October 2021

Enhancing Usability and Explainability of Data Systems

Anna Fariha
University of Massachusetts Amherst

ENHANCING USABILITY AND EXPLAINABILITY OF DATA SYSTEMS

A Dissertation Presented

by

ANNA FARIHA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2021

College of Information and Computer Sciences

ENHANCING USABILITY AND EXPLAINABILITY OF DATA SYSTEMS

A Dissertation Presented

by

ANNA FARIHA

Approved as to style and content by:

Alexandra Meliou, Chair

Peter J. Haas, Member

Emery Berger, Member

Suman Nath, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

ACKNOWLEDGMENTS

First and foremost, I would like to thank my Ph.D. advisor Prof. Alexandra Meliou. I am extremely grateful to her for offering me such a wonderful and positive experience throughout my graduate career through her excellent mentorship. She not only helped me develop a great research philosophy, but also, taught me how to communicate my ideas effectively. I was deeply inspired by her words: "your research is of no use if you can't *communicate* it well". She made me realize the true value of communication: writing research papers, giving talks, interpreting reviewer's comments, arguing concisely in a paper rebuttal, and, finally, reviewing others' work. She helped me become a better version of myself and pushed me further to become a perfectionist. So, thanks Alexandra for helping me grow in many different ways.

I was extremely fortunate to have the guidance of a couple of great mentors along with my advisor. I thank my internship mentor Dr. Ashish Tiwari (Microsoft) who not only guided me in my research through an unconventionally hands-on approach, but also, helped me during difficult times when I was dealing with multiple paper rejections with his words "you should not take comments *personally*". I realized that getting through a Ph.D. is not only about doing good research, rather, a holistic attitude involving all stages and components of research, from validating research ideas to literature survey and handling rejections positively.

I would like to thank my internship mentor Dr. Suman Nath (Microsoft Research), who offered his continuous mentorship throughout my entire graduate career. Suman helped me branch out to research areas beyond my initial expertise that posed a unique challenge to me, and, eventually, it turned out to be an excellent learning experience. At times when I was struggling to position my work, he pointed out critical weaknesses, saying "you should

discuss the *scope* and *limitations* of your work", which was an excellent piece of advice that helped me in all subsequent work.

A great source of inspiration for my research and being able to see beyond research, which is its *impact*, was Dr. Sumit Gulwani (Microsoft). I am also indebted to Prof. Emery Berger and Prof. Peter Haas for their excellent mentorship and encouragement, especially during the last year of my Ph.D. I would also like to extend my thanks to Prof. Yuriy Brun, Prof. Narges Mahyar, Dr. Divesh Srivastava (AT&T), and Prof. Juliana Freire (NYU) for their great mentorship in different projects. I was very fortunate to get to know all these experienced researchers during my Ph.D. and experience a diverse set of research outlooks. I would like to thank all my collaborators and peers: Arjun Radhakrishna (Microsoft), Matteo Brucato, Sainyam Galhotra, Raoni Lourenço (NYU), Nishant Yadav, Oscar Youngquist, Julian Killingback, Lucy Cousins, and others.

I would like to thank Microsoft Research for awarding me a Microsoft Research Dissertation Grant, which significantly supported the last year of my Ph.D. Thanks to all past and current members of the DREAM lab—Prof. Marco Serafini, Prof. Gerome Miklau, Prof. Yanlei Diao, Prof. Laura Haas, Ryan, Iro, Dan, Xiaolan, Maliha, Abhishek, Yue, and others—and members of CICS—Raj, Larkin, Sheshera, Tony, Rachel, Leeanne, Eileen, Malaika, and the CSCF staff—for giving me all kinds of support, encouragement, inspiration, and a great environment to learn and grow.

At the completion of my academic career, I would like to thank all my teachers, friends, and peers, who helped me build the foundation prior to and during my Ph.D., which made this work possible. Finally, I would like to thank my parents and all my extended family members, who always encouraged me and extended their support in every possible way throughout my entire academic career.

ABSTRACT

ENHANCING USABILITY AND EXPLAINABILITY OF DATA SYSTEMS

SEPTEMBER 2021

ANNA FARIHA

B.S., UNIVERSITY OF DHAKA

M.S., UNIVERSITY OF DHAKA

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Alexandra Meliou

The recent growth of data science expanded its reach to an ever-growing user base of nonexperts, increasing the need for *usability*, *understandability*, and *explainability* in these systems. Enhancing usability makes data systems accessible to people with different skills and backgrounds alike, leading to *democratization* of data systems. Furthermore, proper understanding of data and data-driven systems is necessary for the users to *trust* the function of the systems that learn from data. Finally, data systems should be *transparent*: when a data system behaves unexpectedly or malfunctions, the users deserve proper explanation of what caused the observed incident.

Unfortunately, most existing data systems offer limited usability and support for explanations: these systems are usable only by experts with sound technical skills, and even

expert users are hindered by the lack of transparency into the systems' inner workings and functions. The aim of my thesis is to bridge the usability gap between nonexpert users and complex data systems, aid all sort of users—including the expert ones—in data and system understanding, and provide explanations that help reason about unexpected outcomes involving data systems. Specifically, my thesis has the following three goals: (1) enhancing usability of data systems for nonexperts, (2) enable data understanding that can assist users in a variety of tasks such as achieving trust in data-driven machine learning, gaining data understanding, and data cleaning, and (3) explaining causes of unexpected outcomes involving data and data systems.

For enhancing usability, we focus on *example-driven user intent discovery*. We develop systems based on example-driven interactions in two different settings: querying relational databases and personalized document summarization. Towards data understanding, we develop a *new data-profiling primitive* that can characterize tuples for which a machine-learned model is likely to produce untrustworthy predictions. We also develop an explanation framework to explain causes of such untrustworthy predictions. Additionally, this new data-profiling primitive enables interactive data cleaning. Finally, we develop two explanation frameworks, tailored to provide explanations in *debugging data system components*, including the data itself. The explanation frameworks focus on explaining the root cause of a concurrent application's intermittent failure and exposing issues in the data that cause a data-driven system to malfunction.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Rapid increase of computational resources and data-sharing platforms has reached an ever-growing base of users without technical computing expertise, who wish to peruse, analyze, and understand data. From astronomers and scientists who need to analyze data to validate their hypotheses, all the way to computational journalists who need to peruse datasets to validate claims and support their reporting, the broad availability of data has the potential to fundamentally impact the way domain experts conduct their work. Unfortunately, while data is broadly available, access to data systems is seldom unfettered. Challenges in the use of data systems can stem from various aspects of the systems' operation: how users interface with the system, how they reason about the system behavior, whether they trust the system or not, and how they fix the system in the face of system malfunction.

Existing systems typically cater to users with sound technical computing and programming skills, posing significant hurdles to technical novices, who