to unauthorized third parties without the originating agent's consent.

As a concrete example, consider the standard interaction between a smartphone user and a weather application. Under the data-centric paradigm, the mobile system relies on permission models to gate access. The system prompts the user to grant the application permission to read their location. If the user grants permission, the raw GPS coordinates

are handed over to the application. At this point, the user has no visibility into what the application does with the data next. The data might be used to fetch the local weather, or it might be covertly aggregated, tied to a device identifier, and sold to a data broker. Because control ends at the point of access, the user cannot know for certain that the data is used exclusively for its intended purpose.

Furthermore, the problems stemming from this loss of control extend far beyond traditional confidentiality and privacy concerns. For instance, an individual using a mobile application may wish to restrict resource-heavy computations to preserve battery life, or an organization may refuse to share data with a partner to maintain a competitive advantage. If we cannot articulate the specific nature of a data problem, we severely reduce our capacity to design effective tools to address it.

1.2 Shifting to a Dataflow-Centric Paradigm

To address this challenge, we must move away from governing data at rest toward governing data in motion. This requires a paradigm shift from a data-centric model to a *dataflow-centric* model. We need a mental model that allows us to reason not just about the data, but about its movement among agents and its effects.

A dataflow encapsulates not just what data is shared, but with whom it is shared, and for what purpose. By reframing data governance as a dataflow management problem, we elevate the purpose of data use to a first-class primitive. When the purpose of data use is made explicit, agents—whether they are individuals protecting their personal data, organizations managing sensitive datasets, or external parties such as auditors and policymakers—can articulate precise, enforceable preferences over their dataflows; these dataflow preferences inherently subsume traditional privacy preferences.

Returning to the weather application example, under a dataflow-centric paradigm, the user does not merely grant access to their location data. Instead, they specify their preference

over a specific dataflow. The user dictates that their GPS coordinates can flow to the application only for the purpose of determining the local weather forecast. Because the purpose of the data use is elevated to a first-class primitive within the system, the user's dataflow preference is strictly bound. If the application attempts to route the same location data to a third-party data broker, the dataflow violates the user's dataflow preferences and is blocked. The underlying mobile system must employ necessary interventions to enforce the user's dataflow preferences.

This dissertation posits that by providing a structured framework to articulate a dataflow and the specific preferences placed upon it, we establish the critical building blocks necessary to construct interventions that enforce those preferences. By decoupling an agent's preferences from the underlying enforcement mechanisms, we can systematically map complex data infrastructures to relevant dataflows within, and design targeted interventions that incentivize, block, or modify dataflows to align with agents' preferences.

1.3 Related Work

The development of the dataflow preferences framework is informed by decades of research across access control modeling, information flow theories, and privacy frameworks. Examining these foundational models reveals the specific conceptual gaps that necessitate a shift to a dataflow-centric paradigm.

1.3.1 Access Control and Usage Control

The foundational mechanism for data protection and security is access control [Graham and Denning, 1971, Lampson, 1974]. Various access control models have been proposed over the years, such as Mandatory Access Control (MAC) [Bell and La Padula, 1976, Denning, 1976, Sandhu, 2002], Discretionary Access Control (DAC) [Lampson, 1974, Sandhu and Samarati, 2002], Role-Based Access Control (RBAC) [Sandhu, 1998], Purpose Based Access

Control [Byun and Li, 2008], and Attribute-Based Access Control (ABAC) [Hu et al., 2015]. Access control is designed to govern the initial release of data by evaluating credentials of an agent against a set of static policies before granting access to the data. However, once an authorized agent authenticates and retrieves the data, the access control system cannot control what the authorized agent subsequently does with the data—how it is transformed, aggregated, or re-shared.

Usage Control (UCON) [Park and Sandhu, 2004] extends traditional access control models by evaluating usage conditions continuously. While UCON effectively extends the temporal reach of access control, it is typically scoped to continuous access sessions between a discrete subject and an object, and it is less equipped to govern scenarios where data flows among agents and may generate derived data products. Overall, these models control data access, but cannot control the downstream flow of data. The dataflow preferences framework fills this gap and enables agents to express preferences over the computation on the data after the initial point of access.

1.3.2 Information Flow Control

Information Flow Control (IFC) provides a rigorous approach to tracking data propagation within computing systems [Denning and Denning, 1977, Andrews and Reitman, 1980, Myers and Liskov, 1997]. It tracks how information flows through a program's memory and execution paths, ensuring that the handling of data complies with strict confidentiality and integrity policies [Nipkow et al., 2012]. The primary security theorem underpinning IFC is *noninterference* [Goguen and Meseguer, 1982], which guarantees that secret inputs do not observably affect public outputs. To implement IFC, systems such as Asbestos [Efstathopoulos et al., 2005], Flume [Krohn et al., 2007], and TaintDroid [Enck et al., 2014] introduced mechanisms to track the provenance and taint of data as it moves through the system, providing guarantees that data labeled with specific tags cannot flow to unauthorized endpoints.

While IFC provides strong guarantees against data leakage, it operates at a very low level of abstraction. It tracks the movement of data and ensures the data and its derived data does not leak into unauthorized channels, but lacks the semantic awareness to govern why and how the data is transformed. The dataflow preferences framework seeks to provide this missing high-level abstraction by explicitly modeling the computation over data, i.e., the *purpose* of its use, allowing agents to articulate preferences over the computation pipeline.

1.3.3 Contextual Integrity

The meaning of privacy has changed drastically over the past century [Solove, 2010]. The philosophy of Contextual Integrity (CI) [Nissenbaum, 2004] represents a paradigmatic shift, arguing that privacy is not about secrecy but about the appropriate flow of information according to context-specific norms. An information flow is governed by parameters including the data subject, data sender, data receiver, information type, and transmission principle.

Our dataflow preferences framework complements Contextual Integrity by extracting its core insight—that the flow of information is the fundamental unit of analysis, and we model the parameters of information flow in our dataflow definition. Crucially, it extends CI, specifically its transmission principle, by treating the *purpose* of data use as a first-class primitive, thus enabling concrete, enforceable preferences. Furthermore, we claim that preferences over dataflow *subsume* privacy preferences. The dataflow preferences framework aligns well with the philosophy of CI and enables agents to specify preferences on data problems related to privacy. However, agents also have non-privacy-related dataflow preferences; for example, a user using a phone application may not want certain resource-heavy computation to run to preserve phone battery, or a company may not want to share data with another company due to competitive advantages. The framework is agnostic to whether the data problem is privacy-related.

1.4 Technical Interventions to Enforce Dataflow Preferences

Articulating a dataflow preference is insufficient without the mechanisms to enforce it. Moving from theory to practice requires building systems that respect the explicit purpose of a dataflow. This dissertation explores two primary avenues of technical intervention to enforce dataflow preferences:

- **Data Escrows:** To enforce dataflow preferences over where and how computation occurs, we explore architectural interventions that invert the traditional model of data sharing. Instead of pulling raw data to centralized platforms, computation is delegated to a trustworthy data escrow. This intervention ensures that raw data remains protected while the computational purpose is explicitly bound and transparent.

- **Differential Privacy:** In scenarios where aggregate statistics must be computed and released, data controllers require formal guarantees that the result of the computation will not leak sensitive information. Differential Privacy (DP) [Dwork, 2006, Dwork et al., 2006] serves as a rigorous algorithmic intervention designed to enforce controllers' dataflow preferences over the computation on data. However, while theoretically robust, DP is difficult to use in practice because it is challenging to interpret the privacy implications of choosing the abstract privacy parameter ϵ. To make this intervention practically viable, we make contributions that help controllers choose ϵ and enforce their dataflow preferences.

1.5 Thesis Statement

This dissertation argues that articulating and addressing today's data problems requires a shift from a data-centric to a dataflow-centric paradigm. We establish the dataflow preferences framework as a tool for articulating data problems. Building upon this framework, we

design and deploy targeted technical interventions to enforce dataflow preferences. Specifically, we propose a data escrow architecture that enforces dataflow preferences via delegated computation. Furthermore, we introduce solutions that enable practical deployment of Differential Privacy (DP) interventions to enforce dataflow preferences in statistical computations. Together, these contributions provide concrete mechanisms to incentivize, block, or modify dataflows, ensuring that agents' dataflow preferences are effectively enforced.

1.6 Dissertation Contributions and Outline

This dissertation introduces the concept of dataflow preferences and utilizes this framework as a building block to construct concrete technical interventions across different layers of the data ecosystem. The remainder of the dissertation is organized as follows:

1. **Chapter 2: Dataflow Preferences.** We formally define the dataflow preferences framework. We model a dataflow by explicitly capturing the source agent, the target agent, the data, and the computational purpose for which the data is shared. We introduce a boolean predicate function to articulate preferences over the edge between the source and target agent, the data, and the computation function. To demonstrate how the dataflow preferences framework facilitates articulating data problems, we apply this framework to conduct a dataflow analysis of SafeInsights [SafeInsights Team, 2026b], a national-scale education research infrastructure. We map its agents, critical dataflows, and dataflow preferences placed by the agents, explain the interventions deployed to enforce their dataflow preferences, and identify challenges and opportunities.

2. **Chapter 3: Enabling Personal Dataflow Sovereignty via Bolt-on Data Escrow.** Chapter 3 proposes a bolt-on escrow architecture as a technical intervention to enforce dataflow preferences in the application (*app*) ecosystem. We introduce a minimally invasive programming interface, built on a unified relational interface that

virtualizes on-device data sources, that allows application developers to delegate computation to a trustworthy escrow. We present a concrete implementation within the Apple ecosystem, and we demonstrate that the model is expressive enough to handle dataflows in real-world applications while introducing minimal runtime overhead.

3. **Chapter 4: Within-Dataset Disclosure Risk for Differential Privacy.** Chapter 4 proposes solutions to make Differential Privacy (DP)—an intervention to enforce preferences on the computation function of a dataflow—more practical for data controllers. In a standard DP deployment, data controllers struggle to choose the privacy parameter ϵ because it is difficult to interpret the privacy implications of that choice on the within-dataset individuals. To bridge this gap, we derive a Relative Disclosure Risk Indicator (RDR) that indicates the impact of choosing ϵ on a within-dataset individual's disclosure risk relative to others. We present algorithms that help controllers choose ϵ based on their privacy preferences on the within-dataset individuals as a function of RDRs. We conduct a user study showing that RDR helps controllers make more informed choices of ϵ, and our proposed algorithms are efficient.

4. **Chapter 5: Orthogonal and Extended Systems for Articulating and Enforcing Dataflow Preferences.** Chapter 5 introduces other works to address the practical challenges of articulating and enforcing dataflows. First, we present *Ver* [Gong et al., 2023], a data discovery system that identifies project-join views over large, pathless repositories. *Ver* helps agents navigate semantic ambiguity to locate the specific data required to formulate a dataflow in the wild. Second, we introduce *Data Station* [Xia et al., 2022], a data escrow system designed to enable the formation of data-sharing consortia. Building on the delegated computation model, *Data Station* enforces organizations' dataflow preferences while providing transparency and audibility.

5. **Chapter 6: Conclusion.** Chapter 6 summarizes the core contributions of the disser-

tation and outlines future research directions in exploring tools to help agents articulate dataflow preferences and other interventions to enforce dataflow preferences.

CHAPTER 2

DATAFLOW PREFERENCES

2.1 Introduction

The modern digital ecosystems operate on the continuous exchange of data. Individuals routinely provide personal data in return for services, and organizations increasingly pool data to extract collaborative insights. Data affects every aspect of our lives; however, we lack adequate tools to articulate data problems. Existing tools often address data problems by exerting control over the data itself; specifically, they control the characteristics of the data (e.g., whether the data contains Personally Identifiable Information (PII)) and the access to it. However, the complex nature of today's data problems has fundamentally outpaced the *data-centric* paradigm we are familiar with.

A data-centric paradigm treats data as a static asset; once access is granted, the semantic context of why the data was shared and the purpose for which the data was used is lost. Existing tools that only control data itself lack the vocabulary to assert preferences over the *dataflow*, which is an abstraction of the movement of data and its derived data among agents. As a result, they lose control over the subsequent computation on the data or the downstream flow of the derived data. For instance, the data can be copied, merged with external datasets thus revealing sensitive information, fed into proprietary machine learning models, or transmitted to unauthorized third parties. Furthermore, many data problems extend far beyond traditional confidentiality and privacy concerns. Data's effects are multifaceted; it can be social, economic, and political. If we cannot articulate the data problem, we reduce our capacity to design effective tools to address it.

To address this challenge, we must move away from governing data at rest toward governing data in motion, by shifting from a data-centric paradigm to a *dataflow-centric* paradigm. We need a mental model that allows us to reason not just about the data, but about its

movement and effects. This chapter introduces the *Dataflow Preferences* Framework, a tool designed to articulate data problems. By providing a precise language to articulate a dataflow and the specific preferences placed upon it, this framework serves as the building block for constructing interventions that enforce those preferences.

The chapter first formally defines the dataflow preferences framework (Section 2.2). To demonstrate how this framework serves as a tool to articulate complex data problems, we conduct dataflow analysis of SafeInsights [SafeInsights Team, 2026b], a national-scale education research infrastructure (Section 2.3).

2.2 Defining Dataflow Preferences Framework

We formally introduce the dataflow preferences framework by defining the abstraction of data movement and the preferences agents place upon it. The framework provides a formal syntax designed to articulate data problems by decoupling the agents' preferences from the underlying interventions—whether they are technical, social-economic, or legal—to enforce those preferences.

Before defining dataflow and dataflow preferences, we must make clear what is *data* and who is the *agent*. We use definitions from a previous theory and systematization work that defines *data*, *document*, and *agent* [Castro Fernandez, 2025].

Definition 1 (Data)**.** Data is a representation of a state of nature.

Data represent a state of nature, which is a specific condition of an aspect of reality at a point in time—past, present, or future.

Definition 2 (Document)**.** A representation of data or not-data.

All representations of data are documents, some of which are carefully designed, such as PDF files and database tables, while others are not. The same data can be represented in different forms of documents.

Definition 3 (Agent)**.** An entity with the agency to represent, store, and process data.

An agent can be a human, a group, an organization, a bot, or other entities.

Definition 4 (Dataflow)**.** A dataflow ϕ is a tuple $< s, d, f(), t >$ that indicates how data d (in the form of a document) moves from a source agent s to a target agent t via an arbitrary function $f()$.

A dataflow is materialized when $f()$ is applied to d and the output of $f(d)$ flows to the target agent t. The function $f()$ limits the *purposes* for which the data d can be used. Different purposes can be encapsulated by the same function; for instance, an identity function $f()$ that copies the raw data d to t ($f(d) = d$) encapsulates all downstream purposes.

Definition 5 (Dataflow Preference)**.** A dataflow preference $\Psi_a(\phi) \rightarrow \{0, 1\}$ asserted by agent a on dataflow $\phi =< s, d, f(), t >$ is a boolean predicate function that maps ϕ to a binary validity state of either 0 (deny) or 1 (allow):

$$\Psi_a(\phi) = p_{(s,t)} \wedge p_d \wedge p_{f(d)},$$

where $p_{(s,t)} \rightarrow \{0, 1\}$ is a function that represents a's preference on the edge between source agent s and target agent t, $p_d \rightarrow \{0, 1\}$ is a function that represents a's preference on the characteristics of data d, and $p_{f(d)} \rightarrow \{0, 1\}$ is a function that represents a's preference on the function f applied to data d.

We first note that the agent a who is asserting their dataflow preference does not necessarily need to be s, or t, or the data subject of d. For instance, they can be a regulatory auditor, a parent managing a child's device, or a corporate compliance officer. When $\Psi_a(\phi)$ evaluates to 1, the dataflow aligns with agent a's preferences, and the dataflow should materialize. When it evaluates to 0, a *data problem* exists; the dataflow should not materialize or should be modified until the agent's preferences become valid. By decomposing components

of a dataflow and mapping them to distinct preference functions, the framework provides a simple yet powerful grammar to articulate data problems, and opens the door to build effective inventions to incentivize, block, or modify dataflows.

The *edge preference* $p_{(s,t)}$ governs the connection between the source agent s and the target agent t. For example, a company's IT department wants to prevent their employees' devices (s) from sending any data to a known malicious IP address (t); a specific intervention to enforce this preference is to add the IP address to the company's firewall. In practice, a variety of technical, socio-economic, and legal interventions are deployed to enable or block dataflows from one agent to another.

The *data preference* p_d evaluates the intrinsic properties, format, and content of the data itself. It limits the data to which $f()$ can be applied in the first place. For example, an agent might have a data preference that d must not contain PII, or that its size must not exceed a specific bandwidth threshold, or that the data structure must conform to a specific schema before transmission; based on the specific preferences, the corresponding interventions might be applying anonymization techniques to strip away PII, resizing and reformatting data.

The *function preference* $p_{f(d)}$ limits the *purpose* of the data use. For example, an agent might want to share their personal data with an e-commerce store for processing transactions, but they would not want the store to sell their data to a data broker. Potential interventions to restrict the purposes range from technical to social-legal contracts, such as Terms of Service. It is crucial to consider the granularity of a dataflow when articulating a data problem. Since the framework relies on $f()$ to limit purpose, when different purposes are indistinguishable at the function level, the framework cannot express them natively.

2.3 Apply Dataflow Preferences to SafeInsights

To demonstrate the efficacy of the dataflow preferences framework in articulating complex, real-world data problems, we apply the framework to the SafeInsights [SafeInsights Team,

2026b] ecosystem. SafeInsights (SI) is a national-scale education research and development infrastructure designed to connect student data from learning platforms across the country. Its goal is to enable researchers to conduct secondary research and post hoc analyses on large-scale sensitive student data while strictly protecting students' privacy, thereby helping researchers, educators, and policymakers understand how students learn across contexts and over time to drive policy and pedagogical innovation.

Before SafeInsights, education research has been limited to small samples of students in isolated moments because privacy concerns act as a barrier to connecting information across different platforms. SafeInsights overcomes this barrier by keeping student data securely within the learning platforms where it already resides [SafeInsights Team, 2026a]. Instead of moving raw data to a centralized repository for researchers to access, the infrastructure relies on secure data enclaves hosted by individual Data Organizations (DOs)—entities holding sensitive data for research, such as edtech companies, state and local education agencies, and educational institutions. Researchers formulate and submit their analysis code, which is then securely executed directly within the DO's enclave. This approach allows researchers to study large, diverse groups of students and test the efficacy of educational interventions on a nationwide scale without ever revealing the underlying raw student data.

Using the dataflow preferences Framework, the complexities of this infrastructure can be systematically mapped by identifying the relevant agents, the crucial dataflows, and the preferences placed over them by the agents. By applying the framework, we establish a common language that different stakeholders of SafeInsights can use to articulate data problems that may arise, and enable the formulation of interventions to address those data problems.

Figure 2.1 illustrates the dataflows (solid line) and interactions (dashed line) among the relevant agents we identified in the SafeInsights ecosystem. We loosely define interactions as any communication between two agents that does not involve the exchange of data (e.g.,

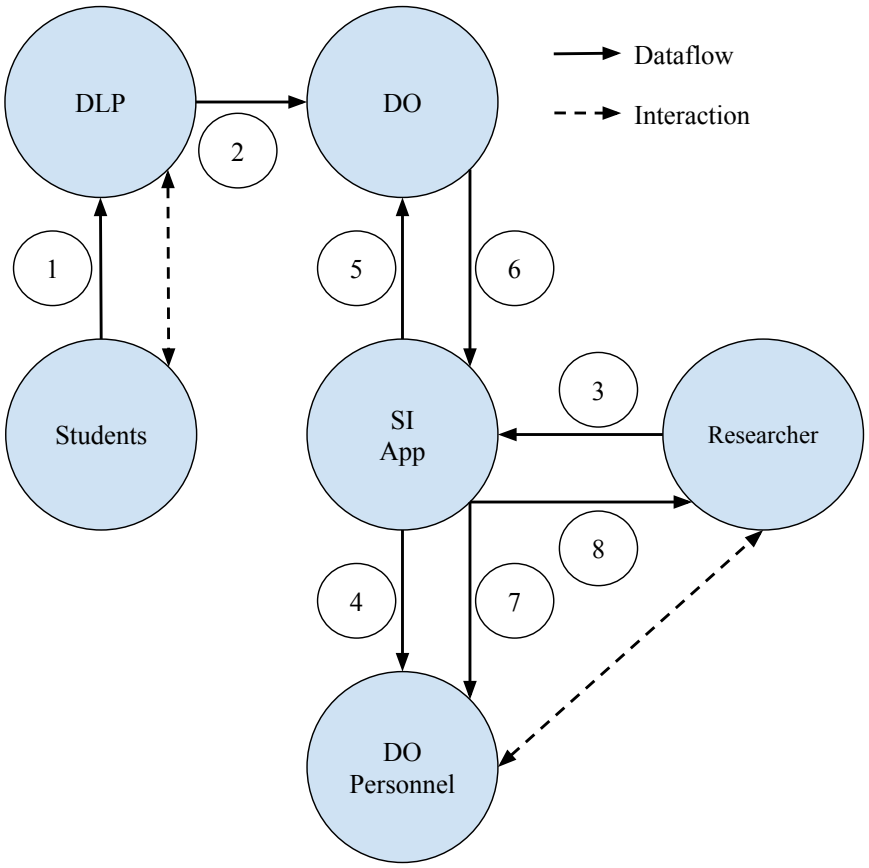

Figure 2.1: Dataflows within the SafeInsights ecosystem

the DLP showing an interface to students). We identify relevant agents of SafeInsights that enable crucial dataflows involving students' data and any derived data products of students' data. These include students, the researcher, Data Organization (DO), Digital Learning Platform (DLP), SafeInsights Application (SI App), and DO Personnel. We distinguish between DO, specifically the system components of DO, and DO Personnel as a human agent, because of the granularity of dataflows we define. In addition, although we do not include them in the figure as we only show dataflows that currently take place within SafeInsights, one agent that play a important role in specifying preferences over dataflows that take place within the SafeInsights ecosystem is the *auditors*, such as the Institutional Review Board (IRB), which is responsible for reviewing research involving human subjects. Potential dataflows involving the auditors may be introduced in the future as they are crucial for the

sustainability of SafeInsights.

We model the lifecycle of a research study in SafeInsights as a sequence of dataflows. At each step, we identify the agents' dataflow preferences.

1. In step 1, students (s) interact with the DLP, generating a dataflow in which $f()$ is a data copy that transmits students' data (d) consisting of their personally identifying information (PII) and data associated with the interactions (e.g., survey responses, grades, submitted homework) to the DLP (t). Students, DO Personnel, and auditors have edge preferences that dictate that the raw data d does not flow to any other agent besides the DLP. Crucially, they also have function preferences that dictate d cannot be used for purposes $f()$ that would reveal the students' PII and compromise their privacy. As a technical intervention to enforce edge preferences, the data is transmitted to the DLP hosted in a Virtual Private Cloud (VPC) to block dataflows to other agents. The entire SafeInsights infrastructure can be considered as an intervention to enforce the agents' dataflow preferences.

2. In step 2, the DLP (s) records students' data d into DO's storage (t), such as a database or a data warehouse. The function $f()$ applied to d can be a simple data transformation or a complex ETL pipeline. The DO may have a data preference over the specific format of d for it to be safely stored in its storage. In addition, to enforce edge preferences by other agents (e.g., DO Personnel, auditors), the DO's storage is hosted in VPC to block outside dataflows.

3. In step 3, the researcher (s) formulates their study and registers their research container (e.g., an R or Python program for their analysis) with the SafeInsights app (t), pending approval from the DO Personnel. In addition, the researcher interacts with the DO Personnel to iterate on their proposed study. In this particular dataflow, the research container is the data (d), and $f()$ is the registration process. The researcher may have

their own preferences about their research container.

4. In step 4, the SafeInsights app (s) sends the research container (d) to the DO Personnel (t) for review, thus $f()$ is the identity function.

5. In step 5, the SafeInsights app (s) sends the research container (d) to the DO (t); $f()$ is a data copy. This dataflow will not materialize if the DO Personnel's data preference on the research container is not satisfied. Their preference may include constraints such that the container is not corrupted and does not contain malicious code. The intervention is the manual review by the DO Personnel in step 4. One opportunity for more efficient interventions is to explore automated tools to assist DO Personnel in reviewing research containers.

6. In step 6, the DO (s) runs the research container ($f()$) on students' data (d) in its storage, and returns the output of the analysis $f(d)$ to the SafeInsights app (t). The DO Personnel and auditors have a function preference dictating that running research container code $f()$ on d does not leak sensitive information about d. One intervention to enforce this preference is through the manual review of the research container in step 4. Furthermore, they also have edge preferences that there are no dataflows from the DO (specifically the research container) to other agents besides the SafeInsights app. Several technical interventions have been deployed: the SafeInsights app is hosted in its own VPC, the research container is blocked from making connections outside the DO's VPC, and the output of the analysis is encrypted (by the Trusted Output App, a component operated by the DO) so that it is only accessible through the SafeInsights app. There exist opportunities to devise alternative interventions to enforce the preferences and evaluate their effectiveness.

7. In step 7, the SafeInsights app (s) sends the output of the analysis (d) to the DO Personnel (t) for review, thus $f()$ is the identity function.

8. In step 8, the SafeInsights app (s) releases the output of the analysis (d) to the researcher (t); $f()$ is also the identity function. This dataflow will not materialize if the DO Personnel's data preference on the output (i.e., it does not leak students' PII) is not satisfied. The intervention is the manual review by the DO Personnel in step 7. One future agenda is to explore interventions that can automatically identify whether the output leaks PII, reducing the burden of DO Personnel and improving the efficiency of the overall study pipeline.

By applying the dataflow preferences framework, we transition from a coarse-grained data-centric analysis to a dataflow-centric articulation of data problems within the SafeInsights ecosystem. When analyzing a system through a data-centric lens, the vocabulary restricts us to describing static data states and initial access permissions; once data is in motion, the analytical framework fails to capture the resulting aggregate output data or articulate its intended purpose. Consequently, it is difficult to design effective interventions to address the data problems because the specific nature of the problem, such as how the derived data flows and for what purpose, remains unarticulated.

In contrast, the dataflow analysis explicitly models the movement of data and its derived products. By decomposing the ecosystem into a sequence of dataflows and dataflow preferences placed by relevant agents, we establish a common language to articulate exactly what the data problems are. This explicit articulation serves as the necessary foundation to design and deploy targeted interventions.

Furthermore, we gain visibility into systemic constraints and opportunities for more efficient interventions. For example, currently, enforcing that the output does not leak PII relies heavily on manual review by DO personnel. By concretely articulating this bottleneck, we motivate the need for future technical interventions, such as automated PII auditing tools, that can enforce the same dataflow preferences while reducing the burden on human agents.

2.4 Conclusion

The dataflow preferences framework provides the mental model and formal syntax to articulate data problems by explicitly defining the preferences that agents place on distinct parameters of a dataflow. However, articulating a preference is insufficient without the mechanisms to enforce it. The remainder of this dissertation utilizes this framework as a building block to construct concrete technical interventions. Specifically, Chapter 3 proposes an escrow architecture to enforce dataflow preferences in the app ecosystem, and Chapter 4 proposes solutions to make differential privacy [Dwork, 2006, Dwork et al., 2006], a specific technical intervention on the function preference, more practical.

CHAPTER 3

ENABLING PERSONAL DATAFLOW SOVEREIGNTY VIA BOLT-ON DATA ESCROW

3.1 Introduction

Modern digital life runs on personal data. Every day, billions of individuals exchange sensitive data with platforms to enable services such as search, social media, health monitoring, entertainment, and access to LLMs. The prevailing model is simple: applications (*apps*) pull sensitive data (e.g., photos, contacts, location, health records) off individuals' devices and process the data on the platform's infrastructure. The moment a *dataflow* crosses the individual's *trust zone*—from the individual and their device to a remote platform—transparency and enforcement over *how* the data will be used are lost. While existing permission systems gate access to raw data, they fail to bind or expose the *purpose of its use*. This gap explains why seemingly benign requests (e.g., "share location for weather") can covertly support unrelated uses such as profiling or resale.

We frame this as a data management problem. Specifically, a *dataflow* management problem. We model a dataflow as a tuple $\langle A_s, D_s, f(), D_t, A_t \rangle$ that moves data D_s from a source agent A_s to a target agent A_t via computation $f()$ that produces D_t. This makes the *purpose* $f()$ explicit and enables *dataflow preferences* that subsume traditional privacy preferences: individuals want control not just over *what* data is shared, but *with whom and for what purpose*, as well as where and how that computation runs based on preferences over performance, battery, cost, and others.

Key idea: Bolt-on Data Escrow via Delegated Computation. We invert the prevailing model: instead of sending individuals' data to platforms, platforms *delegate* their computation to a trustworthy intermediary, a *data escrow* that resides within the individual's *trust zone*. App developers specify their computation using a minimally invasive programming

interface, RUN(ACCESS(), COMPUTE()). The ACCESS() function is built on top of a *unified relational interface* that virtualizes on-device data sources containing individuals' sensitive data, allowing developers to declaratively specify their data needs using standard SQL. The COMPUTE() function contains the concrete implementation of the computation $f()$. The escrow executes both functions and returns only the result D_t. Developers never gain access to the raw D_s, and every dataflow is explicit and auditable by construction. The same interface also supports *offloading* COMPUTE() across devices and escrow-controlled servers—allowing principled trade-offs between privacy, performance, battery, and cost.

Why the Escrow is Deployable. Modern app ecosystems already concentrate data access behind Software Development Kits (SDKs) [Koch et al., 2025, Zhang et al., 2024b, Rodriguez et al., 2025]. We prototype an escrow that is *bolted onto* Apple (e.g., iOS, macOS) SDKs, and describe a practical deployment path: Apple as the escrow provider can enforce all access to sensitive data to be routed through RUN(), making the escrow the non-bypassable bottleneck for data access. We design the escrow to be bolted onto existing SDKs so that the only parties that see the change are developers, who already constantly adapt to mandatory SDK changes. Unlike clean-slate solutions that require total ecosystem rewrites, our design integrates into the existing SDKs' update lifecycle.

Scope and limits. First, the main scope of our work focuses on establishing the *mechanism* (the escrow) required to enforce dataflow preferences. While navigating how individuals practically define and interact with these preferences (i.e., the *policy*) remains an open human-computer interaction challenge, addressing it requires a viable escrow architecture to exist first. Second, delegated computation fully supports dataflows involving a single individual's data. If a computation calls third-party APIs, our model also makes this dataflow transparent (what leaves the device and why). However, two classes of dataflows remain out of scope: computations that inherently combine many individuals' data and platform-confidential computation that cannot be disclosed. In these cases, the delegated model still

improves the status quo by forcing platforms to use the escrow to implement a data copy that is subject to individuals' explicit approval with visible purpose.

Evaluation. To validate our approach, we conduct a comprehensive evaluation that addresses three research questions regarding the escrow's feasibility, efficiency and design. First, we perform a qualitative analysis, porting the dataflows of 10 popular, open-source iOS applications (e.g., Wikipedia[Wikipedia, 2025], DuckDuckGo [DuckDuckGo, 2025]) to our escrow model to demonstrate whether the escrow's programming interface is expressive enough for real-world use. Our analysis shows that most dataflows can be effectively reimplemented using the escrow. Second, to demonstrate the practicality of the escrow, we conduct a quantitative evaluation by comparing the runtime of escrow-based app with a baseline app that uses native Apple SDKs directly. Our results show that the escrow introduces very little overhead, due to the efficiency of the escrow's relational engine implementation. Finally, we evaluate three different strategies for implementing the escrow's relational engine to identify the performance trade-offs. We found that by implementing classic database query optimization techniques such as predicate pushdown, one of our strategies can achieve superior performance while still preserving data freshness. Together, these experiments provide convincing evidence that our data escrow is not only expressive enough to capture the complex dataflows of real-world applications but also introduces negligible overhead, making it a practical and deployable solution for enabling individuals to gain transparency and control over their dataflows.

Contributions. This chapter makes the following contributions:

- A *dataflow model* that elevates the *purpose* ($f()$) and *trust zones*, enabling purpose-based, resource-aware dataflow preferences.
- A *data escrow architecture* centered on a programming interface, RUN(ACCESS(), COMPUTE()), built on top of a unified relational interface and a computation offloading component.

- An *escrow implementation* in the Apple ecosystem. We explain enforcement mechanisms to make the escrow's use mandatory, relational schema design principles, and three implementation strategies of the escrow's relational engine (Materialized Tables, Virtual Tables, and Virtual Tables with Pushdown). In addition, we provide detailed technical descriptions of the escrow's relational engine implementation using Virtual Tables with Pushdown and its end-to-end computation offloading pipeline.
- Both *qualitative and quantitative evaluation* showing that the escrow is expressive enough to implement dataflows from a wide range of real-world applications and that it introduces negligible performance overhead.

Outline. The rest of the chapter is organized as follows. Section 3.2 introduces the dataflow model to reason about personal dataflow sovereignty. Section 3.3 presents the data escrow architecture and its programming interface. Section 3.4 describes the design of our escrow in the Apple ecosystem. Section 3.5 details the technical aspects of our escrow implementation. Section 3.6 evaluates the escrow's expressiveness and performance. Section 3.7 discusses the related work and Section 3.8 concludes.

3.2 A Model for Personal Dataflow Sovereignty

To systematically address the problem of dataflow control, this section develops a model that allows us to reason precisely about the flow of data between different entities. We begin by defining the primary *agents* and the concept of a *trust zone* in the app ecosystem (Section 3.2.1). We then introduce the dataflow as the core unit of analysis (Section 3.2.2). Building on this, we define dataflow preferences as a richer form of control that subsumes traditional privacy preferences (Section 3.2.3). Finally, we use this model to articulate the fundamental transparency and control deficits in the current app ecosystem that our work aims to solve (Section 3.2.4).

3.2.1 Agents and Trust Zones

We define an *agent* as any entity with the ability to store and process data. There are many such agents in today's app ecosystem. We offer a distilled view of the ecosystem that highlights the agents that play a major role in the problem outlined in the introduction and the solution we present in this chapter.

- The **Individual** is the data subject, the person whose personal data is at the center of these interactions.
- The **Device** is the hardware owned and operated by the individual, such as a smartphone, wearable, or computer. It is the primary repository for much of the individual's sensitive data.
- The **Platform** is the entity providing a service, such as a social media company, a search engine, or a health application provider. Platforms are represented by the applications running on the device and the backend infrastructure they control.
- The **Developer** is the entity employed by the **Platform** to develop applications.

Central to our model is the concept of the *trust zone*. We define an individual's trust zone as the set of agents that the individual permits to access their raw data without restriction. In the prevailing model, this zone typically encompasses the individual and their personal devices. A dataflow is said to *cross the trust zone* when its target agent lies outside this boundary. For instance, when a user (Individual) interacts with a social media app such as Instagram on their iPhone, the iPhone (Device) is within the user's trust zone. However, Instagram is operated by Meta (Platform), which is outside the trust zone. When the app requests access to the user's location or photos and transmits that data to Meta's servers for processing, that dataflow crosses the trust boundary, moving out from a trusted domain. It is precisely these boundary-crossing dataflows that require more control.

Figure 3.1(a) provides a visual representation of today's app ecosystem, showing how an individual's data flows from their device, across the trust zone, to the platform's infrastruc-

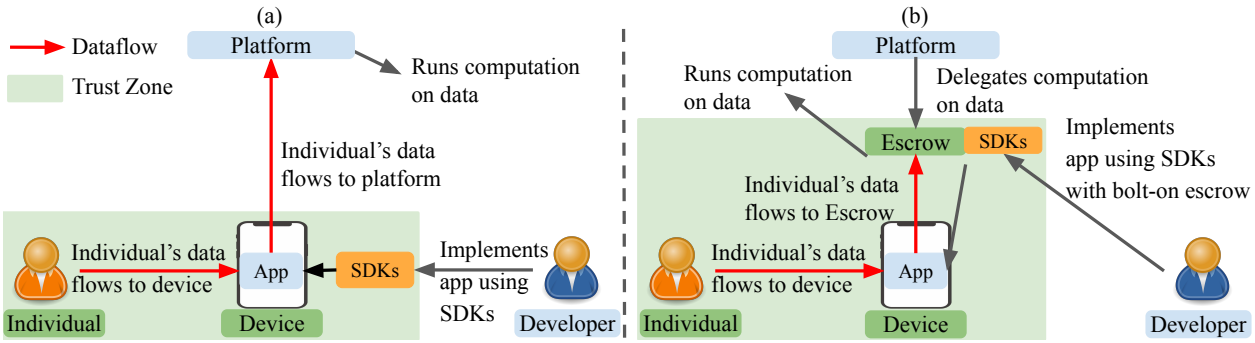

Figure 3.1: Dataflows in today's app ecosystem vs with the data escrow intermediary

ture where it is processed for purposes that are often opaque to the individual.

3.2.2 Defining and Characterizing Dataflows

Individuals share their data with platforms, inducing data to flow. We model such flow with a *dataflow*.

Definition 6 (Dataflow). *A dataflow is a tuple* $< A_s, D_s, f(), D_t, A_t >$*, that indicates how the data* D_s *from a source agent* A_s *is transformed via function* $f()$ *to produce data* D_t*, which is accessible to a different target agent* A_t*.*

A dataflow captures *what* data is shared (D_s), with whom (A_t), and for what purpose[1] ($f()$). Using this model, we can characterize various common operations. A simple data copy, such as uploading a contact list to a platform's server, is a dataflow where $f()$ is the identity function and thus $D_s = D_t$. A more complex operation, like a weather app using an individual's precise location (D_s) to retrieve the local forecast (D_t), involves a dataflow where $f()$ is the API call to the weather service. The key insight is that the function $f()$ explicitly defines the purpose of the data use.

We are primarily interested in dataflows that involve individuals' sensitive personal data such as contacts, health, photos, and location data. While this data is used for purposes

1. Technically, $f()$ limits the purposes for which the data can be used, rather than exactly determining the purpose, but the difference is not important for the technical presentation of this chapter.

many individuals consider positive, such as sharing content with friends on social media or receiving localized weather forecasts, it has also been at the center of widespread privacy violations over the past two decades. Numerous high-profile incidents have revealed how platforms repurpose this data for undisclosed purposes: selling it to data brokers [Crain, 2018], enabling targeted surveillance [Collier and Burke, 2022], training AI models without consent [Yang, 2024], or enabling discriminatory practices in housing and employment [Spinks, 2019]. The opacity of these dataflows, where individuals cannot verify that their location shared for weather updates isn't also sold to advertisers, or that their health data isn't used to adjust insurance premiums, represents a critical failure of the current ecosystem to respect individual preferences and societal values around data use.

3.2.3 Dataflow Preferences

Individuals have preferences on how their data should be used, which go beyond privacy preferences. By making the purpose $f()$ an explicit part of our model, we define a richer and more powerful form of individual preferences that *subsumes* traditional privacy preferences. Standard privacy preferences as presented by apps today are often binary, revolving around access to data (e.g., "Allow access to Location?") [Almuhimedi et al., 2015, Tan et al., 2014, Felt et al., 2012]. *Dataflow preferences*, in contrast, enable nuanced, purpose-based control over individuals' data.

We argue that an individual's privacy is preserved only when individuals' dataflow preferences are preserved. An individual may be perfectly willing to permit a dataflow where their location data (D_s) is used by a weather app (A_t) for the purpose of fetching a forecast ($f()$). However, they may wish to deny a different dataflow where the same location data is sent to an advertising broker (A_t) for the purpose of profiling ($f()$). This aligns with the privacy framework of Contextual Integrity [Nissenbaum, 2004], which states that adequate privacy protection can only be offered when information norms of a specific context are respected.

Our dataflow model provides the formal structure needed to define and enforce these norms. It allows individuals to express preferences not just on what data is shared, but for what purpose, which is a more accurate reflection of real-world privacy expectations.

Furthermore, dataflow preferences extend beyond traditional privacy preferences. For instance, an individual might specify a preference to only allow computationally intensive dataflows (e.g., video processing) to run when their device is connected to Wi-Fi and charging, in order to conserve battery life and mobile data. Another individual might prefer that certain computations are offloaded to a server for faster performance. These dataflow preferences concerning resource consumption, performance, and cost are not related to privacy in the classic sense, but they are critical aspects of controlling one's data use. By capturing a broad set of dataflow preferences, our model allows us to reason about *personal dataflow sovereignty*—individuals' fundamental right to control their dataflows is the right to enforce their dataflow preferences.

3.2.4 The Transparency and Control Deficits

The dataflow model allows us to articulate two key challenges of the current app ecosystem that result in the loss of *personal dataflow sovereignty*:

- **Lack of Transparency**: Individuals do not have clear visibility on what dataflows take place when using an app. When an app requests their data, the purpose $f()$ is at best implied and at worst intentionally obscured [Almuhimedi et al., 2015, Zang et al., 2015, Claesson and Bjørstad, 2020, Xiao et al., 2023, Rodriguez et al., 2024]. The terms of service that supposedly govern data use are notoriously broad and vague, failing to specify the concrete computations that will be performed on the data [Obar and Oeldorf-Hirsch, 2020].
- **Lack of Control**: Even if the purpose $f()$ were made transparent, individuals currently lack the technical mechanisms to enforce their preferences on the dataflows. The only

available control is a blunt instrument: denying access to the source data D_s entirely. This often forces an undesirable trade-off, as denying access may break the primary, desired functionality of the app, leaving the individual with an all-or-nothing choice.

The goal of this chapter is to design and implement the technical *mechanism* that endows individuals with both *transparency* and *control* over their dataflows, providing the necessary foundation for future *policy* design on eliciting individuals' dataflow preferences.

3.3 The Data Escrow Architecture

To address the transparency and control deficits identified in the current app ecosystem, this section details the design of our proposed *data escrow* architecture. We first introduce our key idea of employing the delegated computation model (Section 3.3.1). We then present a minimally invasive programming interface to enable delegated computation (Section 3.3.2). Next, we introduce computation offloading as a core architectural feature that enables moving computation across devices (Section 3.3.3). We then describe the data virtualization layer that provides a unified relational interface for transparent data access (Section 3.3.4). Finally, we provide a taxonomy of common dataflow patterns to clarify the capabilities and limitations of our solution (Section 3.3.5).

3.3.1 Key Idea: Delegated Computation Model

The key idea of our solution is to replace the traditional model of sending individuals' data to platform, illustrated in Figure 3.1(a), with a *delegated computation* model. This architectural shift is visualized in Figure 3.1(b). Instead of the platform pulling an individual's data (D_s) from their device to run computation ($f()$) on its own opaque infrastructure, which crosses the individual's trust zone, the platform must delegate its computation to a trustworthy *data escrow* [Xia et al., 2022] intermediary (i.e., it must be in the individual's trust zone). This inversion of control is made possible because modern applications are not built from scratch;

they are built using *Software Development Kits (SDKs)* provided by the ecosystem owner (e.g., Apple or Android). These SDKs provide the standard tools and APIs for all essential functions, including access to sensitive personal data. Our key insight is that an escrow can be *bolted onto* these existing SDKs. Developers, who already rely on these SDKs to write apps, would now use an escrow-based programming interface to specify their computation.

As shown in Figure 3.1(b), the developer implements their app against this new programming interface, effectively delegating their intended computation, $f()$, to the escrow. The escrow, operating as a trustworthy intermediary, then pulls the required data (D_s) from the device and executes the delegated computation on behalf of the platform. This execution only proceeds if the individual approves the entire dataflow. Because the escrow mediates all access to sensitive data, it becomes a natural *bottleneck*, making every dataflow inherently transparent and controllable by design. The fundamental trust assumption shifts: the individual no longer needs to trust every application platform but instead places their trust in the escrow provider.

3.3.2 The RUN(ACCESS(), COMPUTE()) *Interface*

To enable delegated computation in a developer-friendly manner, we design and implement a minimally invasive programming interface centered on a higher-order function: RUN(ACCESS(), COMPUTE()). This function is designed to be bolted onto existing SDKs of the current app ecosystem (e.g., Apple or Android SDKs), thus lowering the barrier to adoption. The function has two main parameters:

- ACCESS(): The *data access function* that specifies the input data, D_s, required for the subsequent computation $f()$. As detailed in the next section, developers implement this function declaratively by writing a standard SQL query.
- COMPUTE(): The *computation function* that contains the concrete implementation of $f()$. It takes the output of ACCESS() as its input, and it can use any libraries or call third-party

APIs.

The RUN() function is the core entry point executed by the escrow. When an application calls RUN(), the escrow orchestrates the entire dataflow. It first executes the ACCESS() function to retrieve D_s. It then invokes the COMPUTE() function, passing D_s as the input, to produce the result D_t. The critical guarantee of this interface is that the platform and its developers never gain access to the raw data D_s. They provide the code for ACCESS() and COMPUTE(), and they receive the final output D_t, but D_s is only ever handled by the escrow within the individual's trust zone. This cleanly separates the specification of computation from the access to raw data.

Delegated Computation in Action. To make the delegated computation model concrete, consider a simple weather app. The app's goal is to use the individual's current location (D_s) to fetch the local weather forecast (D_t). Without the escrow, a developer would use the native SDK to directly fetch the device's location coordinates, which are considered sensitive, then send those coordinates across the individual's trust zone to the platform's server. A simplified code snippet (in Apple's Swift language) might look like this:

```
// Use native SDK to fetch raw location coordinates
let location = LocationManager.getCurrentLocation()
// Send raw data across the trust zone to the platform
let weather = PlatformAPI.fetchWeather(for: location)
```

In this common pattern, the raw location coordinates leave the individual's device. The individual has no technical guarantee that the platform is only using the data to fetch the weather and not for other purposes, such as selling it to data brokers.

With the escrow, the developer implements the weather app using the RUN() interface.

```
import Escrow
// Declaratively specify the data needed
```

```
let locationSQL = "SELECT cllocation FROM Location
                   ORDER BY timestamp DESC LIMIT 1"
// Define the computation (purpose)
func getWeather(location: CLLocation) -> Weather {
    // The computation only sends an approximate region,
    // not the precise coordinates
    let region = getSurroundingRegion(from: location)
    return PlatformAPI.fetchWeather(for: region) }
// Delegate computation to escrow
let weather = Escrow.run(locationSQL, getWeather)
```

In this revised implementation, the developer never handles the raw location data. The SQL query and the `getWeather` function are submitted to the escrow. The escrow executes the query, passes the resulting `CLLocation` object to the `getWeather` function, sends the approximate region to the platform to fetch weather, and returns only the final `Weather` object to the app. The entire operation is transparent to the individual, who can verify the purpose and approve (or deny) this dataflow, with the understanding that their raw location coordinates never leave their trust zone. Alternatively, if the platform needs the raw location coordinates instead of the approximate region to fetch weather, the developer can still implement the same logic as before, which sends the raw coordinates to the platform; this dataflow will be transparent to the individual via the escrow and they can control it. In contrast, without the escrow, once the user grants permission for the app to access their location, they lose control over their dataflows.

Note that the specific interface that the escrow uses to communicate the content of RUN() and elicit preferences from the individual is beyond the scope of our chapter; it is a complementary *policy* design challenge that relies entirely on the underlying technical *mechanism* presented in our work. One simple strategy, for example, is to summarize the

content of RUN() in the permission pop-ups shown to the individual.

3.3.3 Computation Offloading

The delegated computation model naturally extends to support flexible execution locations. While many computation can run efficiently on-device, some may be too resource-intensive or require access to server-side resources. The practice of computation offloading—transferring intensive tasks to an external location—is a well-established technique to overcome device limitations such as computational power, storage, and energy [Kumar et al., 2013]. Our escrow architecture allows COMPUTE() functions to be offloaded to a trustworthy, *escrow-controlled* server. This is achieved by extending the RUN() function with an optional SERVER_ID parameter: RUN(ACCESS(), COMPUTE(), SERVER_ID).

When SERVER_ID is provided in RUN(), the on-device escrow first executes the ACCESS() function to obtain the input data D_s. It then serializes D_s and transmits it, along with the signature of the COMPUTE() function to be executed, to the specified remote server via a secure channel. The remote server, which is managed by the escrow provider and thus remains within the individual's trust zone, executes the computation and returns the serialized result D_t. This offloading mechanism allows developers to balance on-device resources against the power of server-side processing. It aligns well with emerging industry trends such as Apple's Private Cloud Compute (PCC) [Apple, 2024], which provides a secure server infrastructure designed for privacy-preserving computation offloaded from mobile devices. Our escrow architecture provides a general framework that could be directly implemented on top of such infrastructure, leveraging its strong, hardware-enforced security guarantees.

3.3.4 Data Virtualization via Relational Interface

A key design decision is how developers should implement the ACCESS() function. Simply allowing developers to write arbitrary code would make the data access function opaque,

effectively hiding a secondary computation within the ACCESS() function itself. To ensure transparency, we design a *declarative* interface based on the classic data virtualization approach [Carey et al., 1995].

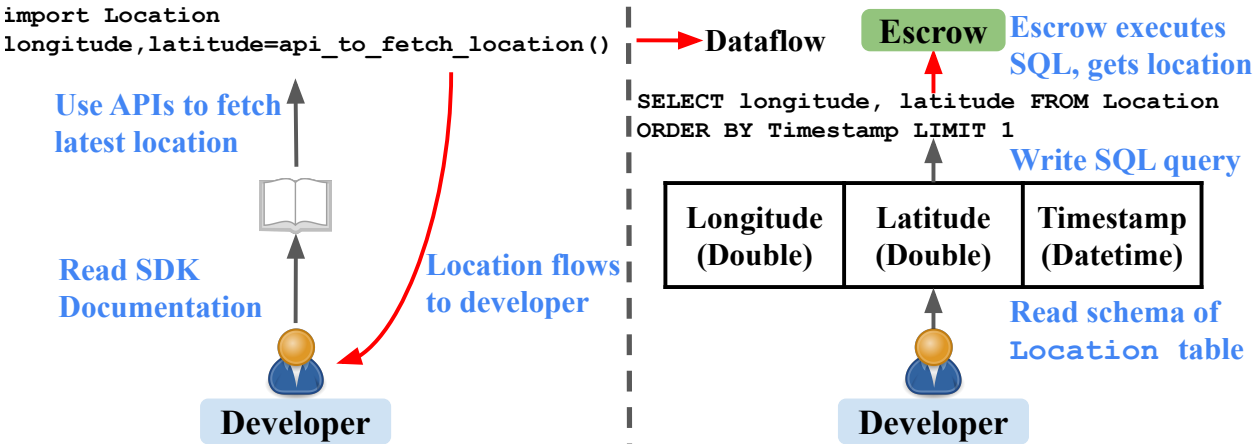

Figure 3.2: Accessing location using native SDKs (left) vs using the escrow's relational interface (right).

The escrow virtualizes the device's heterogeneous data sources, such as contacts, photos, and location updates, into a unified relational schema. Instead of using numerous native SDK APIs to access data, developers now implement ACCESS() by writing a standard SQL query against this public schema. For example, to fetch an individual's most recent location, a developer would write a SQL query rather than importing the `Location` framework and writing imperative code to request location updates. Figure 3.2 contrasts these two approaches. The traditional approach (left) requires developers to learn SDK-specific APIs and gives them direct access to the resulting location. The escrow's relational interface (right) simplifies the data access to a declarative query and ensures the developer never handles the raw location data directly.

3.3.5 Taxonomy of Dataflow Patterns

The delegated computation model is designed to give individuals control over a wide range of dataflows, but its applicability varies depending on the nature of the computation. To clarify

the scope of the escrow, we provide a principled categorization of four common dataflow patterns in Table 3.1.

Pattern	Within individual's trust zone	Computation with external API call	Cross-individual computation	Confidential computation
Description	Computation runs fully within trust zone using a single individual's data	Computation calls third-party or platform API, sending individual's data outside the trust zone	Computation combines data from multiple individuals to produce aggregate results	Platform cannot or will not reveal computation
Example	An app scans individual's photo library to tag pictures of their pets using image classification model hosted on-device or in an escrow-controlled server	A weather app gets individual's location, calculates an approximate region based on the location, and sends the region to a third-party weather API to fetch forecast	A mapping service using location data from multiple individuals to calculate real-time traffic conditions	A social media app using a proprietary algorithm to recommend friends from individual's contacts
Supported by escrow	Fully supported	Supported with transparency - The escrow makes data transfer transparent	Unsupported - Can only be implemented by copying each individual's data to platform	Unsupported - Can only be implemented by copying each individual's data to platform

Table 3.1: Taxonomy of dataflow patterns and whether they are supported by the escrow's delegated computation model

The escrow fully supports computations that are self-contained within an individual's trust zone, which includes computation on-device or in escrow-controlled servers through computation offloading. This category covers a wide range of common and important dataflows in today's apps. The delegated model also supports computation that require calling external APIs, where the escrow's role is to make the data transfer transparent; in the example, individuals can see that the computation function sends the approximate region to an external weather API, not their raw GPS coordinates, and they may decide whether to allow this dataflow. However, the delegated model is not suited for patterns that are fundamentally cross-individual or confidential. For instance, computing real-time traffic conditions requires combining location data from many individuals, and platforms may be unable to delegate proprietary algorithms that are trade secrets. In these unsupported cases, the only option is for the platform to use the escrow to implement a simple data copy [2]. This still represents a significant improvement over the status quo: the data copy becomes transparent and subject to the individual's explicit approval, empowering them with a meaningful choice that is often absent in today's ecosystem.

2. Or via emerging (but not yet existing) platforms, such as data cooperative, data commons, and data trust [researc, 2020].

3.4 Escrow in Apple Ecosystem

To demonstrate the practicality of our proposed escrow architecture, we design and implement a data escrow within the Apple ecosystem. We begin by explaining our rationale for choosing the Apple ecosystem and how it provides a natural bottleneck for data access (Section 3.4.1). We then describe our approach to system integration to make the escrow's use mandatory (Section 3.4.2). Finally, we describe our schema design principles and three implementation strategies of the escrow's relational engine (Section 3.4.3).

3.4.1 Exploiting Apple SDKs as Bottleneck

Mobile devices are the primary repositories of sensitive personal data, and the two dominant ecosystems are Apple's iOS and Google's Android [Zimperium, 2025]. While our escrow architecture could be adapted to either, we choose to implement our prototype in the Apple ecosystem, with Apple acting as the trustworthy escrow, as its vertically integrated nature offers a clear path to deployment. A successful escrow must become the non-bypassable *bottleneck* to mediate all sensitive data access. The Apple ecosystem facilitates the creation of such a bottleneck for three key reasons: **First**, to access sensitive personal data such as health records, photos, contacts, or location, developers are required to use specific Apple SDKs (e.g., `HealthKit` [Apple, 2025i], `PhotoKit` [Apple, 2025f], `Contacts` [Apple, 2025g], `CoreLocation` [Apple, 2025h]). There are no alternative methods to access those data. **Second**, all apps distributed through Apple's App Store must undergo a rigorous review process where Apple can enforce its policies [Apple, 2025a]. This provides a centralized point of control to ensure that all data access in the app must go through the escrow. **Third**, individuals generally exhibit a high degree of trust in their Apple devices and, by extension, in Apple as the platform vendor. This pre-existing trust makes the platform vendor a natural and suitable candidate to assume the role of the data escrow, as it would seamlessly fit within the individual's existing trust zone. Combining these factors makes it

feasible to create an escrow that can be *bolted onto* the existing Apple infrastructure.

3.4.2 Escrow Integration and Enforcement

We implement our escrow prototype as a third-party library that developers would include as a dependency. In this model, the escrow is instantiated as a singleton object within the app's process space, and developers interact with it via the RUN() function.

While our prototype is a third-party library, for the escrow model to be effective, its use must be *mandatory*. We describe a practical enforcement mechanism that a platform vendor like Apple could implement. Data access in iOS is governed by declarations in the app's *Entitlements* [Apple, 2025b] and *Information Property List* [Apple, 2025c] files. Apple could enforce the escrow model by modifying its policies for these configurations. It could block all direct access to sensitive SDKs by rejecting any app that declares usage of, for instance, `HealthKit` in its entitlements but does not route all `HealthKit`-related operations through the mandatory escrow library's RUN() function. This would make the escrow the only entry point for accessing sensitive data, effectively forcing adoption of the delegated computation model and ensuring that no dataflows can bypass the individual's control.

3.4.3 Relational Interface Design

The core component of our escrow implementation is the data virtualization layer that provides the relational interface for the ACCESS() function. This component is responsible for virtualizing on-device data sources and executing SQL queries against them.

Designing the Relational Schema

The escrow exposes a public relational schema that developers use to formulate their data access queries. The design of this schema is guided by the principle of minimizing friction for developers who are already familiar with the native SDKs. Rather than creating an entirely

abstract schema, our tables and columns map closely to the objects and properties provided by Apple's SDKs [3]. For example, Apple's `PhotoKit` [Apple, 2025f] framework represents a photo or video asset as a `PHAsset` [Apple, 2025e] object. Our corresponding `Photos` table in the relational schema includes columns for standard attributes such as `id` (`String`), `mediaType` (`Integer`), and `creationDate` (`Date`). Crucially, it also includes a `phasset` column whose type is the native `PHAsset` object itself. This allows a developer to write a SQL query to filter and select assets declaratively, and then receive a sequence of native `PHAsset` objects in the COMPUTE() function, on which they can perform operations using the standard `PhotoKit` APIs they already know. This design choice makes the transition to the escrow model more intuitive and less disruptive for developers.

Designing the Relational Engine

The relational interface provides a logical view of the data to developers, and they no longer need to implement the concrete mechanisms to retrieve the data—this responsibility falls onto the escrow. We design, implement, and evaluate three approaches to implement the escrow's relational engine, each with different performance trade-offs.

Materialized Tables. In this baseline approach, the escrow upon initialization fetches all relevant data declared by the app's ACCESS() functions using the native Apple SDKs, and materializes it into tables within an on-device SQLite database. The ACCESS() function is then executed as a standard SQL query against this database. While this approach benefits from the performance of a native database engine, it suffers from two significant drawbacks. First, it creates a data freshness problem, as the materialized data can become stale if the underlying source changes. Second, it introduces significant overhead for complex data objects like `PHAsset`, which cannot be stored directly in SQLite. Instead, we must store a

3. Nevertheless, the *object–relational impedance mismatch* [Ireland et al., 2009] always exists. Our goal is not to solve this mismatch, but to design the schema in a developer-friendly manner.

reference ID, and the escrow must perform a costly post-query lookup to map these IDs back to their corresponding native objects.

Virtual Tables. To address the issues of data freshness and object handling, our second approach leverages SQLite's virtual table mechanism [SQLite, 2025a]. A virtual table is a schema object that looks like a normal table but does not store any data in physical storage. Instead, when the database engine needs to scan a virtual table, it makes callbacks to custom C functions provided by our implementation. We implement these callbacks to make live API calls to the relevant Apple SDKs (e.g., `PhotoKit`, `Contacts`) to fetch all relevant data needed to populate the table. This data is then passed back to the SQLite engine, which performs any necessary filtering, sorting, or aggregation in memory. This approach ensures that query results are always up-to-date and avoids data duplication. Furthermore, because the data is only held in memory during query execution, we can pass pointers to complex native Swift objects (like `PHAsset`) through the C-to-Swift bridge, allowing the COMPUTE() function to receive them in their native format without any serialization overhead. However, this approach can be inefficient for selective queries, as it requires fetching all data from an API before filtering.

Virtual Tables with Pushdown. Our third approach enhances the virtual table implementation by applying classic database query optimization principles. The goal of predicate pushdown [Hellerstein and Stonebraker, 1993] is to filter data as early as possible to minimize data processing. We implement logic to push down query operations such as projections, predicates, sorting, and limits, from the SQLite query engine into the underlying native SDK API calls, since many of Apple's SDKs provide APIs that allow for more efficient, constrained data fetching. For example, consider the query `SELECT phoneNumber FROM Contacts WHERE givenName == 'Alice'`. A naive virtual table implementation would fetch *all* contact records from the device and let SQLite perform both the filtering on `givenName` and projection on `phoneNumber`. Our pushdown implementation, however, will translate this

query into an efficient `CNContactFetchRequest` [Apple, 2025d] with a predicate for the name 'Alice' and `keysToFetch` set to only the `phoneNumber`. This pushes down the filtering and projection operations from the SQLite engine into the native API, reducing the amount of data that needs to be fetched from the underlying data source and processed by the SQLite engine, leading to significant performance gains in many cases, as shown in our evaluation.

3.5 Escrow Implementation

This section details the technical implementation of the escrow's two core subcomponents: the relational engine that powers the on-device data virtualization layer, and the secure offloading pipeline that enables delegated computation on escrow-controlled servers. We describe the detailed techniques we employ to implement the escrow's relational engine using Virtual Tables with Pushdown (Section 3.5.1) and its computation offloading pipeline (Section 3.5.2).

3.5.1 Relational Engine Internals

The core of our prototype is a relational engine that bridges the declarative world of SQL with the imperative, object-oriented APIs of Apple's native SDKs. We provide our implementation details of the Virtual Tables with Pushdown approach, as described in Section 3.4.3. The virtual table implementation for each virtualized data source follows a consistent pattern: a thin C layer interfaces directly with SQLite, while a Swift backend contains the primary logic for data fetching and pushdown optimization.

The C-to-Swift Virtual Table Bridge

We leverage SQLite's virtual table module [SQLite, 2025a] to build the relational engine. For each data source (e.g., Contacts, Photos, Location), we implement a standard `sqlite3_module`

[SQLite, 2025c], a C structure that registers a set of callbacks with the SQLite engine. These C callbacks act as a lightweight bridge, delegating the main work to Swift functions exposed to the C runtime via the `@_cdecl` attribute. This hybrid architecture allows us to combine the low-level control of SQLite's C API with the high-level power of Swift for interacting with modern Apple SDKs. The query lifecycle is managed by these key C callbacks:

- `xConnect`: Invoked by SQLite when creating the virtual table. We use them to declare the virtual table's schema (e.g., the columns `givenName`, `familyName`, `phoneNumber` for the `Contacts` table).
- `xBestIndex`: This callback is the heart of our optimization logic. It is called by the SQLite query planner, which passes a representation of the query's constraints (i.e., the `sqlite3_index_info` structure [SQLite, 2025b]), including information such as WHERE and LIKE clause predicates, ORDER BY terms, and LIMIT counts. Our `xBestIndex` implementation parses these constraints to determine which operations can be pushed down to native Apple SDKs, and encodes the optimized indexing strategy into an integer bitmask (for predicates) and a formatted string (for projection mask and other parameters) by filling in the `idxNum` and `idxStr` fields of the `sqlite3_index_info` structure, which forms the reply to the SQLite query planner.
- `xFilter`: This function executes the data fetch according to the indexing strategy selected in `xBestIndex`. Its primary role is to bridge into our Swift implementation to perform the data fetch, passing the predicate values and other details of the indexing strategy; the Swift logic is responsible for translating the encoded indexing strategy into an efficient data fetch request using native Apple SDKs. `xFilter` receives back an opaque pointer to a Swift object that contains an in-memory snapshot of the query results.
- `xNext`/`xColumn`: After `xFilter` fetches the result snapshot, SQLite uses this pair of functions to iterate through the result set. `xNext` simply increments a row counter, while `xColumn` calls back into our Swift implementation to retrieve the data for a specific row

and column from the snapshot.

Zero-Copy Native Object Handling

A key challenge is handling complex native Swift objects (e.g., `PHAsset` or `CLLocation`). A naive approach, such as the one used in our Materialized Tables baseline, would require storing object identifiers in the database and performing costly lookups to reconstruct the native objects post-query. Our virtual table implementation avoids this overhead. When the `xColumn` callback iterates through query results, it obtains an opaque pointer to the Swift object in memory (via `Unmanaged.passRetained(object).toOpaque()`). This pointer is passed across the C bridge and what SQLite stores for the object-typed column. Later, when the escrow processes the query results from SQLite, this pointer is passed back from the C runtime to Swift and safely converted back into a fully-typed Swift object (via `Unmanaged.fromOpaque(pointer).takeRetainedValue()`). This zero-copy mechanism allows passing native objects through the relational engine without any serialization or expensive re-fetching, which is critical for the performance gains shown in our evaluation.

3.5.2 Computation Offloading Pipeline

The escrow architecture supports offloading computation to escrow-controlled servers. This is enabled by an optional SERVER_ID parameter in the RUN function call.

Architecture and Security Model

The offloading pipeline runs on a client-server architecture where the on-device escrow acts as the client and the server is designed to run in a Trusted Execution Environment (TEE) [Sabt et al., 2015], analogous to systems like Apple's Private Cloud Compute [Apple, 2024]. Communication between client and server is encrypted using mutual TLS (mTLS), which authenticates both the device and server to protect data in transit. Furthermore, code integrity is

enforced by requiring developers to pre-register and cryptographically sign any COMPUTE() function intended for offloading; the server verifies this signature against its registry before execution to prevent unauthorized code from running. These measures provide the technical foundation for securely extending the individual's trust zone to include remote infrastructure.

End-to-End Offloading Pipeline

When an app calls RUN(ACCESS(), COMPUTE(), SERVER_ID), it executes the following steps:

1. Local Data Access. The ACCESS() always executes locally on the device via the relational engine to obtain the source data, D_s. This ensures raw data access remains under the local escrow's control.

2. Serialization. The resulting Swift objects are serialized into a portable binary format (e.g., Protocol Buffers [Google, 2025]).

3. Secure Invocation. The on-device escrow establishes an mTLS connection to the server specified by SERVER_ID. It then transmits a request payload containing the serialized D_s and a unique identifier for the pre-registered COMPUTE() function.

4. Remote Execution. The server authenticates the device's TLS certificate and verifies the integrity of the requested function. Upon success, it deserializes D_s and invokes the corresponding COMPUTE() function within a sandboxed process, passing the data as input.

5. Result Return. The final result, D_t, is serialized by the server and sent back to the device over the secure mTLS channel. The on-device escrow deserializes the result and returns it to the app.

This pipeline provides the technical foundation for enforcing resource-aware dataflow preferences, and enables the escrow to act as a scheduler, dynamically placing computation based on individuals' preferences, device state (e.g., battery, network), and the computational cost of the dataflow.

3.6 Evaluation

In this section, we answer three research questions:

- **RQ1**: Can real-world applications be ported to the escrow? (i.e., is the proposed escrow solution feasible?) We collect 10 representative apps and study their implementation on the escrow-based programming interface (Section 3.6.1).
- **RQ2:** Are escrow-based applications efficient? We measure the overhead the escrow introduces compared to applications written without using the escrow (Section 3.6.2).
- **RQ3:** What are the performance trade-offs of different implementations of the escrow's relational engine? We evaluate the performance of three approaches: Materialized Tables, Virtual Tables, and Virtual Tables with Pushdown (Section 3.6.3).

3.6.1 RQ1: Dataflows in Real-World Apps

We design a rubric to collect a representative sample of iOS apps containing various dataflows involving sensitive personal data, and we describe how those dataflows can be reimplemented using the escrow's programming interface.

Rubrics for selecting apps. We selected 10 representative apps based on the following rubric [4]:

- Each app needs to be open-sourced in GitHub so we can find dataflows in its codebase, and it should be currently maintained (i.e., not archived and has commits in the recent month)
- Each app needs to be relatively popular for it to be representative, thus we only select apps with over 1000 Github stars.
- The apps need to contain dataflows covering different types of sensitive data.

4. We include Covid Alert (Canada's COVID contact tracing app) [Canadian Digital Service, 2025], which does not fit all rubrics but has interesting dataflows that are worth discussing in the chapter.

- The apps need to be diverse, spanning multiple categories.

	Bluesky	**Covid Alert**	**DuckDuckGo**	**Nextcloud**	**Signal**	**Simplenote**	**VLC**	**Wikipedia**	**WordPress**	**Home Assistant**
Category	Social media	Contact tracing	Browser	File Management	Messenger	Notes	Media player	Wikipedia	Blogs	Home automation
Contact dataflow	N/A	N/A	N/A	N/A	Find other users from contacts	Share notes with other contacts	N/A	N/A	N/A	N/A
Location dataflow	N/A	N/A	Share location with websites to obtain location based results	Show location on map	Reverse geocode, search and share location	N/A	N/A	Search and recommend articles based on location	Add current location to posts	Send location to local Home Assistant server
Image/ video dataflow	Set profile pictures, create posts	N/A	Perform voice search	Upload image/video to user's cloud	Send image/ video	N/A	Control Apple system event to resume or pause media	N/A	Add photo/media to posts	Perform voice assist
Other dataflow	N/A	Record bluetooth keys and match keys with positive Covid diagnosis	N/A	N/A	Transfer app data to another device via LAN	N/A	Access Desktop/ Documents/ Downloads folder, and network/ removable volume	N/A	N/A	Sends motion data and focus status to Home Assistant

Table 3.2: Dataflows of representative apps. N/A means no dataflow exists in the app

Analysis. We analyze each app's source code, identify dataflows involving sensitive personal data, and summarize our findings in Table 3.2. We include one row describing the dataflow involving contacts, location, and image/video respectively, since most apps contain dataflows that involve at least one of the three data types. We also include one row describing other dataflows.

Location dataflow. Six apps use individuals' current location for various purposes. Five of them (DuckDuckGo [DuckDuckGo, 2025], Signal [Signal, 2025], Wikipedia [Wikipedia, 2025], Home Assistant [Home Assistant, 2025], WordPress [WordPress, 2025]) share the location outside the individuals' devices with servers. For instance, Wikipedia finds and recommends articles that individuals might be interested in based on their surrounding region. The app makes a search request to Wikimedia (the parent company of Wikipedia) server with the individual's surrounding region as input, and the server returns relevant articles. We show a simplified version of how the dataflow can be implemented using the escrow based on

Wikipedia's current implementation:

```
// compute() is written as a trailing closure
Escrow.run("SELECT longitude, latitude FROM Location ORDER BY timestamp
LIMIT 1") { coordinates -> SearchResults in
    let region = getSurroundingRegion(
        location: coordinates, radius: 1000)
    // Create search request using Wikimedia's API
    let url = wikiMediaAPI(region: region)
    let results = fetchArticles(url: url, limit: 50)
    return results }
```

Although the exact details of how Wikimedia finds articles based on a given location are unknown, implementing the dataflow using the escrow makes it transparent exactly *what* data is leaving the individual's device and its purpose through COMPUTE(). In the case of Wikipedia, the search request only needs the approximate region that centers around the individual's current location by a specified radius, and the precise location coordinates or other identifying information should not leave the device. The app only has access to the fetched articles as the output of RUN and nothing else.

Contact dataflow. Two apps, Signal [Signal, 2025] and Simplenote [Simplenote, 2025], contain dataflows involving contacts. Signal performs "contact discovery" by fetching the individual's contacts (specifically phone numbers) and finding which contacts are Signal users. Notably, the computation takes place on Signal servers, which run in a secure hardware enclave with oblivious RAM to ensure Signal or anyone else does not gain access to individuals' contact data [Signal, 2022]; essentially, the computation is equivalent to a private set intersection between individuals' contacts and Signal's users [Popa, 2024]. A simplified implementation of this dataflow using the escrow may look like:

```
Escrow.run("SELECT phoneNumber FROM Contacts") {
```

```
phoneNumbers -> SignalAccounts in
    let url = SignalEnclaveURL()
    let matched = contactDiscovery(phoneNumbers, url)
    return matched }
```

While technically the mechanisms Signal implemented for contact discovery would already ensure individuals' contact data remain private and not accessible to anyone else, implementing this dataflow via the escrow makes it transparent that this computation *only* takes place in Signal's enclave and the data is not used for other purposes. In addition, it is conceivable that the contact discovery can take place in an escrow-controlled server with similar enclave technologies, so individuals gain more control over the specific usage of their data. Lastly, it is worth noting that such emphasis on privacy-preserving computation, like what Signal has implemented, is *not* the current industry norm [Zuo et al., 2019].

On the other hand, Simplenote lets individuals pick which contact they would like to share their notes with, then sends a request to the contact (e.g., via email). This dataflow can also be implemented using the escrow; for instance, ACCESS() fetches necessary contact records (e.g., name and email), and COMPUTE() implements the interface that allows the individual to pick which contact to share with, then sends the request to the contact. The escrow makes it transparent what data Simplenote has access to (in this case it would only have access to the specific contact the individual picked) and how the data is used.

Image/video dataflow. Seven apps have read access to the device's Photos Library. Most apps access the Photo Library for functionalities such as setting avatars (Bluesky [Bluesky, 2025]), creating posts (Bluesky, WordPress [WordPress, 2025]), sending / uploading messages / files (Nextcloud [Nextcloud, 2025], Signal), or playing videos (VLC [VLC, 2025]). We concentrate on one unique use case from two apps (DuckDuckGo, Home Assistant), which is speech recognition. Generally, there are two ways to perform speech recognition: one is to transcribe a recording that's already stored on-device, and the other is to transcribe live

speech. Implementing the former task using the escrow is straightforward; for example, to transcribe the most recently stored audio recording, developers can write:

```
Escrow.run("SELECT asset FROM Photos WHERE mediaType =='audio' ORDER BY
creationDate LIMIT 1") { recording -> String in
    let recognizer = SpeechRecognizer()
    let text = recognizer.transcribe(recording)
    return text }
```

The SQL query fetches the most recently created asset stored in the Photos library that has an audio media type. The query returns a native `PHAsset` object that can be directly consumed by the `SpeechRecognizer`. On the other hand, transcribing a live speech is generally achieved by storing the individual's live speech in a buffer in memory. When the buffer is full (or after a small time interval), the speech recognizer will transcribe the new buffer (along with the previously stored buffers), thus achieving the effect of transcribing speech on the fly. Since no speech data is physically stored on-device, the escrow cannot access the data, thus we consider implementing this dataflow to be outside the scope of this chapter. However, it is possible that by integrating with Apple's infrastructure the escrow will be able to access structured data that is stored in memory and enable dataflows such as live speech transcription.

Other dataflows. Four apps contain dataflows that do not involve contact, location, or image/video data, each with its distinct purpose. All dataflows from Covid Alert, VLC, and Home Assistant can be implemented using the escrow except Signal, since Signal's dataflow involves app-specific data that is not accessible using Apple SDKs. We explain how to implement the dataflow from Covid Alert, which we consider to be the most interesting one among the four apps.

Covid Alert is Canada's official Covid contact tracing app. The app uses Apple's Exposure Notification framework [Apple and Google, 2025]. It works by recording each individ-

ual's on-device Bluetooth keys when their device exchanges Bluetooth beacons with other devices. When an individual receives a positive COVID diagnosis, they can voluntarily share their keys with the government-controlled remote key server, which stores all Bluetooth keys from individuals who recently tested positive for COVID. Periodically, the app downloads keys from the key server and tries to find whether the individual's keys from the last 14 days match any of the downloaded keys (from those who tested positive). If there is a match, the app notifies the individual.

The two important dataflows—sharing keys with the key server when an individual tests positive, and matching the individual's keys with keys downloaded from the key server, can both be implemented using the escrow. For example, an approximate implementation of the former dataflow may look like:

```
Escrow.run("SELECT * FROM KeyTable ORDER BY data LIMIT 14") { keys in
    let url = keyServerURL()
    shareKeysWithServer(url) }
```

Using the escrow provides transparency of the dataflow and ensures that the individual's private health information is not exposed to anyone else other than the government-controlled key server.

In summary, we show that it is feasible to re-implement many dataflows in real-world apps via the escrow's programming interface. We note that since the apps we collected are all open-source, they are a "biased" sample in the sense that the apps are already designed with transparency as a priority. Many closed-source apps have undisclosed dataflows for purposes not covered in our analysis (e.g., profiling, resale). Enforcing those dataflows to be implemented using the escrow would enable individuals to gain transparency and control over these dataflows.

3.6.2 RQ2: Runtime Efficiency of the Escrow

We measure the overhead introduced by the escrow by comparing the time to access data and run computation on the data using the escrow's RUN function versus without the escrow (i.e., the baseline) for different data sizes. Table 3.3 shows the dataflows used in our evaluation, which involve three data types that are representative in many of today's apps, as we have shown in Section 3.6.1.

	Data Access	Computation
Contacts	Get the phone number of the contact whose given name is 'uniqueName' `SELECT phoneNumber FROM Contacts WHERE givenName = 'uniqueName'`	Check if the result is a valid US phone number
Photos	Get the most recent 100 images (as `PHAsset`s) from the Photos library `SELECT phasset FROM Photos WHERE mediaType=='image' ORDER BY creationDate DESC LIMIT 100`	Transform the result to a list of `NSImage`
Location	Get the most recent location (as `CLLocation`) `SELECT cllocation from Location ORDER BY timestamp DESC LIMIT 1`	Get weather using OpenWeatherMap [OpenWeatherMap, 2025] API

Table 3.3: Dataflows run by the escrow-based and baseline app, which uses SQL query and Apple SDKs to access data respectively

Implementation. We implement an escrow-based app, which uses the escrow's RUN function to access data and run computation, and the baseline app, which uses Apple SDKs to access data. The computation functions in both apps are implemented in a similar way and the output of the computation are exactly the same.

The escrow-based app uses the escrow as a static singleton instance (note that the escrow prototype is deployed as a library). We implement its relational engine using Virtual Tables with Pushdown. The schemas of the virtual tables are:

- `Contacts`: `id TEXT, givenName TEXT, familyName TEXT, phoneNumber TEXT`
- `Photos`: `id TEXT, phasset PHAsset, mediaType INT, creationDate DATE, collectionID TEXT, collectionName TEXT`
- `Location`: `cllocation CLLocation, longitude REAL, latitude REAL, horizontalAccuracy REAL, timestamp TIMESTAMP`

We implement projection pushdowns for all three virtual tables. In addition, for Contacts table, we implement predicate pushdown for all four columns, For Photos table, we implement

predicate pushdown for `id`, `mediaType`, `collectionID` and `collectionName`, and order by / limit pushdown for `creationDate`. For Location table, we implement order by / limit pushdown for `timestamp`.

Data preparation. We prepare and run the experiments for different data sizes. For contacts, we add 100, 1k, 10k, and 100k contacts records to the Contacts app; each record has a unique given name, family name, and phone number. For photos, we add 100, 1k, 10k, and 100k photos to the Photos library; each photo is 16KB. Location data is updated in real time.

Configurations. The apps are implemented as multi-platform (iOS/macOS) apps in Swift 6.1. We deploy the app as a macOS app and run the experiments using a 2021 MacBook Pro with Apple M1 Pro chip, 16 GB memory, and macOS 15.6.1, since iPhones cannot handle the large data volumes we will insert.

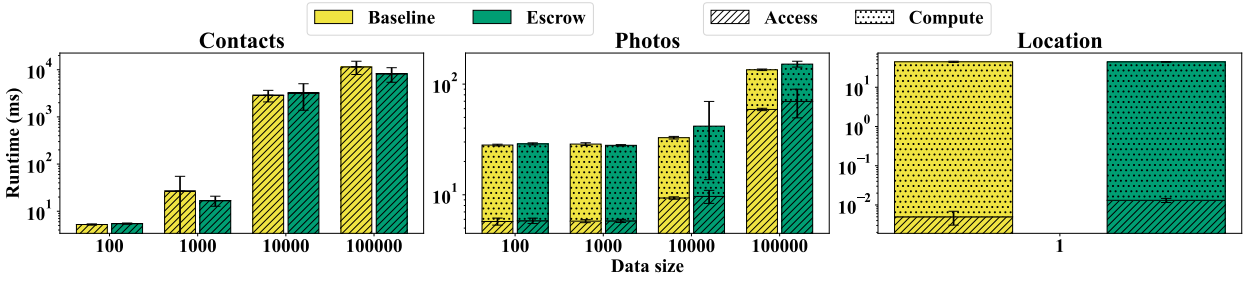

Figure 3.3: Average access and compute runtime (with std) by escrow-based app vs baseline app over 10 runs

Result. We run each app 10 times for each dataflow and data size, and we show the average access and computation time separately (with standard deviation) in Figure 3.3. In short, the escrow introduces very little overhead. For contacts dataflows, the runtime is dominated by the data access time, since checking whether a given number is a valid US phone number only involves regex matching and is very fast. Because the escrow executes the SQL query via the Contacts virtual table, which implements pushdown for projections and predicates, the database does not perform additional data filtering, hence the overhead is small compared

to fetching data directly using Apple's Contacts SDK. For photos dataflow, as data size increases, the majority of runtime shifts from computation to data access, since the number of photos to process stays the same in the computation function. Similarly to Contacts, the Photos virtual table also implements pushdown so there is no additional data filtering on the database side. For Location dataflows, the runtime is dominated by the computation time, since data is transferred to and from the OpenWeatherMap server.

In summary, the escrow's overhead depends on the specific dataflow and the escrow's relational engine implementation. Since the escrow's programming interface is minimal, the main overhead comes from executing the data access function. If the dataflow is data access heavy instead of computation heavy, then the overhead will be more significant. Implementing pushdown in virtual tables reduces the amount of data filtering the database needs to do, especially if the data access function is selective, and implementing the zero-copy mechanism for passing native Swift objects through the relational engine ensures there is no serialization overhead, thus reducing the overall overhead.

3.6.3 RQ3: Relational Engine Comparison

We compare the runtime performance of three implementations of the escrow's relational engine (Section 3.4.3): Materialized Tables, Virtual Tables, and Virtual Tables with Pushdown. Table 3.4 shows the queries used in the evaluation. We categorize the queries into full, projection, predicate and order by / limit queries in order to evaluate how each implementation performs for each category.

	Full	Projection	Predicate	Order By / Limit
Contacts	`SELECT * FROM Contacts`	`SELECT familyName, givenName FROM Contacts`	`SELECT * FROM Contacts WHERE givenName = 'uniqueName'`	\
Photos	`SELECT * FROM Photos`	`SELECT phasset FROM Photos`	`SELECT * FROM Photos WHERE collectionName = 'uniqueAlbum'`	`SELECT * FROM Photos ORDER BY creationDate DESC LIMIT 1`
Location	\	\	\	`SELECT * FROM Location ORDER BY timestamp DESC LIMIT 1`

Table 3.4: Queries run by the escrow's relational interface

Setup. All three implementations are exposed to the same schema as described in the previous experiment (Section 3.6.2). We also use the same data preparation and experiment configurations as in the previous experiment. We create a unique album with one photo inside, in addition to the photos we have already added. We only measure the performance of running the *data access* function and there is no computation function involved. All three implementations produce query result in the same format.

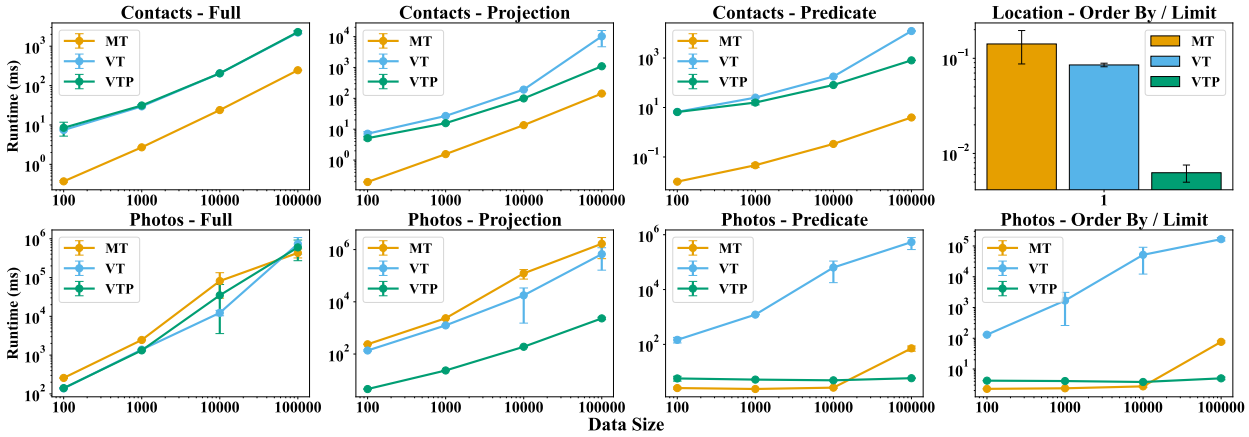

Figure 3.4: Average runtime (with std) to execute queries over 10 runs using Materialized Tables (MT), Virtual Tables (VT), and Virtual Tables with Pushdown (VTP)

	size=100	size=1000	size=10000	size=100000
Contacts	10.85	32.68	279.28	3402.41
Photos	290.72	1714.91	19150.86	130038.08

Table 3.5: Time (ms) to materialize Contacts and Photos table

Result. We run each query 10 times, and we show the average runtime with standard deviation in Figure 3.4. In addition, for Materialized Tables, we report the time to materialize the Contacts and Photos table for each data size in Table 3.5. The time grows linearly against the data size. Materializing Photos table take considerably more time since it needs to enumerate each photo asset and fetch its collection ID and name. The runtime performance of the three implementations varies for different categories of queries. As expected, Virtual Tables with Pushdown perform better than Virtual Tables for all except full (SELECT *)

queries, in which case the two implementations have roughly the same runtime since no pushdown can be applied. Notably, Virtual Tables with Pushdown performs magnitudes faster for highly selective queries (Photos queries with predicate and order / limit), because it asks the `PhotoKit` framework for a single album with only one photo inside or the single most recent photo, avoiding the materialization of unnecessary data, whereas Virtual Tables implementation must filter or sort all photos. It is also the fastest for Photos queries with projection since the other two implementations both need to fetch collection information for each `phasset`, which is costly. Materialized Tables implementation is about an order of magnitude faster than the other two for Contacts queries but not for Photos queries, because for Contacts queries the data is already in the database and SQLite can employ any internal optimization it needs to execute the query efficiently, but for Photos queries the escrow needs to map the result `phasset` IDs to `phasset` objects when the query finalizes, which introduces overhead. Both Virtual Tables implementations use zero-copy mechanism to pass `phasset` objects through the database thus do not incur serialization overhead.

In summary, while Materialized Tables offers speed for full scans, it suffers from high initialization costs and data staleness. By applying classic query optimization principles, Virtual Tables with Pushdown provides the data freshness of on-the-fly data virtualization with performance that is competitive with, and in some cases superior to, querying materialized data.

3.7 Related Work

Personal Data Sovereignty. The concept of personal data sovereignty has emerged as a critical paradigm for rebalancing control in the digital economy [Ernstberger et al., 2023, Polčák and Svantesson, 2017, Hummel et al., 2021, Lauf et al., 2022]. It is founded on the principle that individuals have the right to control their own data, including its collection, storage, and use. This stands in contrast to the current reality where large platforms have

become de-facto data sovereigns, collecting vast amounts of personal data and challenging the traditional notions of individual autonomy and privacy. The goal of personal data sovereignty is to shift control back to the individual. Our work's central goal of *Personal Dataflow Sovereignty* is a direct, technical instantiation of this principle, providing a concrete mechanism for individuals to exercise control over how their data is used. While legal frameworks like the GDPR [European Parliament and Council of the European Union, 2016] provide a foundation for personal data sovereignty, their practical implementation often falls short [Ryan et al., 2024], highlighting the need for a technical foundation like the one we propose. The data management community has a rich history of engaging with the core technical problems under personal data sovereignty, from *Hippocratic Databases* [Agrawal et al., 2002] that articulates a vision for database systems designed with privacy as a core principle, to seminal work on purpose-based access control databases [Byun and Li, 2008], to more recently data escrow system [Xia et al., 2022] that enables delegated computation.

Data Governance Frameworks. The concept of rebalancing power in the data economy has given rise to various data governance frameworks [Abraham et al., 2019, Delacroix and Lawrence, 2019, researc, 2020, Institute, 2021]. These include Data Unions, Data Trusts, and Data Cooperatives, which advocate for collective organizations that would manage data and negotiate on behalf of individuals. A related economic perspective reframes individual-generated data as a form of digital labor, arguing that individuals should be compensated for their contributions to the data economy, a concept known as Data as Labor [Posner and Weyl, 2018, Arrieta-Ibarra et al., 2018]. While these frameworks provide compelling visions for a more equitable data ecosystem, they often lack a concrete technical foundation for their implementation. Our data escrow architecture can be viewed as a critical piece of enabling the technical infrastructure that serves as the trustworthy intermediary required for such frameworks to operate effectively.

Privacy design and enforcement mechanism. A large body of research focuses on

usable privacy and policy design in apps. For instance, many works analyze the effect of privacy labels, finding they often fail to fully disclose intentions for accessing data [Xiao et al., 2023, Andow et al., 2020, Bui et al., 2021, Rajivan and Camp, 2016, Hagen, 2016]. Other works focus on privacy designs to guide individuals toward informed privacy choices [Acquisti et al., 2017, Zhang and Sadeh, 2023, Feng et al., 2021, Zhang et al., 2022, Habib and Cranor, 2022, Schaub et al., 2015, Zhang et al., 2024a]. Recent research has also explored the use of Large Language Models (LLMs) to facilitate privacy comprehension [Duesterwald et al., 2025, Feng et al., 2024]. However, a critical gap exists: usable privacy solutions frequently lack the underlying technical feasibility to be deployed effectively in practice [Iravantchi et al., 2025, Brodie et al., 2005, Fischer-Hübner and Karegar, 2024, Distler et al., 2021, Pan et al., 2024]. Our escrow provides the technical *mechanism* required to enforce dataflow preferences, thereby motivating and providing the necessary foundation for future *policy* design on eliciting individuals' dataflow preferences.

Relational model over heterogeneous data sources. The challenge of providing a unified query interface over multiple heterogeneous data sources is a classic problem in the database community [Halevy, 2001]. Data virtualization techniques are pioneered by early data integration systems like Garlic [Carey et al., 1995], where source-specific wrappers translate queries against a global schema into the native languages or API calls of the underlying sources. This principle is now widely adopted in production systems, for instance, through PostgreSQL's Foreign Data Wrapper (FDW) [PostgreSQ, 2025]. The escrow's relational interface applies this same principle of data virtualization, creating a unified schema over disparate, programmatic APIs to access data. Our approach is also analogous to how TinyDB [Madden et al., 2005] overlays a relational model over data collected from sensor networks.

3.8 Conclusion

In this chapter, we contribute a solution for a critical problem in the modern data economy: the loss of personal data sovereignty. Individuals lack the transparency and control necessary to govern how their personal data is used by platforms. To address this, we propose a data escrow architecture that leverages delegated computation. We design and implement a minimally invasive programming interface and an efficient data virtualization layer to make this model practical and developer-friendly. Our concrete implementation in the Apple ecosystem demonstrates a feasible path to deployment. Finally, our comprehensive evaluation shows both qualitatively and quantitatively that our escrow solution is expressive enough to implement dataflows in a wide range of real-world applications, and that it does so efficiently, with minimal performance overhead.

CHAPTER 4

WITHIN-DATASET DISCLOSURE RISK FOR DIFFERENTIAL PRIVACY

4.1 Introduction

The potential value of releasing statistics from a dataset is offset by the risk of disclosing sensitive information about individuals represented in the dataset. Differential privacy (DP) [Dwork, 2006, Dwork et al., 2006] permits releasing statistics while providing theoretical guarantees of privacy, where the degree of privacy is determined by a *privacy parameter* ϵ — smaller ϵ corresponds to a stronger privacy guarantee. While DP presents a rigorous mathematical proof of privacy, choosing ϵ is challenging in practice since it is difficult to interpret the privacy consequences of making such a choice [Dwork et al., 2019, Cummings et al., 2021, Garrido et al., 2022, Desfontaines, 2024, Smart et al., 2022, Xiong et al., 2020, Nanayakkara et al., 2023, Dibia et al., 2024]. In this chapter, we propose a method to equip DP practitioners with additional information about the impact of choosing ϵ on the *within-dataset individuals*' disclosure risk, and we show that this additional information helps controllers choosing ϵ.

In this chapter, we consider a private data analysis problem with three types of agents: *analyst*, *controller*, and *individuals*. Each *individual* contributes a data record to a trusted *controller*, who gathers those records into a dataset. *Analysts* want to submit statistical queries over the controller's dataset. The *controller* wants to give *analysts* useful answers to their queries as long as the identities of the *individuals* are protected. To achieve this, the *controller* uses differential privacy, which requires them to choose ϵ to determine the level of privacy protection offered to *individuals*.

The privacy parameter ϵ indicates the maximum difference between the probabilities of producing the same query output with or without an arbitrary individual's record, *whether*

the individual is in the controller's dataset or not. This maximum difference measures the *worst-case disclosure risk*, which is the probability of identifying an individual given the DP release of a query output. In other words, ϵ bounds the disclosure risk of *all* individuals across *all* possible datasets. On one hand, this means DP offers a robust privacy guarantee that is independent of any dataset instance. On the other hand, this makes it difficult for controllers to reason about the concrete privacy implications on the *within-dataset* individuals when choosing ϵ. The main challenge of choosing ϵ is that ϵ offers a global and probabilistic worst-case measure of disclosure risk, which complicates the controller's decision-making process [Lee and Clifton, 2011].

In this chapter, we provide controllers additional information that complements the worst-case measure of disclosure risk by introducing a *relative, within-dataset* measure of disclosure risk. We call this **Relative Disclosure Risk Indicator (RDR)**. Given the controller's dataset and the analyst's query, the RDR indicates the disclosure risk of a within-dataset individual *relative* to other within-dataset individuals when choosing a specific ϵ; this is different from another absolute measure of disclosure risk defined by Lee and Clifton [Lee and Clifton, 2011], which is the maximum probability of identifying *any* within-dataset individuals. With RDR, controllers understand the impact of choosing ϵ by observing the *distribution* of disclosure risks among the within-dataset individuals. We hypothesize that RDR helps controllers make more informed choices of ϵ as this additional information empowers controllers to make explicit decisions on their preferred risk distribution over the within-dataset individuals. To test our hypothesis, we run a between-subjects user study where we ask participants to choose ϵ for a given query and dataset, and we present the treatment group with the additional information of the within-dataset individuals' RDRs. We observe that the treatment group makes more consistent choices of ϵ compared to the control group, and all participants in the treatment group used RDRs in their decision-making process, thus demonstrating that RDR is effective in helping participants choose ϵ.

Equipped with RDRs, we make several technical contributions. **First**, we develop the key insight that controllers can express their privacy preferences on the within-dataset individuals as a *function* of the within-dataset individuals' RDRs and a *threshold* over the function output. For instance, controllers can specify the maximum permissible difference between the most at-risk and least at-risk within-dataset individuals. **Second**, we design an algorithm, called Find-ϵ-from-RDR, that takes controllers' privacy preferences in terms of the within-dataset individuals' RDRs and derives an ϵ that satisfies those preferences. Because RDRs are data-dependent, they are only visible to controllers and never to analysts, and the derived ϵ cannot be released to the analysts either. Thus **third**, we introduce an alternative algorithm, called Find-and-release-ϵ-from-RDR, to enable DP release of the derived ϵ. **Lastly**, in practice, analysts may submit multiple queries to the controller, and the controller needs to set a *total privacy budget* of their dataset. Choosing this budget suffers from the same difficulty as choosing ϵ [Dwork et al., 2019, Cummings et al., 2021, Garrido et al., 2022]. We propose an extension to Find-and-release-ϵ-from-RDR that allows controllers to *dynamically* determine whether to allow or deny answering analysts' queries, hence circumventing the need to choose a total budget. We construct a *privacy odometer* [Rogers et al., 2016] to provide an end-to-end DP guarantee for our solution.

Summary of evaluation results. In our evaluation, we test the hypothesis that the new RDR helps controllers choose ϵ by conducting an IRB-approved [1] user study with 56 participants and the study results demonstrate that participants make more consistent choices of ϵ when presented with the additional information of within-dataset individuals' RDRs. We show that our proposed algorithms are efficient and scalable; the runtime scales linearly with the dataset size and takes under a few minutes even for one million data records. We conduct ablation studies to analyze the effect of varying controllers' privacy preferences in terms of the within-dataset individuals' RDRs, and we observe that stricter

1. The Institutional Review Board (IRB) is responsible for reviewing studies involving human research subjects and ensuring adequate protection of participants.

privacy preferences lead to smaller ϵ chosen.

Outline. The rest of the chapter is organized as follows. In Section 4.2, we introduce the background on DP and the relevant agents in a standard DP deployment. In Section 4.3, we introduce the new RDR definition. In Section 4.4, we present the Find-ϵ-from-RDR algorithm. In Section 4.5, we present the Find-and-release-ϵ-from-RDR algorithm and our solution to answer multiple queries without a total privacy budget while providing end-to-end DP guarantee. We finish with evaluation (Section 4.6), related work (Section 4.7), and conclusions (Section 4.8).

4.2 Background

In this section, we first introduce the definition of (ϵ, δ)-differential privacy (Section 4.2.1), then explain the different agents in a standard DP deployment (Section 4.2.2).

4.2.1 Differential Privacy

Differential privacy (DP) [Dwork, 2006, Dwork et al., 2006] is a mathematical definition of privacy that bounds the effect a single individual's record has on the output of a query. Formally, let the dataset $x = (x_1, \cdots, x_n) \in \mathcal{X}^n$ be a set of n records drawn from a data universe $\mathcal{X}$; we assume that each individual (labeled 1 to n) contributes exactly one record to x. A *randomized* mechanism $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy if for any pair of neighboring datasets x and x' where x and x' differ by a single record, and for all sets of possible output $O \subseteq Range(\mathcal{M})$:

$$\Pr(\mathcal{M}(x) \in O) \leq e^{\epsilon} \Pr(\mathcal{M}(x') \in O) + \delta.$$

When the *failure parameter* δ is 0, $\mathcal{M}$ satisfies ϵ-differential privacy. One way of achieving DP is by constructing a mechanism that adds random noise to the query output. The amount

of noise is controlled by a tunable *privacy parameter* ϵ. Smaller ϵ leads to more noise and less accurate output: the choice of ϵ determines the *privacy-accuracy tradeoff* in DP.

In this paper, we focus on two foundational DP mechanisms that controllers frequently use in practice: the Laplace Mechanism [Dwork et al., 2014] and the Gaussian Mechanism [Dwork et al., 2014].

Definition 7 (**Laplace Mechanism**). Given query $q : \mathcal{X}^n \to \mathbb{R}^k$, dataset $x \in \mathcal{X}^n$, $\epsilon > 0$, Laplace Mechanism is defined as:

$$\mathcal{M}_{Lap}(x, q, \epsilon) = q(x) + (Z_1, \ldots, Z_k),$$

where Z_i are i.i.d random variables drawn from the Laplace distribution with probability density function $p_b(z) = \frac{1}{2b}\exp(-|z|/b)$ and the scale $b = \frac{\Delta_1}{\epsilon}$ [2]. The Laplace Mechanism satisfies ϵ-DP.

Definition 8 (**Gaussian Mechanism**). Given query $q : \mathcal{X}^n \to \mathbb{R}^k$, dataset $x \in \mathcal{X}^n$, $\epsilon, \delta > 0$, Gaussian Mechanism is defined as:

$$\mathcal{M}_{Gau}(x, q, \epsilon, \delta) = q(x) + (Z_1, \ldots, Z_k),$$

where Z_i are i.i.d random variables drawn from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with center 0 and variance $\sigma^2 = \frac{2\Delta_2^2 \ln(1.25/\delta)}{\epsilon^2}$ [3]. The Gaussian Mechanism satisfies (ϵ, δ)-DP.

There are two important properties of differential privacy, *post processing* and *sequential composition*, which are building blocks for designing sophisticated mechanisms that satisfy DP.

2. The global sensitivity of q in the ℓ_1 norm is defined as $\Delta_1 = \max\|q(x) - q(x')\|_1$, where x and x' are neighboring datasets in $\mathcal{X}^n$.

3. The global sensitivity of q in the ℓ_2 norm is defined as $\Delta_2 = \max\|q(x) - q(x')\|_2$, where x and x' are neighboring datasets in $\mathcal{X}^n$.

Post processing [Dwork et al., 2014]. DP is immune to post processing, which means any computation on the output of a DP mechanism will not undermine the DP guarantees. Formally, let $\mathcal{M}$ be an (ϵ, δ)-DP mechanism, then for any function f, $f(\mathcal{M}(\cdot))$ is (ϵ, δ)-DP.

Sequential composition [Dwork et al., 2014]. Sequentially combining multiple DP mechanisms on the same input dataset results in quantifiable degradation of privacy. Formally, given k independent mechanisms $\mathcal{M}_1, \ldots, \mathcal{M}_k$ that satisfy $(\epsilon_1, \delta_1), \ldots, (\epsilon_k, \delta_k)$-DP respectively, any function $g(\mathcal{M}_1(\cdot), \ldots, \mathcal{M}_k(\cdot))$ is $(\sum_{i=1}^{i=k} \epsilon_i, \sum_{i=1}^{i=k} \delta_i)$-DP.

4.2.2 Agents in DP Deployment

Figure 4.1 shows a diagram of a standard DP deployment called the *central* DP model [Near et al., 2021, Wagh et al., 2021]. Specifically, we show the dataflows, i.e., how data moves from one agent to another. *Individuals* send their raw data to a *controller*. The *analyst* submits queries to the controller in order to obtain information about the individuals. To protect the privacy of the within-dataset individuals, the controller answers the analyst's queries using a DP mechanism. Crucially, the controller needs to choose the privacy parameter ϵ, which requires them to reason about what is the *worst-case* disclosure risk they are willing to accept not only for the within-dataset individuals, but for all individuals that could be represented in the data universe [Lee and Clifton, 2011].

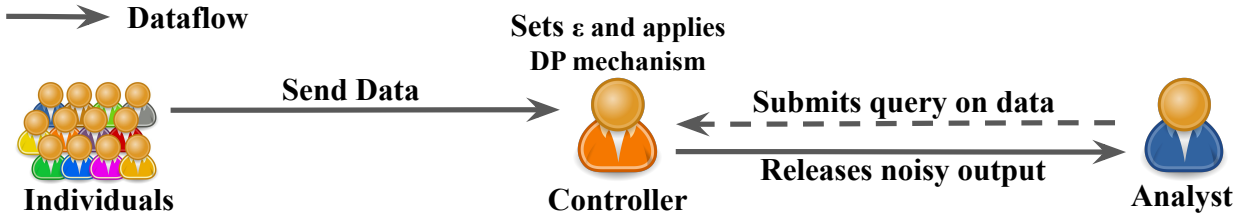

Figure 4.1: Agents in standard DP deployment

One notable example of the central DP model is the application of DP by the US Census in 2020 [Abowd, 2018]. In this case, the Census Bureau is the controller who has access to individuals' (i.e., the US population) data. The analysts' queries are pre-determined by

the Census Bureau; the census data is the output of various census queries (e.g., population counts in different geographical levels). The Census Bureau made considerable effort to choose the privacy parameter ϵ for the census queries [Bureau, 2021] [4]. However, when deploying DP in practice, choosing ϵ is an extremely challenging task for controllers without the assistance of DP experts [Dwork et al., 2019, Garrido et al., 2022, Desfontaines, 2024, Cummings et al., 2024].

While there exist other DP deployment models, i.e., the local DP [Kasiviswanathan et al., 2011] or shuffler model [Erlingsson et al., 2019] where individuals no longer send their raw data directly to the controller, in this chapter we focus on addressing the controller's challenges of choosing ϵ in the central model since this is the most common DP deployment model.

4.3 Relative Disclosure Risk Indicator

In this section, we first explain the intuition behind the need for a Relative Disclosure Risk Indicator (Section 4.3.1). We then propose two concrete definitions of the Relative Disclosure Risk Indicator (RDR) tailored to the Laplace and Gaussian mechanisms as proofs of concept (Section 4.3.2), and discuss an interesting property connecting our definitions to the expected distance of the query output (Section 4.3.3).

4.3.1 Why Choosing ϵ is Hard and RDR Overview

When controllers decide which ϵ to choose for an analyst's query, they may naturally ask the question: *how would choosing this ϵ affect the privacy of the within-dataset individuals I am trying to protect?* However, the value of ϵ itself does not provide a straightforward answer. Because ϵ quantifies a data-independent worst-case privacy guarantee, it is challeng-

4. Technically, the Census Bureau sets the privacy parameter ρ in zero-concentrated differential privacy [Bun and Steinke, 2016], which can be converted to the ϵ in (ϵ, δ)-differential privacy.

ing to translate this privacy guarantee to a practical measure of disclosure risk of specific within-dataset individuals. The global and probabilistic nature of DP gives a robust privacy guarantee, but it also makes it difficult for controllers to reason about the concrete privacy implications for each within-dataset individual when they apply DP in practice.

We hypothesize that *controllers can only comfortably choose ϵ if they understand how each within-dataset individual's privacy is affected by that choice.* A fundamental challenge in deploying differential privacy is translating the abstract guarantee of ϵ into a concrete measure of disclosure risk. Lee and Clifton address this challenge by defining disclosure risk as the maximum posterior probability of re-identifying *any* within-dataset individuals [Lee and Clifton, 2011]. This provides a single, *absolute* measure of the worst-case disclosure risk, establishing a ceiling on the harm to the single most vulnerable within-dataset individual. While valuable, this single-point estimate conceals the underlying *distribution* of risk below that ceiling. A controller informed only of the maximum disclosure risk of the within-dataset individuals cannot distinguish between the scenario where the risk is borne equally by all individuals and one where it is dangerously concentrated on a vulnerable few. Therefore, our goal is to design a *relative, per-individual* indicator that complements this absolute, worst-case measure of disclosure risk; we call it the **Relative Disclosure Risk Indicator (RDR)**. Crucially, the RDR gives controllers a view into the risk distribution of the within-dataset individuals. For instance, by comparing the RDRs of the most and least at-risk individuals, or by examining the overall variance of the RDR distribution, a controller can make a more informed judgment about whether a given ϵ provides an equitable level of protection across the within-dataset individuals. The RDR complements any information controllers already have access to, and it remains private information to controllers only.

4.3.2 The Relative Disclosure Risk Indicator (RDR)

To construct a practical and interpretable indicator, we root our RDR definitions in the concept of *per-instance sensitivity* (PIS) [Wang, 2019].

Definition 9 (**Per-instance sensitivity (PIS)**)**.** Given a fixed dataset $x \in \mathcal{X}^n$ and its neighbor x_{-i} without individual $i \in [n]$'s record, the per-instance sensitivity (PIS) of query $q : \mathcal{X}^n \to \mathbb{R}^k$ is defined as:

$$||q(x) - q(x_{-i})||_p,$$

where p denotes the ℓ_p norm.

Intuitively, PIS signals how sensitive a within-dataset individual's record is; if removing an individual's record causes large differences in the raw query output, then this individual is more *distinguishable* than others and bears a higher baseline risk. We propose two concrete RDR definitions that combine a within-dataset individual's PIS with the scale of the noise introduced by the chosen DP mechanism.

Definition 10 (**RDR for Laplace Mechanism**)**.** Given query $q : \mathcal{X}^n \to \mathbb{R}^k$, dataset $x \in \mathcal{X}^n$, and parameter ϵ, we define the relative disclosure risk indicator (RDR) of an individual $i \in [n]$ under the Laplace Mechanism as:

$$RDR_i = ||q(x) - q(x_{-i})||_1 + k\frac{\Delta_1}{\epsilon}$$

Definition 11 (**RDR for Gaussian Mechanism**)**.** Given query $q : \mathcal{X}^n \to \mathbb{R}^k$, dataset $x \in \mathcal{X}^n$, and parameters ϵ, δ, we define the relative disclosure risk indicator (RDR) of an individual $i \in [n]$ under the Gaussian Mechanism as:

$$RDR_i = \sqrt{||q(x) - q(x_{-i})||_2^2 + k\sigma^2}$$

Interpretation of RDR. The RDR provides a direct lens into how the choice of ϵ com-

presses the risk distribution among individuals. When no DP noise is added to the query output (i.e., as ϵ approaches infinity), the noise component in both RDR definitions (i.e., $k\Delta_1/\epsilon$ and $k\sigma^2$) approaches 0. In this state, RDR reduces to per-instance sensitivity, indicating a within-dataset individual's relative disclosure risk if the raw query output was released. As ϵ gets smaller, the constant noise component becomes larger. Because the noise component is added uniformly to all individuals' PIS, a within-dataset individual with a large per-instance sensitivity will be able to "blend in" with others as their RDRs become dominated by the shared noise component. On the far end of the spectrum, as ϵ approaches 0, the noise term eclipses the PIS, meaning the RDR of every within-dataset individual approaches infinity uniformly. Consequently, no within-dataset individual is at more risk than others, and they share the strongest privacy.

We now make a few remarks:

Limitations of RDR. It is important to note that RDR relies fundamentally on the computation of per-instance sensitivity. Consequently, RDR is only an effective indicator if the controller can accurately compute and inspect the PIS of the within-dataset individuals. A controller's ability to express privacy preferences using RDR is bottlenecked by their ability to interpret these baseline PIS differences.

RDR is only meaningful when compared to other RDRs. RDR is a relative indicator; the value of a within-dataset individual's RDR alone does not indicate absolute disclosure risk, but rather whether they are more or less at risk relative to others in the same dataset.

RDR is data-dependent thus leaks information about within-dataset individuals. Controllers already have access to individuals' raw data, and they can compute the RDRs of the within-dataset individuals from the raw data. The RDR is private information available to controllers *only* and should never be shared with others.

4.3.3 Connection to Expected Output Distance

While we define RDR constructively as a combination of per-instance sensitivity and DP mechanism noise to serve as a practical tool for controllers, we observe an interesting mathematical property regarding these definitions. Specifically, the RDR definitions we propose for the Laplace and Gaussian mechanisms serve as strict upper bounds for the expected distance between the DP output and the $q(x_{-i})$, i.e., the query output without individual i's record.

Formally, for a Laplace or Gaussian mechanism $\mathcal{M}$ producing output o, the definitions satisfy:

$$RDR_i \geq \mathbb{E}_{o \sim \mathcal{M}(x,q,\epsilon,\delta)}[||o - q(x_{-i})||_p],$$

where p denotes the ℓ_1 norm and ℓ_2 norm for Laplace and Gaussian mechanism respectively. We provide the derivations for this property in Appendix A.1. While we do not make formal claims bounding expected privacy loss, this property demonstrates that our RDR definitions capture the mathematical intuition of expected distinguishability. We anticipate that DP practitioners can leverage this intuition to derive other concrete RDRs or indicators for different mechanisms in future work.

4.4 Deriving Epsilon based on RDR

In this section, we describe the key idea of using RDR to express controllers' privacy preferences over the within-dataset individuals (Section 4.4.1), then introduce an algorithm to choose ϵ by using RDR as a more intuitive proxy (Section 4.4.2).

4.4.1 Using RDR to Express Privacy Preferences

The RDR complements the worst-case bound of disclosure risk given by ϵ and becomes an additional indicator of relative disclosure risk associated with each within-dataset individual.

A controller may express their privacy preference over the within-dataset individuals in terms of their RDRs, and choose an ϵ such that the corresponding RDRs satisfy their privacy preference. Next, we convey this intuition through an example.

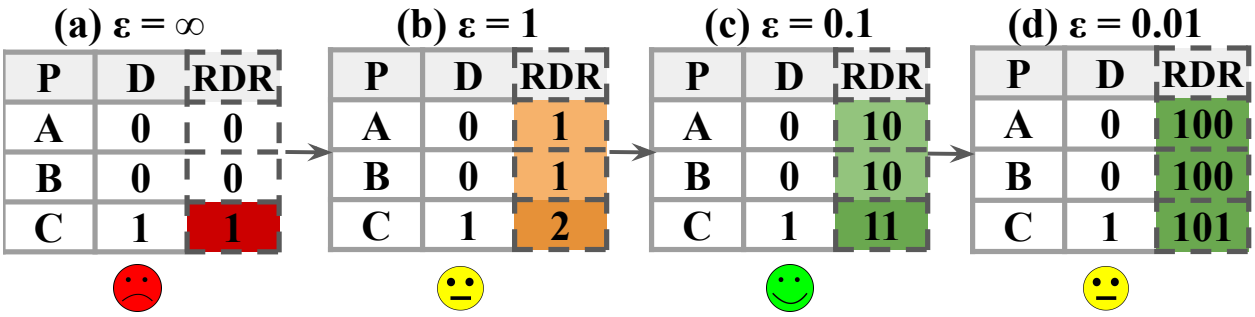

Figure 4.2: Example of using RDR to find ϵ. The query is to count the number of patients (column P) who have a certain disease (column D) and Laplace Mechanism is used to compute the query. For each ϵ, we show the corresponding RDR of each within-dataset patient.

Example illustrating one use of the RDR. Figure 4.2 shows a dataset of three records, where each record consists of a patient's ID (column P) and whether the patient has a certain disease (column D); the value 1 means the patient has the disease and 0 otherwise. Consider an analyst issuing a query about the number of patients who have the disease. To protect the privacy of the within-dataset patients, the controller decides to use the Laplace Mechanism to compute a noisy count. The controller does not know which ϵ they should use, so they first select a set of candidate ϵ values (e.g., $\infty, 1, 0.1, 0.01$) and compute the associated RDR of each within-dataset patient (shown as an additional column by the dataset).

When $\epsilon = \infty$, i.e., when no DP is applied, the controller observes that patient C who is the only one with the disease will have an RDR of 1, and others will have an RDR of 0, which indicates patient C has significantly greater disclosure risk than others (as discussed in Section 4.3.2, when $\epsilon = \infty$, RDR is reduced to per-instance sensitivity). By decreasing ϵ, the resulting RDRs become *closer* across the within-dataset patients. The RDR is computed by adding $\frac{1}{\epsilon}$ to each within-dataset patient's per-instance sensitivity since the Laplace Mechanism is used and the query's sensitivity $\Delta_1 = 1$ (Definition 10). Consider the range

of RDRs for each ϵ: even though the difference between the minimum RDR and maximum RDR of the within-dataset patients is always 1 (this only holds for our specific example [5]), the *ratio* between the minimum and maximum RDR becomes closer to 1 as ϵ gets smaller. In other words, the within-dataset patients' RDRs become relatively closer to each other, and specifically patient C becomes less distinguishable.

The controller chooses ϵ based on their privacy preference in terms of the ratio between the maximum and maximum RDR of the within-dataset patients. Between $\epsilon = 1$ and $\epsilon = 0.1$, the controller might be more comfortable choosing $\epsilon = 0.1$ since the within-dataset patients' RDRs are closer together (the ratio between the minimum and maximum RDR increases to $\frac{10}{11}$ from $\frac{1}{2}$). Between $\epsilon = 0.1$ and $\epsilon = 0.01$, although the latter produces even closer RDRs (the ratio becomes $\frac{100}{101}$), it comes at the cost of worse output accuracy. Thus one sensible choice is to choose $\epsilon = 0.1$ as it yields higher accuracy for the analyst receiving the output.

Key idea. When a controller computes a query by applying a DP mechanism with a specific ϵ, observing the RDRs associated with the within-dataset individuals reveals the distribution of the individuals' disclosure risks. This enables the controller to formally express their privacy preference over the within-dataset individuals by specifying a *preference function* $f : \mathbb{R}^k \to \mathbb{R}$ in terms of individuals' RDRs, e.g., $f(RDR) = RDR_{min}/RDR_{max}$, and a *threshold* $\tau \in \mathbb{R}$ over the output of $f(RDR)$, e.g., a percentage $0 \leq \tau_p \leq 1$. The controller's privacy preference are satisfied when $f(RDR) \geq \tau$. In the above example, if the controller sets $\tau_p = 0.9$, their privacy preference are satisfied when $\epsilon = 0.1$ since $\frac{10}{11} \geq 0.9$.

4.4.2 The Find-ϵ-from-RDR Algorithm

We design the Find-ϵ-from-RDR algorithm (Algorithm 1) to operationalize the key idea presented above. The algorithm provides a general framework for finding a suitable ϵ that aligns

5. In fact, given a fixed dataset, query, DP mechanism, for any ϵ, the maximum difference among the within-dataset individuals' RDRs is the query's local sensitivity [Nissim et al., 2007].

with a controller's specific privacy preference on the within-dataset individuals. Instead of prescribing a single, fixed RDR metric, our approach allows the controller to define their own *preference function* over RDRs, $f(RDR)$, and a corresponding threshold, τ. The algorithm then finds the *largest* candidate ϵ (preferring higher utility) that satisfies the controller's stated preference, $f(RDR) \geq \tau$.

Algorithm 1: Find-ϵ-from-RDR

Input : DP mechanism $\mathcal{M}$, dataset x, query q, preference function f, threshold τ, set of candidate ϵ values $\mathcal{E}$, failure parameter δ

Output: DP output o

```
1  x' ← PROJECT-QUERY-ATTRIBUTES(x, q)
2  PIS ← map()
3  for x_i ∈ x' do
4      if x_i ∉ PIS then
5          PIS[x_i] ← COMPUTE-PER-INSTANCE-SENSITIVITY(x', x_i, q, M)
   /* Controllers may observe the computed per-instance sensitivity of each individual first, and set f
   and τ based on their privacy preferences                                                          */
6  for ε ∈ E do
7      RDR ← list()
8      for x_i ∈ x do
9          RDR_i ← COMPUTE-RDR(PIS[x_i], M, q, ε, δ)
10         RDR.add(RDR_i)
       /* Check whether f(RDR) satisfy controller's privacy preference                              */
11     if f(RDR) ≥ τ then
           /* Found suitable ε.  Compute DP output using chosen ε                                   */
12         o ← M(x', q, ε, δ)
13         return o
14 return nil
```

Choosing preference function. The central argument of our work is that the controller, who has complete knowledge of individuals' data and is responsible for protecting those within-dataset individuals, is the best-positioned agent to define what constitutes an acceptable risk distribution. By first computing and observing the within-dataset individuals' per-instance sensitivities (Algorithm 1, lines 2 - 5, which will be used to compute individ-

uals' RDRs later), the controller gains a concrete understanding of the baseline disclosure risk of each individual. Armed with this information, they can formulate a preference f, τ that is not abstract, but rather grounded in the specific characteristics of their data. This reframes the problem from the difficult task of interpreting a global, worst case ϵ to the more intuitive one of expressing a direct policy on the relative risks they are willing to accept for the individuals they protect.

The flexibility of the preference function is a key strength of our framework, allowing controllers to express a wide variety of privacy preferences. As shown in our previous example, a controller may wish to ensure that no single individual is disproportionately more at risk than any other. This can be expressed using the preference function $f(RDR) = RDR_{min}/RDR_{max} \geq \tau_p$, where τ_p is a value close to 1 (e.g., 0.95). Alternatively, a controller might be less concerned with outliers and more with ensuring the overall risk distribution is tight. This goal can be captured by minimizing variance: $f(RDR) = -Var(RDR) \geq -\tau_{var}$ (equivalent to $Var(RDR) \leq \tau_{var}$), for a small variance threshold τ_{var}. In addition, a controller may be particularly concerned about protecting vulnerable subgroups (e.g., minorities in a demographic dataset). They could define a preference that compares the median risk of a minority group to that of the majority group: $f(RDR) = RDR_{median\_majority}/RDR_{median\_minority} \geq \tau_g$, ensuring the risks between groups remain balanced. The choice of the preference function f and threshold τ is not an arbitrary task but a deliberative, socio-technical process that depends on the controller's privacy goals and their understanding of the data, now complemented with the additional information of the per-instance sensitivities and RDRs of within-dataset individuals.

Detailed Algorithm. Algorithm 1 shows the pseudo-code for finding ϵ from RDR and releasing DP output o. The input includes the set of candidate ϵ values to choose from; for example, the values could range from 10 to 0.01 in a fixed interval. The algorithm first projects relevant query attributes from the dataset (line 1) to avoid unnecessary recomputa-

tion of duplicate data (e.g., if the query only involves the `age` attribute, the algorithm only needs to compute the RDRs of hundreds of unique ages, regardless of the original size of the dataset). Then, for each unique within-dataset individual's data x_i, the algorithm computes its per-instance sensitivity, which depends on the query and which DP mechanism will be used (line 2 - 5); the per-instance sensitivity will be used to compute the RDR of each within-dataset individual. At this point, the controller may observe each within-dataset individual's per-instance sensitivity and sets the preference function f and corresponding threshold τ based on their privacy preference over the within-dataset individuals.

Next, the algorithm iterates over the set of candidate ϵ values in descending order (line 6). For each ϵ, the algorithm computes the RDR of each individual (line 7 - 10). If $f(RDR) \geq \tau$, then the algorithm has found a suitable ϵ that satisfies the controller's privacy preference, so it computes a DP output using the chosen ϵ and returns the output (line 12 - 13). Note that the algorithm does *not* release the chosen ϵ, since ϵ is chosen based on data-dependent RDRs and only the controller can access the RDRs and chosen ϵ.

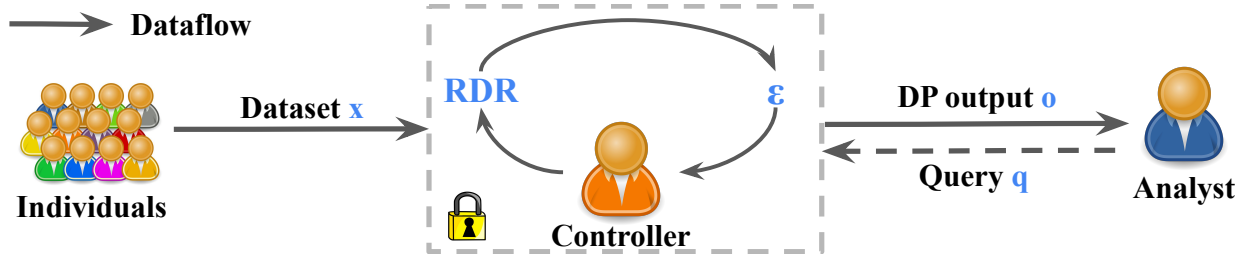

Figure 4.3: Dataflow of `Find-`ϵ`-from-RDR` Algorithm

Dataflows and leakage. It is critically important to understand what information is released to whom. Figure 4.3 shows the involved dataflows in a scenario where a controller uses the `Find-`ϵ`-from-RDR` algorithm to find ϵ and release query output to the analyst. Individuals send their data to the controller; this dataflow remains unchanged from the standard central DP deployment shown in Figure 4.1. The analyst submits a query to the controller. The controller expresses their privacy preference in terms of τ_p (and not in terms

of ϵ) and runs the algorithm to release an output produced by a DP mechanism (Laplace Mechanism or Gaussian Mechanism in our work) with an ϵ selected by the algorithm. At this point, no information has been returned to the analyst yet. The RDRs of within-dataset individuals are *never* released to the analyst. The DP output is only released with the controller's approval. The chosen ϵ is also *not* released since ϵ is chosen based on data-dependent RDRs, and releasing it directly jeopardizes the DP guarantee. In Section 4.5, we explain a modification of the algorithm that permits finding and releasing ϵ while satisfying DP.

Algorithm complexity and optimizations. The algorithm has a worst-case time complexity of $O(n|\mathcal{E}|)$, where n is the number of individual records in the dataset. Computing the RDRs requires computing the per-instance sensitivity first. When the number of individual records is high, this becomes a performance bottleneck: using memoization saves computation if the number of unique projected records is smaller than n. In addition, the computation of each individual's per-instance sensitivity can be parallelized since they are independent of one another, i.e., the problem is embarrassingly parallel. Thus, we implement parallelism and compute the per-instance sensitivity for a chunk of records in each process. We show in Section 4.6.2 that the algorithm is both efficient and scalable.

4.5 Releasing Epsilon Privately and Bounding Privacy Loss Across Queries

In Section 4.5.1, we propose a modification of the `Find-`ϵ`-from-RDR` algorithm that permits finding and releasing ϵ while preserving DP. In Section 4.5.2, we discuss how the new algorithm answers multiple queries without the need to set a total privacy budget, and we provide the end-to-end DP guarantee in Section 4.5.3.

4.5.1 Finding and Releasing ϵ using SVT

Since `Find-ϵ-from-RDR` finds ϵ in a data-dependent manner, releasing the chosen ϵ directly may reveal additional information about the within-dataset individuals. Therefore, a modification of the algorithm is needed to preserve the DP guarantee if the controller wants to release both ϵ and DP output o to the analyst.

Key Idea. The `Find-ϵ-from-RDR` algorithm iterates through candidate ϵ values and selects the first one satisfying the controller's privacy preference, i.e., $f(RDR) \geq \tau$. Our key idea is to make this selection process itself differentially private. We leverage the Sparse Vector Technique (SVT) [Dwork et al., 2009] to test candidate ϵ values and release the first value such that $f(RDR)$ *approximately* passes the threshold, thereby ensuring releasing ϵ preserves DP.

Algorithm 2: SVT algorithm proposed by Lyu et al. [Lyu et al., 2016]

Input : Dataset x; a stream of SVT queries $\mathcal{Q}_{svt} = q_1, q_2 \cdots$ with global sensitivity at most Δ_{svt}; threshold τ; privacy budget ϵ'_{svt} and ϵ''_{svt}

Output: A stream of answers $a_1, a_2 \cdots$ where each $a_i \in \{\top, \bot\}$

$\rho = Lap\left(\frac{\Delta_{svt}}{\epsilon'_{svt}}\right)$
for $q_i \in \mathcal{Q}_{svt}$ **do**
  if $q_i(x) + Lap\left(\frac{2\Delta_{svt}}{\epsilon''_{svt}}\right) \geq \tau + \rho$ **then**
    Output $a_i = \top$
    return
  else
    Output $a_i = \bot$

The SVT algorithm. Algorithm 2 shows the standard SVT algorithm [6] proposed by Lyu et al. [Lyu et al., 2016], which we will incorporate into our algorithm. SVT takes a dataset, a stream of queries (each with a global sensitivity of at most Δ_{svt}), threshold τ, and privacy

6. The algorithm from [Lyu et al., 2016] has an additional parameter c (the maximum number of queries the algorithm can answer $\top$). We set $c = 1$, since we only need one positive answer for our modified algorithm.

parameter ϵ_{svt} that is divided into ϵ'_{svt} and ϵ''_{svt} (i.e., $\epsilon_{svt} = \epsilon'_{svt} + \epsilon''_{svt}$), then releases a stream of answers indicating whether each query output is above or below the threshold. The algorithm works by first initializing a Laplacian noise ρ scaled to $\Delta_{svt}/\epsilon'_{svt}$ (line 1). For each query, it generates another term of Laplacian noise scaled to $2\Delta_{svt}/\epsilon''_{svt}$ and adds the noise to the query output. Then the algorithm compares the noisy query output with the noisy threshold (line 3); if it is above or equal to the threshold, the algorithm outputs $\top$ indicating the query output *approximately* passes the threshold, otherwise, it outputs $\perp$. Overall, the algorithm satisfies ϵ_{svt}-DP.

A Low-Sensitivity SVT Query. Intuitively, the controller's preference function becomes the SVT query. While the `Find-`ϵ`-from-RDR` algorithm offers the controller the flexibility to define any preference functions, many functions have high global sensitivity, making them unsuitable for SVT. For example, the function RDR_{min}/RDR_{max} has a sensitivity of 1. This means to apply SVT, the algorithm will add a Laplacian noise scaled to $1/\epsilon'_{svt}$ to τ_p and another Laplacian noise scaled to $2/\epsilon''_{svt}$ to RDR_{min}/RDR_{max}. Clearly, when ϵ_{svt} is small (e.g., less than 1), the threshold will be too noisy and make the algorithm useless for finding ϵ.

To address this caveat, we propose an SVT query based on the *variance of normalized RDRs*. We first normalize each within-dataset individual's RDR, RDR_i, by a global, data-independent RDR upper bound, RDR^*_{max}, which depends only on the query's global sensitivity, output dimensionality k, and the candidate ϵ value. Based on Definition 10 and 11, $RDR^*_{max} = \Delta_1 + k\Delta_1/\epsilon$ if Laplace Mechanism is used, and $RDR^*_{max} = \sqrt{\Delta_2^2 + k\sigma^2}$ if Gaussian Mechanism is used. Then the normalized RDR, RDR'_i, for a candidate ϵ is $RDR'_i(\epsilon) = RDR_i(\epsilon)/RDR^*_{max}(\epsilon)$.

Definition 12 (SVT query). The SVT query for each candidate ϵ is defined as:

$$q_{svt}(\epsilon) = Var(\{RDR'_i(\epsilon) | i \in [n]\}),$$

where n is the number of within-dataset individuals.

The SVT query has a global sensitivity $\Delta_{svt} = 1/n$, since $0 \leq RDR'_i(\epsilon) \leq 1$. This means the noise added by SVT will be very small for reasonably large datasets.

Algorithm 3: Find-and-release-ϵ-from-RDR

Input : DP mechanism $\mathcal{M}$, dataset x, query q, threshold τ_{var}, set of candidate ϵ values $\mathcal{E}$, failure parameter δ, SVT budget ϵ_{svt}

Output: Chosen ϵ, DP output o

1 $\epsilon'_{svt}, \epsilon''_{svt} \leftarrow$ ASSIGN-SVT-BUDGET(ϵ_{svt})
2 $\rho = Lap\left(\frac{\Delta_{svt}}{\epsilon'_{svt}}\right)$
3 $x' \leftarrow$ PROJECT-QUERY-ATTRIBUTES(x, q)
4 $PIS \leftarrow map()$
5 **for** $x_i \in x'$ **do**
6 **if** $x_i \notin PIS$ **then**
7 $PIS[x_i] \leftarrow$ COMPUTE-PER-INSTANCE-SENSITIVITY(x', x_i, q, M)

/* Controllers may observe the computed per-instance sensitivity of each individual first, and set τ_{var} based on their privacy preferences */

8 **for** $\epsilon \in \mathcal{E}$ **do**
9 $RDR \leftarrow list()$
10 **for** $x_i \in x$ **do**
11 $RDR_i \leftarrow$ COMPUTE-RDR($PIS[x_i], M, q, \epsilon, \delta$)
12 $RDR.add(RDR_i)$

/* Run SVT to find the first ϵ that approximately passes threshold */

13 **if** $-q_{svt}(\epsilon) + Lap\left(\frac{2\Delta_{svt}}{\epsilon''_{svt}}\right) \geq -\tau_{var} + \rho$ **then**
14 $o \leftarrow \mathcal{M}(x', q, \epsilon, \delta)$
15 **return** ϵ, o
16 **return** nil

The `Find-and-release-`ϵ`-from-RDR` algorithm. Algorithm 3 integrates this SVT-based query. First, the algorithm allocate the SVT budget ϵ_{svt} (line 1); one allocation strategy [Lyu et al., 2016] is to have $\epsilon'_{svt} : \epsilon''_{svt} = 1 : 2^{2/3}$ to achieve better accuracy of SVT. Then it iterates over each candidate ϵ and computes the corresponding RDRs the same way as in Algorithm 1 (line 3 - 12). For each ϵ, it computes the SVT query and tests if the variance is approximately below the threshold τ_{var} (line 13). The first ϵ that passes this noisy test is

selected and released along with the final DP output o (line 15).

The main advantage of using SVT is that it always consumes a fixed budget ϵ_{svt} *regardless of the number of SVT queries* (i.e., the number of candidate ϵ values). Controllers understand the consequence of running SVT since they know exactly how much noise is added to both the SVT query and the threshold. They can decide whether they are comfortable choosing an ϵ that does not exactly conform to their privacy preferences; if not, they may use a different ϵ_{svt} or τ_{var} and rerun the algorithm.

Claim: The `Find-and-release-`ϵ`-from-RDR` algorithm satisfies $(\epsilon + \epsilon_{svt}, \delta)$-DP. The algorithm can be divided into two steps. In the first step, it applies SVT, which is ϵ_{svt}-DP, to determine whether each candidate ϵ satisfies the controller's privacy preferences. In the second step, if a suitable ϵ is found, it uses an (ϵ, δ)-DP mechanism to compute output o, and releases both ϵ and o. Therefore, by *sequential composition* [Dwork et al., 2014], the algorithm satisfies $(\epsilon + \epsilon_{svt}, \delta)$-DP.

Claim: Assume the SVT budget ϵ_{svt} budget is split such that $\epsilon'_{svt} = \epsilon_{svt}/3$ and $\epsilon'_{svt} = 2\epsilon_{svt}/3$. **For any failure probability $\beta \in (0, 1)$, `Find-and-release-`ϵ`-from-RDR` algorithm is (α, β)-accurate, where** $\alpha = \frac{6}{n\epsilon_{svt}} \ln(\frac{2|\mathcal{E}|}{\beta})$; the accuracy of the algorithm refers to the correctness of the chosen ϵ. This means that with probability at least $1 - \beta$, if the algorithm returns a value ϵ^*, then the true variance associated with that value satisfies $q_{svt}(\epsilon^*) \leq \tau_{var} + \alpha$. And for any candidate $\epsilon' \in \mathcal{E}$ such that $q_{svt}(\epsilon') < \tau_{var} - \alpha$, the algorithm will not halt at a value $\epsilon < \epsilon'$, thus returning a value at least as large as ϵ'(since candidate ϵ values are tested in descending order). A complete proof is provided in Appendix A.2.

4.5.2 Bounding Privacy Loss Across Queries

The preceding discussion is valid for one query. In practice, analysts often want to run multiple queries. Combining these queries' outputs will reveal new information, eventually breaking privacy as per the fundamental Law of Information Recovery [Dwork et al., 2014].

In today's deployment of DP, controllers need to set a total privacy budget on the dataset *a priori* and stop accepting queries when the composed ϵ of all answered queries exceeds the budget. The need for setting a total budget poses a critical challenge for controllers:

Controllers do not know the privacy implications of total privacy budget. Similar to choosing ϵ for a single query, when choosing the total privacy budget, controllers face the same challenge of not knowing the actual privacy implications on the within-dataset individuals after releasing multiple outputs. Oftentimes, they resort to selecting the budget arbitrarily [Dwork et al., 2019].

The solution. Instead of requiring controllers to choose a fixed total budget, the algorithm keeps track of the already consumed budget on a dataset, i.e., the sum of $\epsilon + \epsilon_{svt}$ of all previously answered queries by the `Find-and-release-`ϵ`-from-RDR` algorithm; we call it ϵ_c. For an incoming query, the algorithm *truncates* the range of candidate ϵ values to consider only those larger than ϵ_c. The algorithm rejects the query if it fails to find a suitable ϵ larger than ϵ_c, meaning the controller's privacy preferences cannot be met for any candidate ϵ.

Our solution creates a *dynamic* bound of total privacy loss. As ϵ_c keeps growing for each answered query, it becomes more difficult to find a suitable ϵ that is larger than ϵ_c since larger ϵ may not produce RDRs that satisfy controllers' privacy preferences. At each point the algorithm fails to find a suitable ϵ for a query, the total privacy loss is bounded. The bound is refreshed if a suitable ϵ is found for a different query. A potential concern is whether there is additional privacy leakage using this dynamic bound. In Section 4.5.3, we show the bound can be formalized as a *privacy odometer* in *fully-adaptive composition* [Rogers et al., 2016] and there is no leakage.

Considering analysts' preferences over queries. A natural tension exists between analysts and controllers: analysts want more accurate output, whereas controllers want to protect the within-dataset individuals' privacy by releasing noisy output. Because it gets increasingly harder to find a suitable ϵ after answering multiple queries as ϵ_c keeps increasing,

analysts may need to prioritize queries of which they need more accurate output. One approach to incorporate analysts' preferences when finding ϵ is to allow them to assign weights (i.e., priority scores) to their queries, then controllers can adjust the threshold τ based on analysts' preferences, e.g., they can set a larger τ_{var} for high-priority queries so the algorithm can find larger ϵ. Nevertheless, solving related problems, such as how to communicate analysts' preferences to controllers, and how to help controllers decide an appropriate τ that takes both analysts' accuracy preferences and their own privacy preferences into account, are outside the scope of the chapter but remain interesting directions to explore.

4.5.3 End-to-End DP Guarantee

Implications of choosing ϵ *adaptively.* If the algorithm has only answered one query, releasing the query output to the analyst satisfies DP and the total privacy loss is bounded by $\epsilon + \epsilon_{svt}$ in `Find-and-release-`ϵ`-from-RDR`. However, if the algorithm answered multiple queries (for the same dataset), for each query, ϵ is chosen *adaptively* because it is chosen based on previously answered queries — a suitable ϵ needs to be larger than ϵ_c. Then the total privacy loss is not bounded by any fixed constant since the algorithm does not require setting a total privacy budget. This brings the question of whether releasing multiple query outputs computed using adaptively chosen ϵ causes additional privacy leakage.

Privacy odometer* in *fully-adaptive composition. To answer this question, we construct a *privacy odometer* [Rogers et al., 2016], which provides a probabilistic running upper bound on the realized privacy loss after answering each query. The privacy odometer is proposed to provide privacy guarantees in *fully-adaptive composition* [Rogers et al., 2016], in which the privacy parameters are chosen adaptively based on previous queries. In short, *there is no additional leakage to choose privacy parameters adaptively if the privacy bound is computed using sequential composition.* Therefore, the privacy odometer of the `Find-and-release-`ϵ`-from-RDR` algorithm under sequential composition [Dwork et al.,

2014] is defined as:

Definition 13 (**Privacy Odometer of Find-and-release-ϵ-from-RDR**). Given dataset x, for a sequence of m queries answered by Find-and-release-ϵ-from-RDR using a fixed failure parameter δ, from which the i-th query uses a fixed SVT budget $\epsilon_{svt}^{(i)}$ and selects $\epsilon^{(i)}$, the algorithm's *privacy odometer* is the function $\texttt{COMP}_{\delta_g}(\cdot)$, where

$$\texttt{COMP}_{\delta_g}(\epsilon^{(1)}, \cdots, \epsilon^{(m)}) = \begin{cases} \sum_{i=1}^{i=m} \epsilon_{svt}^{(i)} + \epsilon^{(i)} & \text{if } \delta_g \geq \sum_{i=1}^{i=m} \delta \\ \infty & \text{otherwise} \end{cases}$$

The privacy odometer updates its running upper bound on privacy loss to $\epsilon_c = \sum_{i=1}^{i=m} \epsilon_{svt}^{(i)} + \epsilon^{(i)}$ after the algorithm answers all m queries, and releasing the composed output on the same dataset x satisfies (ϵ_c, δ_g)-DP.

It is also worth noting that under certain conditions [Whitehouse et al., 2023], the privacy odometer achieves the same asymptotic bound as advanced composition [Dwork et al., 2010]. For simplicity, we define the privacy odometer of our algorithm under sequential composition instead of advanced composition, and we leave further improvements for future work.

4.6 Evaluation

We answer the following research questions:

- **RQ1**: Does presenting the within-dataset individuals' RDRs help controllers choose ϵ?
- **RQ2**: Are our proposed algorithms, `Find-`ϵ`-from-RDR` and `Find-and-release-`ϵ`-from-RDR` efficient and scalable?
- **Microbenchmarks**: What is the effect of varying τ_p in `Find-`ϵ`-from-RDR` and τ_{var} in `Find-and-release-`ϵ`-from-RDR`?

4.6.1 RQ1: Does RDR Help Controllers Choose ϵ?

To study whether presenting the within-dataset individuals' RDRs helps controllers choose ϵ, we conducted an IRB-approved between-subjects user study where the participants were divided into control and treatment groups. To prevent learning effects, only the treatment group sees the within-dataset individuals' RDRs. We chose to conduct the study in the form of an online survey because our tasks do not require in-person guidance, and it is more convenient for participants to complete the study. In addition, it allows us to collect data from a more representative sample.

Recruitment and data collection

We recruited 100 participants using Prolific [pro, 2026]. To complete the study, each participant must sign a consent form, which describes the purpose and content of the study, compensation, and data confidentiality. We did not collect identifiable information, and the survey responses were anonymized, meaning the participant's IP address, location data, and contact information were not recorded.

Prescreening criteria. When recruiting on Prolific, we prescreened participants based on three criteria: i) they are fluent in English, to avoid confusion due to misunderstanding language; ii) they have completed an undergraduate degree or above; iii) they study Computer Science, Computing (IT) or Mathematics; this is to ensure that the participants have a sufficient quantitative background to complete the study.

Tutorial and Screening Quiz. Since the main questions in the study ask the participant to choose ϵ, the participant needs to have basic knowledge about DP and understand the implications of the ϵ they chose. To ensure each participant knows enough about DP to complete the study, we asked each participant to read a short tutorial about DP at the beginning. This tutorial explains the intuition of DP and the effect of the parameter ϵ at a high level. Mainly, it explains that DP is generally achieved by adding random noise to

the query output, and larger ϵ means less noise will be added and vice versa. After reading the tutorial, the participant needs to complete a screening quiz that tests their knowledge about the tutorial they just read. Only participants who pass the quiz proceed to conduct the study. Of the 100 participants we recruited, 56 passed the quiz and their responses were used for the study analysis. This quiz ensures that participants have understood the principles of DP, thus strengthening the validity of the user study.

Study procedure

First, the participants were randomly divided into control or treatment groups; 29 participants were in the control group, and 27 were in the treatment group. The survey presents a sample from the UCI Adult dataset [Kohavi and Becker, 2026] and explains a DP mechanism is used to answer queries on the dataset by adding noise controlled by the parameter ϵ. Then, it presents two queries in random order:

- The number of people with income greater than 50K who are less than 40 years old.
- The number of people who have a Master's degree or above and have income $\leq$ 50k, grouped by each race.

For each query, the survey asks the participant to choose ϵ from a slider, with values from 0.5 to 5.0 in an interval of 0.5. Since we want all participants to have a clear task goal, the survey includes a description of the task: choose ϵ to protect the within-dataset individuals' privacy while yielding as accurate answers as possible.

For the control group, the participant was presented with a table showing the range of possible outputs for each candidate ϵ (and the raw output when no DP is applied) to indicate the resulting accuracy of each ϵ. The table includes information that any controller can access in a DP deployment today when choosing ϵ. For the treatment group, the participant was presented with the same table, along with the additional information: a plot of the range of within-dataset individuals' RDRs for each candidate ϵ, i.e., a box drawn from the minimum

to the maximum RDR. An explanation of the RDR was provided to the participants. We also conducted another study in which we recruited a different treatment group and showed the participants RDR histograms instead of range plots. The additional information of RDR is the only difference between the control and treatment groups to ensure we isolate the intervention. Participants in the treatment group were also asked a follow-up question about whether they used the within-dataset individuals' RDRs when choosing ϵ; we included it to ensure that if we observe an effect, we can attribute the effect to the use of RDRs.

Remarks. An alternative way to design the intervention for the treatment group is to present participants with the raw RDRs for each candidate ϵ (along with the per-instance sensitivity of within-dataset individuals), then ask participants to explicitly define a preference function in terms of the RDRs and a threshold to choose ϵ, i.e., by running the `Find-`ϵ`-from-RDR` algorithm. However, doing so would require recruiting participants who have real-world experiences as data controllers managing private datasets about individuals to devise a privacy preference, who have sufficient quantitative background to examine the per-instance sensitivity and RDR in raw numbers and translate their preference into a preference function, and who have more than a surface-level understanding of DP. In contrast, the participants we recruited come from various backgrounds, and we cannot make those assumptions about them. Instead, our study evaluates whether the visual presentation of RDRs, through range plots or histograms, enables participants to form *implicit* preferences about the risk distribution of the within-dataset individuals and choose ϵ based on their preferences.

Results

If the intervention (showing the range plots of within-dataset individuals' RDRs) is effective, we expect this would manifest as more consistent answers among participants, since all participants were given the same training and the same task. Therefore, we compare the

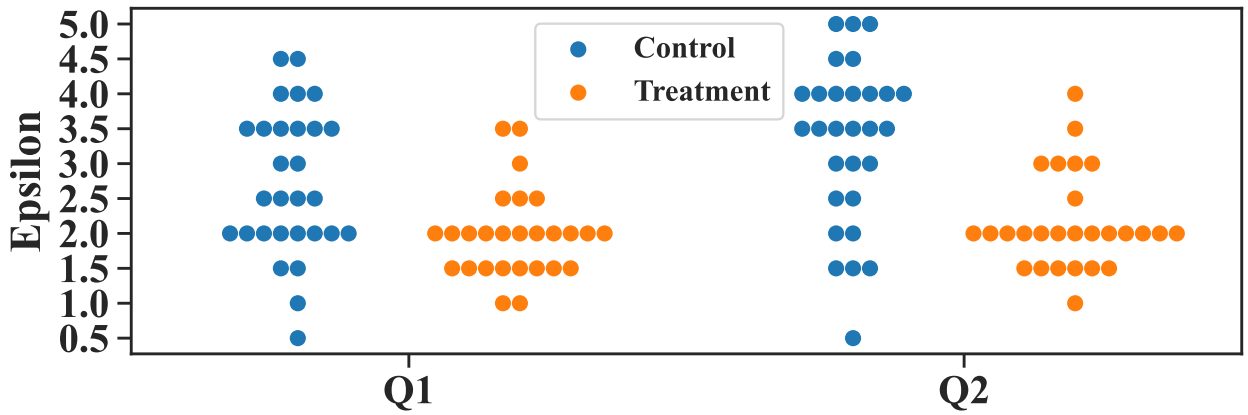

Figure 4.4: ϵ chosen by each participant

distribution of ϵ chosen by participants in the control and treatment groups. Figure 4.4 shows a swarm plot where each point represents the ϵ chosen by each participant. For both queries, 19 out of 27 participants in the treatment group chose either 2 or 1.5, showing a more consistent answer than the control group. This is because the range of the within-dataset individuals' RDRs when ϵ is 2 or 1.5 is considerably smaller than that of larger ϵ, and participants know that smaller ϵ has lower accuracy. This shows that most participants could connect the within-dataset individuals' RDRs and their task goal, and they were able to use the RDRs to choose an ϵ that satisfies the goal. On the contrary, the ϵ chosen by participants in the control group are more spread out and have greater variance, suggesting participants shared no general consensus as to what is a good ϵ to choose. In addition, we ran Mann–Whitney test [McKnight and Najab, 2010] on the ϵ chosen from the control and treatment group, with the alternative hypothesis that the two samples come from different distributions. For both queries, we obtained $p < 0.005$. Thus we reject the null hypothesis and conclude that showing the within-dataset individuals' RDRs helped participants choose ϵ.

Lastly, we asked participants in the treatment group if they used the within-dataset individuals RDRs to complete the task: all 27 participants responded Yes, confirming they were indeed "treated".

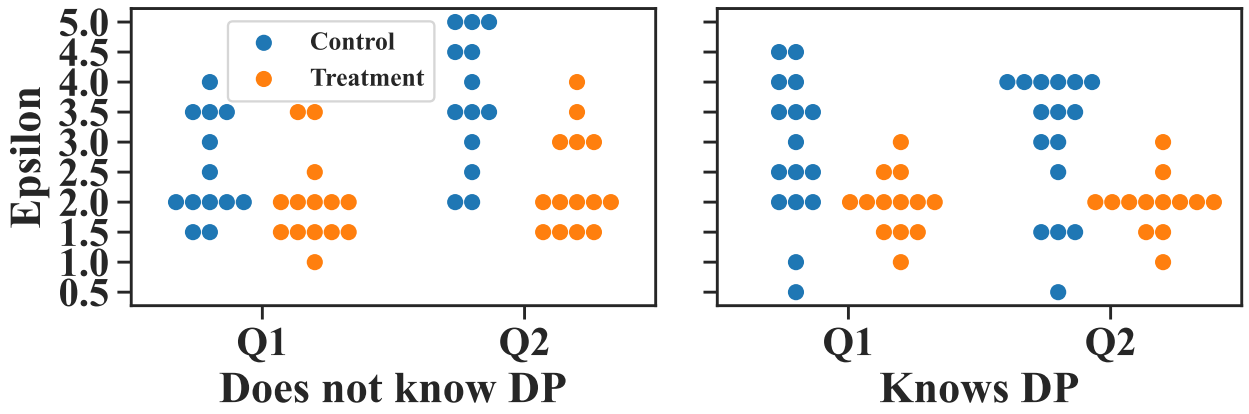

Figure 4.5: ϵ chosen by participants who do not know DP (left) and those who know DP (right)

Does knowing DP matter? Another interesting question is whether the participant's prior familiarity with DP affects how they choose ϵ. At the start of the survey before showing the tutorial, we asked each participant whether they know about DP. 27 participants (out of a total of 56) answered they do not know anything about DP and 29 participants answered they know about DP. Figure 4.5 shows the ϵ chosen by participants based on their familiarity of DP. We find similar patterns. Participants in the treatment group tend to choose either 2 or 1.5, regardless of their familiarity with DP, whereas participants in the control group chose ϵ with greater variance. This suggests participants' prior knowledge of DP does not transfer to making better choices of ϵ; on the other hand, showing the within-dataset individuals RDRs helped participants make more consistent choices whether or not they knew about DP previously.

Range plot vs histogram? To analyze the effect of different RDR visualizations, we recruited another group of participants - 20 of them passed the quiz, and we asked them to complete the same survey questions as in the previous treatment group, except they were presented with histograms of RDRs instead of range plots.

Figure 4.6 shows ϵ chosen by participants who were presented with histograms or range plots; we include the ϵ chosen by participants in the previous control group as reference.

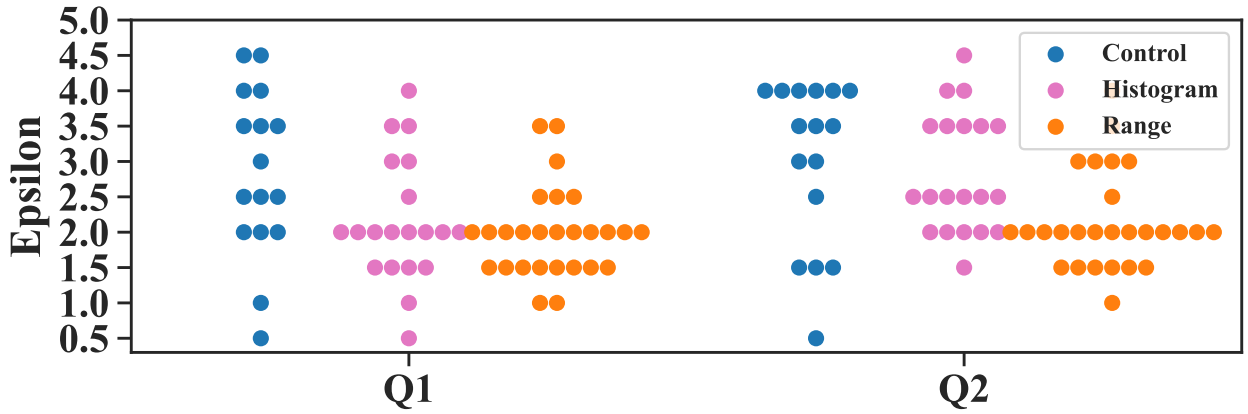

Figure 4.6: ϵ chosen by participants who were presented with histograms or range plots, or in the control group

Participants who saw histograms did not perform better than participants who saw range plots. Especially for Q2, eight participants chose larger ϵ (3.5, 4, and 4.5). Since the responses are anonymous, we cannot directly ask participants about their decision-making after the survey ends. We deduce one possible explanation might be that for the large ϵ, the majority of RDRs are already small and similar, and only a small portion of RDRs have large values. The participants might think that as long as the privacy of the majority of people is being protected, the task goal is achieved; hence, they chose larger ϵ, so the query outputs are more accurate. However, if the participants only saw the range of RDR, they would be more inclined to choose smaller ϵ with a smaller range of RDR. Another possible explanation could be that some participants had difficulties interpreting histograms, whereas the range plots are generally much easier to understand.

The discrepancy of results by tweaking how we present the RDRs to the participants demonstrates that the way we convey information affects their implicit privacy preferences on the within-dataset individuals, which in term affects their decision-making for choosing ϵ. Exploring other tools for presenting and helping data controllers interpret RDR remains an interesting problem.

Threats to validity

External validity: We recruited participants from diverse backgrounds worldwide via Prolific, with few prescreen criteria so that the participants could complete the study without difficulty. **Internal validity:** To ensure participants have the minimum knowledge of DP needed to complete the study, each participant must read a tutorial and pass a quiz. Participants can only take the quiz once to minimize the chance of them abusing it if they have multiple chances. To avoid learning effects, we used a between-subject study to isolate the condition we intended to test. We also randomized the two questions for choosing ϵ.

4.6.2 RQ2: Are Both Algorithms Practical?

Both `Find-`ϵ`-from-RDR` and `Find-and-release-`ϵ`-from-RDR` need to be efficient so that it is practical for controllers to use them to find ϵ for various queries. Thus, we evaluate both algorithms' runtime for different queries and dataset sizes.

Datasets. We controlled the experiment by generating datasets with 1000, 10000, 100000, and 1 million records from the Adult dataset [Kohavi and Becker, 2026]. We sampled records for datasets of sizes (number of records) 1000 and 10000. For datasets of size 100000 and 1 million, we replicated and added random noise to the replicated records to ensure that they were not the same copy of each other. We included all 15 attributes in the dataset.

Queries. We ran both algorithms for the five SQL queries below. We chose a diverse set of queries with attributes of different domain sizes and types. The queries are run independently.

- Q1: `SELECT COUNT(*) FROM adult WHERE income=='>50K' AND education_num==13 AND age==25`
- Q2: `SELECT marital_status, COUNT(*) FROM adult WHERE race=='Asian-Pac-Islander' AND 30<=age<=40 GROUP BY marital_status`

- Q3: `SELECT COUNT(*) FROM adult WHERE native_country!='United-States' AND sex =='Female'`
- Q4: `SELECT AVG(hours_per_week) FROM adult WHERE workclass in ('Federal-gov', 'Local-gov', 'State-gov')`
- Q5: `SELECT SUM(capital_gain) FROM adult`

Parameters. We initialize candidate ϵ values in fixed intervals; specifically, we test ϵ in $[10, \cdots, 1, 0.9, \cdots, 0.1, 0.09, \cdots, 0.01, 0.099, \cdots, 0.001]$ (37 values in total). We use RDR_{min}/RDR_{max} as the preference function and set $\tau_p = 0.95$ in `Find-`ϵ`-from-RDR`, and we set $\epsilon_{svt} = 1, \tau_{var} = 10^{-5}$ in `Find-and-release-`ϵ`-from-RDR`. We used Laplace Mechanism as the underlying DP mechanism $\mathcal{M}$. We used 8 parallel processes to compute the per-instance sensitivity of each within-dataset individual.

Configurations. We ran the experiments on a Chameleon Cloud [Keahey et al., 2020] instance (Intel Xeon Gold 6242 CPU) with 32 cores and 188GB memory. We used Ubuntu 24.04 and Python 3.12 for all experiments.

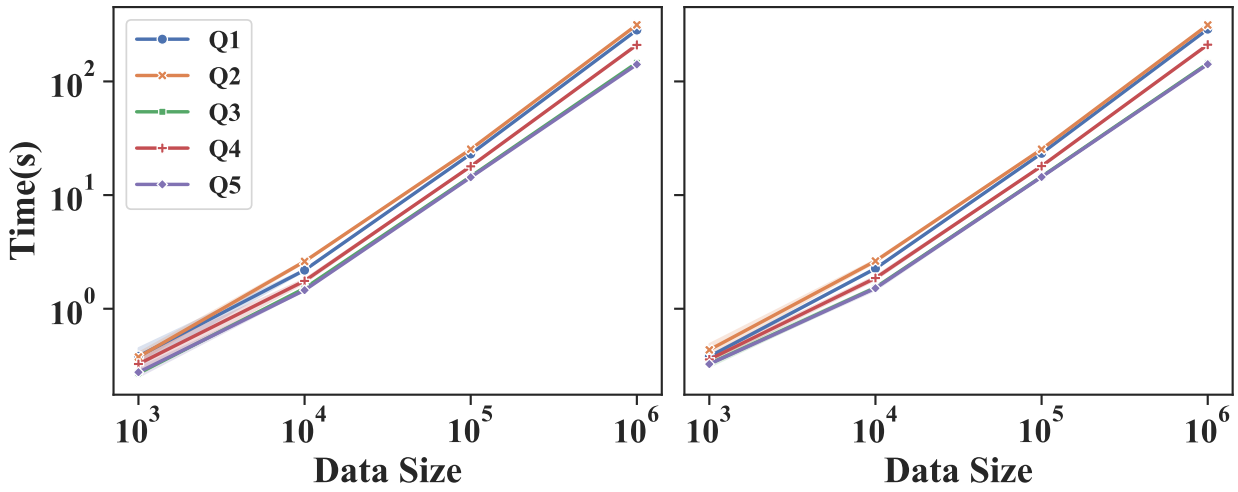

Figure 4.7: Average runtime (in seconds) over 10 runs to run each query. Left is `Find-`ϵ`-from-RDR` and right is `Find-and-release-`ϵ`-from-RDR`

Results. Figure 4.7 shows two line plots where the x-axis is the data size and the y-

axis is the runtime in seconds; both axes are in log scale. The left plot corresponds to the run time of `Find-`ϵ`-from-RDR`, and the right plot corresponds to the run time of `Find-and-release-`ϵ`-from-RDR`. Each line in the plot represents a query, with the error band showing the standard deviation. As shown in both plots, the runtime of both algorithms increases linearly with the data size, and the difference between the two algorithms is negligible. All queries took less than a few minutes to finish; the longest runtime was 5 minutes by Q2 with 1 million records. The runtime is dominated by the time to compute the per-instance sensitivity of each within-dataset individual since this involves first computing the query output on each x_{-i} (the dataset without i's record).

4.6.3 Microbenchmarks

Effect of τ_p in `Find-`ϵ`-from-RDR`

Given the preference function RDR_{min}/RDR_{max}, we test the effect of varying threshold τ_p and report the ϵ found.

Setup. We used the original Adult dataset with 48842 rows, and the same five queries and candidate ϵ values in Section 4.6.2. We run the algorithm by varying $\tau_p = 0.95, 0.75, 0.5, 0.25, 0.05$ and the DP mechanism $\mathcal{M}$ as Laplace Mechanism or Gaussian Mechanism.

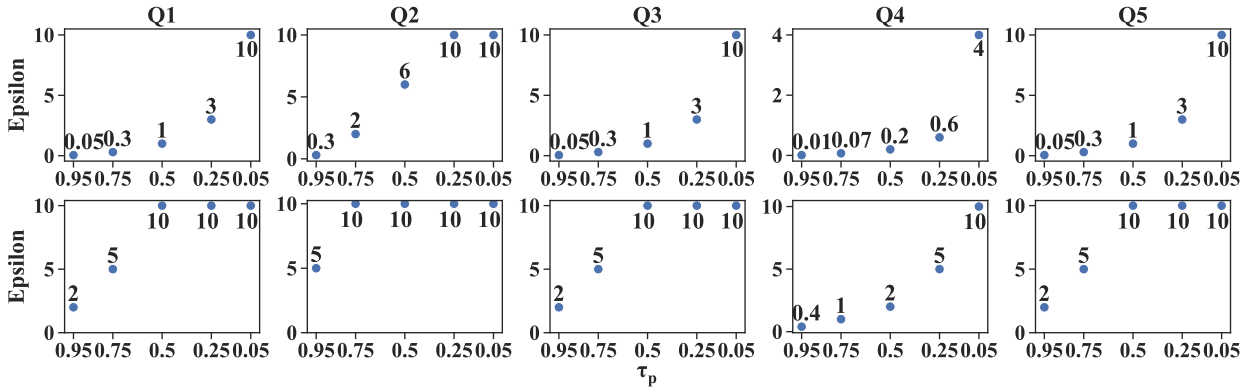

Figure 4.8: ϵ found by `Find-`ϵ`-from-RDR` under Laplace Mechanism (top) and Gaussian Mechanism (bottom)

Results. Figure 4.8 shows the ϵ found by `Find-`ϵ`-from-RDR` under Laplace Mechanism and Gaussian Mechanism respectively. As shown in both figures, smaller τ_p leads to larger ϵ, as expected. Using Gaussian Mechanism as the underlying mechanism to compute RDR caused the algorithm to find larger ϵ compared to using Laplace Mechanism, because with the same ϵ, Gaussian Mechanism is more likely to produce noisier output than the Laplace Mechanism and RDRs that are closer. Overall, the results empirically demonstrate that the algorithm is useful for finding a suitable ϵ consistent with the controller's privacy preferences.

Effect of τ_{var} in `Find-and-release-`ϵ`-from-RDR`

Due to the randomness of SVT, we evaluate the effect of varying τ_{var} by running the algorithm 50 times and report ϵ found in each round.

Setup. We use the same dataset, queries, and candidate ϵ values in Section 4.6.3. We fix $\epsilon_{svt} = 1$. We run the algorithm by varying $\tau_{var} = 10^{-11}, 10^{-9}, 10^{-7}, 10^{-5}$ and the DP mechanism $\mathcal{M}$ as Laplace Mechanism or Gaussian Mechanism.

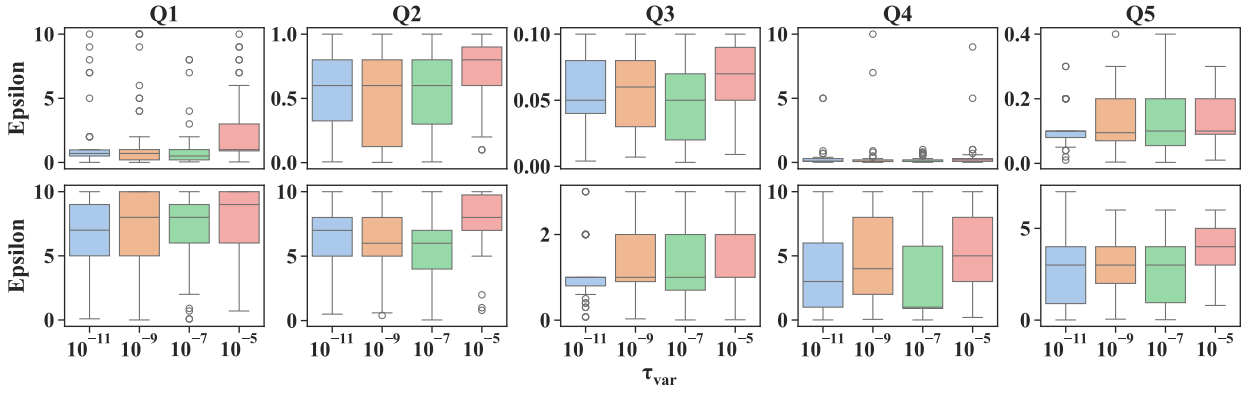

Figure 4.9: ϵ found by `Find-and-release-`ϵ`-from-RDR` under Laplace Mechanism (top) and Gaussian Mechanism (bottom)

Results. Figure 4.9 shows the boxplots of ϵ found by `Find-and-release-`ϵ`-from-RDR` under Laplace Mechanism and Gaussian Mechanism respectively. In both figures, for each query, the ranges of ϵ found are mostly similar for different τ_{var} except in some cases when $\tau_{var} =$

10^{-5} and larger ϵ are found. In general, controllers should still observe the per-instance sensitivity of within-dataset individuals to choose an appropriate τ_{var}. And similarly to what we observed when running `Find-`ϵ`-from-RDR`, the algorithm tends to find larger ϵ using Gaussian Mechanism compared to using Laplace Mechanism. Because of the randomness introduced by SVT, the spread of ϵ found by the algorithm can cover the entire range of candidate ϵ values, thus controllers may want to rerun the algorithm multiple times (or with different τ_{var} and ϵ_{svt}) to observe the effect of SVT and decide which ϵ to release.

4.7 Related Work

Privacy loss accounting. Various methods have been proposed to account for privacy loss in different granularities. Ex-post per-instance DP [Redberg and Wang, 2021] is built upon Ex-post DP [Ligett et al., 2017], which depends on the realized output of a DP mechanism, and Per-instance DP [Wang, 2019], which depends on a fixed dataset and a specific record. While ex-post metrics accurately quantify realized privacy loss, they are output-dependent and therefore difficult for controllers to use when comparing disclosure risks before a query is run. Conversely, per-instance sensitivity [Wang, 2019] offers an output-independent measure of an individual's baseline distinguishability, but alone it offers no framework to compare risks and choose ϵ. RDR bridges this gap by combining per-instance sensitivity with the expected noise of a DP mechanism to serve as a practical, relative indicator of disclosure risk that controllers can use to express privacy preferences and choose ϵ. Feldman and Zrnic [Feldman and Zrnic, 2021] proposed a privacy filter under Renyi Differential Privacy by tracking the personalized privacy loss for each individual. Zhu et al. [Zhu et al., 2022] proposed a unified approach for privacy accounting via a characteristic function. Seeman et al. [Seeman et al., 2024] proposed Per Record Differential Privacy, in which only the policy function on each individual's privacy loss is published but not the actual privacy loss value, and individuals can compute their own privacy loss using the function. Unlike the

aforementioned works where the main goal is to provide tighter privacy accounting than standard DP, *RDR is designed to help controllers understand the privacy implications on the within-dataset individuals when choosing ϵ in standard DP.*

Choosing privacy loss budget. One line of work [Murtagh et al., 2018, Nanayakkara et al., 2022, Hay et al., 2016, Thaker et al., 2020, John et al., 2021, Nanayakkara et al., 2023] implements interfaces to visualize the effect of different variables (e.g., utility metrics) in relation to ϵ. John et al. [John et al., 2021] defines and visualizes the data sharing risk. Both metrics quantify potential data leakage, but neither metric applies to each within-dataset individual, unlike our RDR. Ligett et al. [Ligett et al., 2017] empirically finds the smallest ϵ consistent with an accuracy requirement in the private empirical risk minimization setting. Ge et al. [Ge et al., 2019] translates a user-specified accuracy bound to a suitable DP mechanism, along with an ϵ that incurs the least privacy loss while satisfying the accuracy bound. Hsu et al. [Hsu et al., 2014] proposes an economic model for choosing ϵ by balancing the interests of analysts and prospective participants who will participate if the benefit outweighs the risk. However, this model assumes participants can quantitatively express the "cost" of whether to participate in the study, which is impractical for non-experts. Kohli and Laskowski [Kohli and Laskowski, 2018] envisions a voting system where individuals submit their preferences over ϵ. Cummings et al. [Cummings et al., 2021] and Xiong et al. [Xiong et al., 2020] investigate users' understanding of DP guarantees through user studies. To the best of our knowledge, *no known approach helps controllers understand the effect of ϵ on each within-dataset individual's privacy and uses the information to design an algorithm that finds ϵ automatically, as we do.*

Customizing Privacy. Variants of DP have been proposed that allow individuals to specify their own privacy preferences. Jorgensen et al. [Jorgensen et al., 2015] proposes Personalized Differential Privacy, in which each within-dataset individual specifies their own privacy preference value instead of using a global privacy loss budget. He et al. [He et al., 2014]

proposes Blowfish Privacy, in which data publishers specify their privacy policies in addition to specifying ϵ. Both works extend the standard differential privacy framework and require individuals to define their privacy preferences. Instead, *RDR is only used by data controllers, who are responsible for protecting individuals' data and they can now express their privacy preferences in terms of RDRs.*

4.8 Conclusion

In this chapter, we showed that by presenting additional information (the RDR) on the relative disclosure risk of the within-dataset individuals when applying DP, controllers gain more understanding of the effect of choosing ϵ. We leverage the RDR to design an algorithm, `Find-`ϵ`-from-RDR`, which finds ϵ based on controllers' privacy preferences in terms of the RDRs of the within-dataset individuals. In addition, we propose an alternative algorithm, called `Find-and-release-`ϵ`-from-RDR`, which finds and releases ϵ privately. We present our solution to bound total privacy leakage when answering multiple queries without requiring controllers to set a total privacy budget. Overall, our work reduces the burden for controllers to choose ϵ in practice.

CHAPTER 5
ORTHOGONAL AND EXTENDED SYSTEMS FOR ARTICULATING AND ENFORCING DATAFLOW PREFERENCES

This chapter presents two other systems that address the practical challenges of articulating and enforcing dataflow preferences. *Ver* [Gong et al., 2023] is an orthogonal contribution that tackles the problem of view discovery, enabling agents to locate and combine the data required to formulate a dataflow. *Data Station* [Xia et al., 2022] extends the data escrow architecture to support data-sharing consortia, providing the technical infrastructure to enforce organizations' dataflow preferences.

5.1 Ver: View Discovery in the Wild

To articulate a dataflow preference, an agent must first identify the relevant data. In large, decentralized repositories, such as data lakes or open data portals, this identification process is non-trivial. Data is often scattered across disparate tables without explicit schema or join information, creating what we term pathless table collections. A pathless table collection contains noisy, incomplete tables that lack predefined join paths. Consequently, when an agent attempts to combine datasets for a downstream task, they face semantic ambiguity, noisy data, and an overwhelmingly large search space of potential joins.

Ver is an end-to-end data discovery system designed to identify project-join views over these pathless table collections. It allows agents to provide examples of the data they need via a Query-by-Example (QBE) interface. Ver processes these examples through a reference architecture that tackles both the technical challenge of searching millions of join paths and the human challenge of navigating ambiguous results. It utilizes a view distillation component to categorize and filter redundant views automatically. To address semantic ambiguity,

Ver employs a bandit-based view presentation component that asks agents targeted questions, adapting to their evolving knowledge to rapidly narrow down the search space to the exact view they require.

Ver is orthogonal to the core dataflow preferences framework. While the dataflow preferences framework governs how and why data moves between agents, Ver addresses the independent problem of what data exists and how it can be meaningfully assembled in the wild. Before an agent can specify a dataflow preference, they must discover and define the data source. By solving the discovery problem in pathless environments, Ver provides the necessary tooling to locate the data that the dataflow preferences framework subsequently governs.

5.2 Data Station: Delegated, Trustworthy, and Auditable Computation to Enable Data-Sharing Consortia with a Data Escrow

While Chapter 3 demonstrated a bolt-on data escrow for the application ecosystem, enforcing dataflow preferences at the organizational level introduces different challenges. Organizations often wish to pool their data to train machine learning models or extract collaborative insights, but they are prevented from doing so by regulatory, legal, and privacy barriers.

Data Station is a data escrow system designed to overcome these barriers and enable the formation of data-sharing consortia. It employs a delegated computation model where data owners share their data with the escrow, and data users delegate their computations to the escrow. The escrow executes the users' queries without releasing the raw data. To ensure trustworthy computation, Data Station leverages secure hardware enclaves, which encrypt data in memory during processing and provide remote attestation to prove software integrity. Furthermore, Data Station provides auditable computation by maintaining a tamper-proof,

immutable log of all data access and computation, ensuring compliance with organizational and regulatory rules.

Data Station directly extends the thesis by scaling the technical intervention of a data escrow to cross-organizational environments. It provides the concrete infrastructure needed to enforce the dataflow preferences of multiple organizations simultaneously. By guaranteeing that data will not be released without explicit consent and by making the computational purpose explicit and auditable, Data Station operationalizes the dataflow-centric paradigm.

5.3 Conclusion

This chapter outlined *Ver* and *Data Station*, two systems that expand the practical applicability of this dissertation's main thesis. *Ver* orthogonally solves the data discovery problem in pathless table collections, ensuring agents can identify the data they need to formulate their dataflows. *Data Station* provides a trustworthy, auditable escrow infrastructure to enforce complex dataflow preferences across organizational boundaries.

CHAPTER 6

CONCLUSION

This dissertation argues that articulating and addressing data problems requires a fundamental shift from a data-centric to a dataflow-centric paradigm. This shift enables agents to assert granular, enforceable control over their data in motion. By establishing the dataflow preferences framework, we elevate the computational purpose of data use to a first-class primitive and provide a precise vocabulary to articulate data problems, as demonstrated through the dataflow analysis of the SafeInsights ecosystem. Building upon this framework, we designed and deployed concrete technical interventions. These include a bolt-on data escrow architecture for the application ecosystem and solutions to enable practical deployment of differential privacy.

Establishing the dataflow preferences framework and its initial technical interventions opens several avenues for future research. First, while we proposed the technical interventions for enforcing dataflow preferences, navigating how agents practically define and interact with these preferences remains an open challenge. Future work may explore tools to help agents practically define and interact with their dataflow preferences. Second, as identified in the SafeInsights analysis, enforcing that the output of an analysis does not violate data preferences (e.g., ensuring aggregate outputs do not leak Personally Identifiable Information) currently relies heavily on manual review. Future research may explore automated, scalable auditing tools, such as static code analysis or verifiable computation techniques, to verify that an untrusted party's proposed computation strictly aligns with the established dataflow preferences before execution. Third, future work may explore other interventions to enforce dataflow preferences across emerging architectures and broader ecosystems, such as Large Language Models (LLMs).

APPENDIX: WITHIN-DATASET DISCLOSURE RISK FOR DIFFERENTIAL PRIVACY

A.1 Derivation of RDR Upper Bounds

Claim: RDR for Laplace Mechanism satisfies

$$RDR_i \geq \mathbb{E}_{o\sim\mathcal{M}_{Lap}(x,q,\epsilon)}[||o - q(x_{-i})||_1].$$

Proof. Let $o = q(x) + Z$, where $Z = (Z_1, \ldots, Z_k)$ and $Z_j \sim Lap(b)$ are i.i.d. with $b = \Delta_1/\epsilon$. Let $C_i = q(x) - q(x_{-i})$. Then $\mathbb{E}_{o\sim\mathcal{M}_{Lap}(x,q,\epsilon)}\left[\|o - q(x_{-i})\|_1\right] = \mathbb{E}[\|C_i + Z\|_1]$.

By the definition of the ℓ_1 norm and the linearity of expectation: $\mathbb{E}[\|C_i + Z\|_1] = \mathbb{E}\left[\sum_{j=1}^{k} |(C_i)_j + Z_j|\right] = \sum_{j=1}^{k} \mathbb{E}[|(C_i)_j + Z_j|]$.

By the triangle inequality: $\sum_{j=1}^{k} \mathbb{E}[|(C_i)_j + Z_j|] \leq \sum_{j=1}^{k} \mathbb{E}[|(C_i)_j| + |Z_j|]$.

Since $(C_i)_j$ is a constant for each dimension j, this simplifies to: $\sum_{j=1}^{k} \mathbb{E}[|(C_i)_j| + |Z_j|] = \left(\sum_{j=1}^{k} |(C_i)_j|\right) + \left(\sum_{j=1}^{k} \mathbb{E}[|Z_j|]\right) = \|C_i\|_1 + \left(\sum_{j=1}^{k} \mathbb{E}[|Z_j|]\right)$.

Since $Z_j \sim Lap(b)$, $\mathbb{E}[|Z_j|] = b = \Delta_1/\epsilon$. Since all Z_j are i.i.d., $\sum_{j=1}^{k} \mathbb{E}[|Z_j|] = \sum_{j=1}^{k} k\Delta_1/\epsilon$.

Combining the terms and substituting back $C_i = q(x) - q(x_{-i})$, we get the upper bound: $RDR_i = \|q(x) - q(x_{-i})\|_1 + k\Delta_1/\epsilon \geq \mathbb{E}_{o\sim\mathcal{M}_{Lap}(x,q,\epsilon)}[||o - q(x_{-i})||_1]$.

This completes the proof.

□

Claim: RDR for Gaussian Mechanism satisfies

$$RDR_i \geq \mathbb{E}_{o\sim\mathcal{M}_{Gau}(x,q,\epsilon)}[||o - q(x_{-i})||_2].$$

Proof. Let $o = q(x) + Z$, where $Z = (Z_1, \ldots, Z_k)$ and $Z_j \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. with $\sigma^2 = \frac{2\Delta_2^2 \ln(1.25/\delta)}{\epsilon^2}$. Let $C_i = q(x) - q(x_{-i})$. Then $\mathbb{E}_{o\sim\mathcal{M}_{Gau}(x,q,\epsilon,\delta)}[\|o - q(x_{-i})\|_2] = \mathbb{E}[\|C_i + Z\|_2]$.

By Jensen's inequality, $(\mathbb{E}[\|C_i + Z\|_2])^2 \leq \mathbb{E}[\|C_i + Z\|_2^2]$.

By linearity of expectation: $\mathbb{E}[\|C_i + Z\|_2^2] = \mathbb{E}\left[\sum_{j=1}^{k} ((C_i)_j + Z_j)^2\right] = \sum_{j=1}^{k} \mathbb{E}[((C_i)_j + Z_j)^2]$.

Expanding the squared term: $\sum_{j=1}^{k} \mathbb{E}[(C_i)_j^2 + 2(C_i)_j Z_j + Z_j^2] = \sum_{j=1}^{k} (\mathbb{E}[(C_i)_j^2] + 2(C_i)_j \mathbb{E}[Z_j] + \mathbb{E}[Z_j^2])$.

Since $(C_i)_j$ is a constant for each dimension j, $\mathbb{E}[(C_i)_j^2] = (C_i)_j^2$.

Since each $Z_j \sim \mathcal{N}(0, \sigma^2)$, $\mathbb{E}[Z_j] = 0$ and the variance $Var(Z_j) = \mathbb{E}[Z_j^2] - (\mathbb{E}[Z_j])^2 = \sigma^2$, thus $\mathbb{E}[Z_j^2] = \sigma^2$.

Substituting these values back, we get: $\mathbb{E}[\|C_i + Z\|_2] = \sum_{j=1}^{k} ((C_i)_j^2 + 2(C_i)_j \cdot 0 + \sigma^2) = \sum_{j=1}^{k} \left((C_i)_j^2\right) + \sum_{j=1}^{k} \left(\sigma^2\right) = \|C_i\|_2^2 + k\sigma^2$.

We have thus shown that: $(\mathbb{E}[\|C_i + Z\|_2])^2 \leq \|C_i\|_2^2 + k\sigma^2$.

Taking the square root of both sides and substituting back $C_i = q(x) - q(x_{-i})$, we get the upper bound: $RDR_i = \sqrt{\|q(x) - q(x_{-i})\|_2^2 + k\sigma^2} \geq \mathbb{E}_{o\sim\mathcal{M}_{Gau}(x,q,\epsilon)}[||o - q(x_{-i})||_2]$.

This completes the proof.

□

A.2 Accuracy Analysis of `Find-and-release-`ϵ`-from-RDR` algorithm

Claim: Assume the SVT budget ϵ_{svt} budget is split such that $\epsilon'_{svt} = \epsilon_{svt}/3$ and $\epsilon'_{svt} = 2\epsilon_{svt}/3$. For any failure probability $\beta \in (0, 1)$, `Find-and-release-`ϵ`-from-RDR` algorithm is (α, β)-accurate, where $\alpha = \frac{6}{n\epsilon_{svt}} \ln(\frac{2|\mathcal{E}|}{\beta})$; the accuracy of the algorithm refers to the correctness of the chosen ϵ. This means that with probability at least $1 - \beta$, if the algorithm returns a value ϵ^*, then the true variance for that value satisfies $q_{svt}(\epsilon) \leq \tau_{var} + \alpha$. And for any candidate $\epsilon' \in \mathcal{E}$ such that $q_{svt}(\epsilon') < \tau_{var} - \alpha$, the algorithm will not halt at a value

$\epsilon < \epsilon'$, thus returning a value at least as large as ϵ'(since candidate ϵ values are tested in descending order).

Proof. The SVT mechanism in `Find-and-release-`ϵ`-from-RDR` algorithm checks if $-q_{svt}(\epsilon) + Lap\left(\frac{2\Delta_{svt}}{\epsilon''_{svt}}\right) \geq -\tau_{var} + \rho$, where $\rho = Lap\left(\frac{\Delta_{svt}}{\epsilon'_{svt}}\right)$. Let $\nu = Lap\left(\frac{2\Delta_{svt}}{\epsilon''_{svt}}\right)$. Since we assume a budget split of $\epsilon'_{svt} = \epsilon_{svt}/3$ and $\epsilon'_{svt} = 2\epsilon_{svt}/3$, we have $\nu \sim Lap(3\Delta_{svt}/\epsilon_{svt})$ and $\rho \sim Lap(3\Delta_{svt}/\epsilon_{svt})$; let $b = 3\Delta_{svt}/\epsilon_{svt}$.

Let the total noise on the comparison be $Z = \nu - \rho$. An error occurs if the noise flips the outcome of the comparison, which happens if $|Z| > |q_{svt}(\epsilon) - \tau_{var}|$. We bound the probability that $|Z| > \alpha$ for any of the $\mathcal{E}$ queries.

Using a standard tail bound for the difference of two i.i.d. Laplace variables, ν, ρ: $\Pr[|Z| > \alpha] \leq 2\exp(-\frac{\alpha}{2b})$.

Then we apply a union bound over all $\mathcal{E}$ candidate ϵ values. The probability that the noise for any query exceeds α is at most: $\Pr[\exists \epsilon \in \mathcal{E}, |Z_\epsilon| > \alpha] \leq |\mathcal{E}| \cdot 2\exp(-\frac{\alpha}{2b})$.

We set this failure probability to be at most β: $|\mathcal{E}| \cdot 2\exp(-\frac{\alpha}{2b}) \leq \beta \Rightarrow \alpha \geq 2b\ln(\frac{2|\mathcal{E}|}{\beta})$.

Substituting $b = 3\Delta_{svt}/\epsilon_{svt}$ and $\Delta_{svt} = 1/n$: $\alpha \geq 2(\frac{3(1/n)}{\epsilon_{svt}})(\ln(\frac{2|\mathcal{E}|}{\beta})) = \frac{6}{n\epsilon_{svt}}\ln(\frac{2|\mathcal{E}|}{\beta})$.

This establishes the (α, β)-accuracy guarantee. With probability at least $1 - \beta$, for all $|\mathcal{E}|$ queries, the noisy comparison is within α of the true comparison.

□

REFERENCES

Prolific, 2026. URL https://www.prolific.co.

John M Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2867–2867, 2018.

Rene Abraham, Johannes Schneider, and Jan Vom Brocke. Data governance: A conceptual framework, structured review, and research agenda. *International journal of information management*, 49:424–438, 2019.

Alessandro Acquisti, Idris Adjerid, Rebecca Balebako, Laura Brandimarte, Lorrie Faith Cranor, Saranga Komanduri, Pedro Giovanni Leon, Norman Sadeh, Florian Schaub, Manya Sleeper, et al. Nudges for privacy and security: Understanding and assisting users' choices online. *ACM Computing Surveys (CSUR)*, 50(3):1–41, 2017.

Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 143–154. Elsevier, 2002.

Hazim Almuhimedi, Florian Schaub, Norman Sadeh, Idris Adjerid, Alessandro Acquisti, Joshua Gluck, Lorrie Faith Cranor, and Yuvraj Agarwal. Your location has been shared 5,398 times! a field study on mobile app privacy nudging. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 787–796, 2015.

Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words:{Entity-Sensitive} privacy policy and data flow analysis with {PoliCheck}. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 985–1002, 2020.

Gregory R Andrews and Richard P Reitman. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2(1): 56–76, 1980.

Apple. Private cloud compute: A new frontier for ai privacy in the cloud, 2024. URL https://security.apple.com/blog/private-cloud-compute/.

Apple. App review guidelines, 2025a. URL https://developer.apple.com/app-store/review/guidelines/.

Apple. Entitlements, 2025b. URL https://developer.apple.com/documentation/bundleresources/entitlements.

Apple. Information property list, 2025c. URL https://developer.apple.com/documentation/bundleresources/information_property_list.

Apple. Cncontactfetchrequest, 2025d. URL https://developer.apple.com/documentation/contacts/cncontactfetchrequest.

Apple. Phasset, 2025e. URL https://developer.apple.com/documentation/photokit/phasset.

Apple. Photokit, 2025f. URL https://developer.apple.com/documentation/photokit.

Apple. Contacts framework, 2025g. URL https://developer.apple.com/documentation/contacts.

Apple. Core location framework, 2025h. URL https://developer.apple.com/documentation/corelocation.

Apple. Swift healthkit, 2025i. URL https://developer.apple.com/documentation/healthkit/.

Apple and Google. Privacy-preserving contact tracing, 2025. URL https://covid19.apple.com/contacttracing.

Imanol Arrieta-Ibarra, Leonard Goff, Diego Jiménez-Hernández, Jaron Lanier, and E Glen Weyl. Should we treat data as labor? moving beyond "free". In *aea Papers and Proceedings*, volume 108, pages 38–42. American Economic Association 2014 Broadway, Suite 305, Nashville, TN 37203, 2018.

D Elliott Bell and Leonard J La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, 1976.

Bluesky. Bluesky social app, 2025. URL https://github.com/bluesky-social/social-app.

Carolyn Brodie, Clare-Marie Karat, John Karat, and Jinjuan Feng. Usable security and privacy: a case study of developing privacy management tools. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 35–43, 2005.

Duc Bui, Yuan Yao, Kang G Shin, Jong-Min Choi, and Junbum Shin. Consistency analysis of data-usage purposes in mobile apps. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2824–2843, 2021.

Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of cryptography conference*, pages 635–658. Springer, 2016.

US Census Bureau. Census bureau sets key parameters to protect privacy in 2020 census results. https://www.census.gov/newsroom/press-releases/2021/2020-census-key-parameters.html, 2021.

Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *The VLDB Journal*, 17(4):603–619, 2008.

Canadian Digital Service. Covid alert mobile app, 2025. URL `https://github.com/cds-snc/covid-alert-app`.

Michael J Carey, Laura M Haas, Peter M Schwarz, Manish Arya, William F Cody, Ronald Fagin, Myron Flickner, Allen W Luniewski, Wayne Niblack, Dragutin Petkovic, et al. Towards heterogeneous multimedia information systems: The garlic approach. In *Proceedings RIDE-DOM'95. Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management*, pages 124–131. IEEE, 1995.

Raul Castro Fernandez. What is the value of data? a theory and systematization. *ACM/IMS Journal of Data Science*, 2(1):1–25, 2025.

Andreas Claesson and Tor E Bjørstad. Technical report:'out of control'—a review of data sharing by popular mobile apps. *Report for the Norwegian Consumer Council*, 14, 2020.

Kevin Collier and Minyvonne Burke. Facebook turned over chat messages between mother and daughter now charged over abortion, 2022.

Matthew Crain. The limits of transparency: Data brokers and commodification. *new media & society*, 20(1):88–104, 2018.

Rachel Cummings, Gabriel Kaptchuk, and Elissa M Redmiles. " i need a better description": An investigation into user expectations for differential privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3037–3052, 2021.

Rachel Cummings, Damien Desfontaines, David Evans, Roxana Geambasu, Yangsibo Huang, Matthew Jagielski, Peter Kairouz, Gautam Kamath, Sewoong Oh, Olga Ohrimenko, Nicolas Papernot, Ryan Rogers, Milan Shen, Shuang Song, Weijie Su, Andreas Terzis, Abhradeep Thakurta, Sergei Vassilvitskii, Yu-Xiang Wang, Li Xiong, Sergey Yekhanin, Da Yu, Huanyu Zhang, and Wanrong Zhang. Advancing Differential Privacy: Where We Are Now and Future Directions for Real-World Deployment. *Harvard Data Science Review*, 6(1), jan 16 2024. https://hdsr.mitpress.mit.edu/pub/sl9we8gh.

Sylvie Delacroix and Neil D Lawrence. Bottom-up data trusts: Disturbing the 'one size fits all'approach to data governance. *International data privacy law*, 9(4):236–252, 2019.

Dorothy E Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

Dorothy E Denning and Peter J Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

Damien Desfontaines. What's up with all these large privacy budgets? `https://desfontaines.es/blog/large-epsilons.html`, 09 2024. Ted is writing things (personal blog).

Onyinye Dibia, Prianka Bhattacharjee, Brad Stenger, Steven Baldasty, Mako Bates, Ivoline C Ngong, Yuanyuan Feng, and Joseph P Near. Sok: Usability studies in differential privacy. *arXiv preprint arXiv:2412.16825*, 2024.

Verena Distler, Matthias Fassl, Hana Habib, Katharina Krombholz, Gabriele Lenzini, Carine Lallemand, Lorrie Faith Cranor, and Vincent Koenig. A systematic literature review of empirical methods and risk representation in usable privacy and security research. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 28(6):1–50, 2021.

DuckDuckGo. Duckduckgo ios, 2025. URL `https://github.com/duckduckgo/ios`.

Lea Duesterwald, Ian Yang, and Norman Sadeh. Can a cybersecurity question answering assistant help change user behavior? an in situ study. Symposium on Usable Security and Privacy (USEC) 2025-NDSS Symposium, 2025.

Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, July 2006. ISBN 3-540-35907-9. URL `https://www.microsoft.com/en-us/research/publication/differential-privacy/`.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

Cynthia Dwork, Moni Naor, Omer Reingold, Guy N Rothblum, and Salil Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 381–390, 2009.

Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2010.

Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

Cynthia Dwork, Nitin Kohli, and Deirdre Mulligan. Differential privacy in practice: Expose your epsilons! *Journal of Privacy and Confidentiality*, 9(2), 2019.

Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazieres, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. *ACM SIGOPS Operating Systems Review*, 39(5):17–30, 2005.

William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2468–2479. SIAM, 2019.

Jens Ernstberger, Jan Lauinger, Fatima Elsheimy, Liyi Zhou, Sebastian Steinhorst, Ran Canetti, Andrew Miller, Arthur Gervais, and Dawn Song. Sok: data sovereignty. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 122–143. IEEE, 2023.

European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council, 2016. URL `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679`.

Vitaly Feldman and Tijana Zrnic. Individual privacy accounting via a renyi filter. *Advances in Neural Information Processing Systems*, 34:28080–28091, 2021.

Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, pages 1–14, 2012.

Yuanyuan Feng, Yaxing Yao, and Norman Sadeh. A design space for privacy choices: Towards meaningful privacy control in the internet of things. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2021.

Yuanyuan Feng, Abhilasha Ravichander, Yaxing Yao, Shikun Zhang, and Rex Chen. Understanding how to inform blind and {Low-Vision} users about data privacy through privacy question answering assistants. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2065–2082, 2024.

Simone Fischer-Hübner and Farzaneh Karegar. Overview of usable privacy research: Major themes and research directions. *The Curious Case of Usable Privacy: Challenges, Solutions, and Prospects*, pages 43–102, 2024.

Gonzalo Munilla Garrido, Xiaoyuan Liu, Florian Matthes, and Dawn Song. Lessons learned: Surveying the practicality of differential privacy in the industry. *arXiv preprint arXiv:2211.03898*, 2022.

Chang Ge, Xi He, Ihab F Ilyas, and Ashwin Machanavajjhala. Apex: Accuracy-aware differentially private data exploration. In *Proceedings of the 2019 International Conference on Management of Data*, pages 177–194, 2019.

Joseph A Goguen and José Meseguer. Security policies and security models. In *1982 IEEE symposium on security and privacy*, pages 11–11. IEEE, 1982.

Yue Gong, Zhiru Zhu, Sainyam Galhotra, and Raul Castro Fernandez. Ver: View discovery in the wild. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 503–516. IEEE, 2023.

Google. Protocol buffers, 2025. URL `https://protobuf.dev/`.

G Scott Graham and Peter J Denning. Protection: principles and practice. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 417–429, 1971.

Hana Habib and Lorrie Faith Cranor. Evaluating the usability of privacy choice mechanisms. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*, pages 273–289, 2022.

Margaret Hagen. {User-Centered} privacy communication design. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

Alon Y Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. Exploring privacy-accuracy tradeoffs using dpcomp. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2101–2104, 2016.

Xi He, Ashwin Machanavajjhala, and Bolin Ding. Blowfish privacy: Tuning privacy-utility trade-offs using policies. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1447–1458, 2014.

Joseph M Hellerstein and Michael Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 267–276, 1993.

Home Assistant. Home assistant for apple platforms, 2025. URL `https://github.com/home-assistant/iOS`.

Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 398–410. IEEE, 2014.

Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

Patrik Hummel, Matthias Braun, Max Tretter, and Peter Dabrock. Data sovereignty: A review. *Big Data & Society*, 8(1):2053951720982012, 2021.

Ada Lovelace Institute. Exploring legal mechanisms for data stewardship. *Ada Lovelace Institute and UK AI Council*, 2021.

Yasha Iravantchi, Pardis Emami-Naeini, and Alanson Sample. Sok:(un) usable privacy: the lack of overlap between privacy-aware sensing and usable privacy research. *Proceedings on Privacy Enhancing Technologies*, 2025.

Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. A classification of object-relational impedance mismatch. In *2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications*, pages 36–43. IEEE, 2009.

Mark F St John, Grit Denker, Peeter Laud, Karsten Martiny, Alisa Pankova, and Dusko Pavlovic. Decision support for sharing data using differential privacy. In *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 26–35. IEEE, 2021.

Zach Jorgensen, Ting Yu, and Graham Cormode. Conservative or liberal? personalized differential privacy. In *2015 IEEE 31St international conference on data engineering*, pages 1023–1034. IEEE, 2015.

Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

Simon Koch, Manuel Karl, Robin Kirchner, Malte Wessels, Anne Paschke, and Martin Johns. The impact of default mobile sdk usage on privacy and data protection. *Proceedings on Privacy Enhancing Technologies*, 2025.

Ronny Kohavi and Barry Becker. UCI Adult dataset, 2026. URL `https://www.kaggle.com/datasets/uciml/adult-census-income`.

Nitin Kohli and Paul Laskowski. Epsilon voting: Mechanism design for parameter selection in differential privacy. In *2018 IEEE Symposium on Privacy-Aware Computing (PAC)*, pages 19–30. IEEE, 2018.

Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. *ACM SIGOPS Operating Systems Review*, 41(6):321–334, 2007.

Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile networks and Applications*, 18:129–140, 2013.

Butler W Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

Florian Lauf, Simon Scheider, Jan Bartsch, Philipp Herrmann, Marija Radic, Marcel Rebbert, André T Nemat, Christoph Schlueter Langdon, Ralf Konrad, Ali Sunyaev, et al. Linking data sovereignty and data economy: arising areas of tension. 2022.

Jaewoo Lee and Chris Clifton. How much is enough? choosing ϵ for differential privacy. In *International Conference on Information Security*, pages 325–340. Springer, 2011.

Katrina Ligett, Seth Neel, Aaron Roth, Bo Waggoner, and Steven Z Wu. Accuracy first: Selecting a differential privacy level for accuracy constrained erm. *Advances in Neural Information Processing Systems*, 30, 2017.

Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *arXiv preprint arXiv:1603.01699*, 2016.

Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 30(1):122–173, 2005.

Patrick E McKnight and Julius Najab. Mann-whitney u test. *The Corsini encyclopedia of psychology*, pages 1–1, 2010.

Jack Murtagh, Kathryn Taylor, George Kellaris, and Salil Vadhan. Usable differential privacy: A case study with psi. *arXiv preprint arXiv:1809.04103*, 2018.

Andrew C Myers and Barbara Liskov. A decentralized model for information flow control. *ACM SIGOPS Operating Systems Review*, 31(5):129–142, 1997.

Priyanka Nanayakkara, Johes Bater, Xi He, Jessica Hullman, and Jennie Rogers. Visualizing privacy-utility trade-offs in differentially private data releases. *arXiv preprint arXiv:2201.05964*, 2022.

Priyanka Nanayakkara, Mary Anne Smart, Rachel Cummings, Gabriel Kaptchuk, and Elissa M Redmiles. What are the chances? explaining the epsilon parameter in differential privacy. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1613–1630, 2023.

Joseph P Near, Xi He, et al. Differential privacy for databases. *Foundations and Trends® in Databases*, 11(2):109–225, 2021.

Nextcloud. Nextcloud ios ap, 2025. URL `https://github.com/nextcloud/ios`.

T Nipkow et al. A perspective on information-flow control. *Software Safety and Security: Tools for Analysis and Verification*, page 319, 2012.

Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004.

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

Jonathan A Obar and Anne Oeldorf-Hirsch. The biggest lie on the internet: Ignoring the privacy policies and terms of service policies of social networking services. *Information, Communication & Society*, 23(1):128–147, 2020.

OpenWeatherMap. Openweathermap api for current weather data, 2025. URL https://openweathermap.org/current.

Shidong Pan, Zhen Tao, Thong Hoang, Dawen Zhang, Tianshi Li, Zhenchang Xing, Xiwei Xu, Mark Staples, Thierry Rakotoarivelo, and David Lo. A {NEW}{HOPE}: Contextual privacy policies for mobile applications and an approach toward automated generation. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5699–5716, 2024.

Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM transactions on information and system security (TISSEC)*, 7(1):128–174, 2004.

Radim Polčák and Dan Jerker B Svantesson. *Information sovereignty: data privacy, sovereign powers and the rule of law*. Edward Elgar Publishing, 2017.

Raluca Ada Popa. Confidential computing or cryptographic computing? tradeoffs between cryptography and hardware enclaves. *Queue*, 22(2):108–132, 2024.

Eric Posner and Eric Weyl. *Radical markets: Uprooting capitalism and democracy for a just society*. Princeton University Press, 2018.

PostgreSQ. Foreign data, 2025. URL https://www.postgresql.org/docs/current/ddl-foreign-data.html.

Prashanth Rajivan and Jean Camp. Influence of privacy attitude and privacy cue framing on android app {Choices}. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

Rachel Redberg and Yu-Xiang Wang. Privately publishable per-instance privacy. *Advances in Neural Information Processing Systems*, 34:17335–17346, 2021.

Mozilla researc. What does it mean? | shifting power through data governance, 2020. URL https://foundation.mozilla.org/en/data-futures-lab/data-for-empowerment/shifting-power-through-data-governance/.

David Rodriguez, Jose M Del Alamo, Celia Fernández-Aller, and Norman Sadeh. Sharing is not always caring: Delving into personal data transfer compliance in android apps. *IEEE Access*, 12:5256–5269, 2024.

David Rodriguez, Joseph A Calandrino, Jose M Del Alamo, and Norman Sadeh. Privacy settings of third-party libraries in android apps: A study of facebook sdks. *Proceedings on Privacy Enhancing Technologies*, 2025.

Ryan M Rogers, Aaron Roth, Jonathan Ullman, and Salil Vadhan. Privacy odometers and filters: Pay-as-you-go composition. *Advances in Neural Information Processing Systems*, 29, 2016.

Mark Ryan, Paula Gürtler, and Artur Bogucki. Will the real data sovereign please stand up? an eu policy response to sovereignty in data spaces. *International Journal of Law and Information Technology*, 32:eaae006, 2024.

Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, volume 1, pages 57–64. IEEE, 2015.

SafeInsights Team. Safeinsights press kit, 2026a. URL `https://www.safeinsights.org/docs/SafeInsights-Press-Kit.pdf`.

SafeInsights Team. safeinsights.org, 2026b. URL `https://www.safeinsights.org/`.

Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.

Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, 2002.

Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 2002.

Florian Schaub, Rebecca Balebako, Adam L Durity, and Lorrie Faith Cranor. A design space for effective privacy notices. In *Eleventh symposium on usable privacy and security (SOUPS 2015)*, pages 1–17, 2015.

Jeremy Seeman, William Sexton, David Pujol, and Ashwin Machanavajjhala. Privately answering queries on skewed data via per-record differential privacy. 17(11):3138–3150, August 2024. ISSN 2150-8097. doi:10.14778/3681954.3681989. URL `https://doi.org/10.14778/3681954.3681989`.

Signal. Technology deep dive: Building a faster oram layer for enclaves, 2022. URL `https://signal.org/blog/building-faster-oram/`.

Signal. Signal ios, 2025. URL `https://github.com/signalapp/Signal-iOS`.

Simplenote. Simplenote for ios, 2025. URL `https://github.com/automattic/simplenote-ios`.

Mary Anne Smart, Dhruv Sood, and Kristen Vaccaro. Understanding risks of privacy theater with differential privacy. *Proceedings of the ACM on Human-Computer Interaction*, 6 (CSCW2):1–24, 2022.

Daniel J Solove. *Understanding privacy*. Harvard university press, 2010.

Chandler Nicholle Spinks. Contemporary housing discrimination: Facebook, targeted advertising, and the fair housing act. *Hous. L. Rev.*, 57:925, 2019.

SQLite. The virtual table mechanism of sqlite, 2025a. URL `https://www.sqlite.org/vtab.html`.

SQLite. Virtual table indexing information, 2025b. URL https://www.sqlite.org/c3ref/index_info.html.

SQLite. Virtual table object, 2025c. URL https://www.sqlite.org/c3ref/module.html.

Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 91–100, 2014.

Pratiksha Thaker, Mihai Budiu, Parikshit Gopalan, Udi Wieder, and Matei Zaharia. Overlook: Differentially private exploratory visualization for big data. *arXiv preprint arXiv:2006.12018*, 2020.

VLC. Vlc media player, 2025. URL https://github.com/videolan/vlc.

Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. Dp-cryptography: marrying differential privacy and cryptography in emerging applications. *Communications of the ACM*, 64(2):84–93, 2021.

Yu-Xiang Wang. Per-instance differential privacy. *Journal of Privacy and Confidentiality*, 9 (1), 2019.

Justin Whitehouse, Aaditya Ramdas, Ryan Rogers, and Steven Wu. Fully-adaptive composition in differential privacy. In *International Conference on Machine Learning*, pages 36990–37007. PMLR, 2023.

Wikipedia. Wikipedia ios, 2025. URL https://github.com/wikimedia/wikipedia-ios.

WordPress. Wordpress for ios, 2025. URL https://github.com/wordpress-mobile/WordPress-iOS.

Siyuan Xia, Zhiru Zhu, Chris Zhu, Jinjin Zhao, Kyle Chard, Aaron J Elmore, Ian Foster, Michael Franklin, Sanjay Krishnan, and Raul Castro Fernandez. Data station: delegated, trustworthy, and auditable computation to enable data-sharing consortia with a data escrow. *Proceedings of the VLDB Endowment*, 15(11):3172–3185, 2022.

Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing {Non-Compliance} of apple privacy labels. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1091–1108, 2023.

Aiping Xiong, Tianhao Wang, Ninghui Li, and Somesh Jha. Towards effective differential privacy communication for users' data sharing decision and comprehension. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 392–410. IEEE, 2020.

Angela Yang. More than 11,000 creatives condemn unauthorized use of content for ai development, 2024.

Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. Who knows what about me? a survey of behind the scenes personal data sharing to third parties by mobile apps. *Technology Science*, 30:2–1, 2015.

Shikun Zhang and Norman Sadeh. Do privacy labels answer users' privacy questions. In *Network and Distributed System Security Symposium*, 2023.

Shikun Zhang, Yuanyuan Feng, Yaxing Yao, Lorrie Faith Cranor, and Norman Sadeh. How usable are ios app privacy labels? *UMBC Faculty Collection*, 2022.

Shikun Zhang, Lily Klucinec, Kyerra Norton, Norman Sadeh, and Lorrie Faith Cranor. Exploring {Expandable-Grid} designs to make {iOS} app privacy labels more usable. In *Twentieth Symposium on Usable Privacy and Security (SOUPS 2024)*, pages 139–157, 2024a.

Yifan Zhang, Zhaojie Hu, Xueqiang Wang, Yuhui Hong, Yuhong Nan, XiaoFeng Wang, Jiatao Cheng, and Luyi Xing. Navigating the privacy compliance maze: Understanding risks with {Privacy-Configurable} mobile {SDKs}. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6543–6560, 2024b.

Yuqing Zhu, Jinshuo Dong, and Yu-Xiang Wang. Optimal accounting of differential privacy via characteristic function. In *International Conference on Artificial Intelligence and Statistics*, pages 4782–4817. PMLR, 2022.

Zimperium. 2025 zimperium global mobile threat report, 2025. URL `https://lp.zimperium.com/hubfs/Reports/2025%20Global%20Mobile%20Threat%20Report.pdf`.

Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1296–1310. IEEE, 2019.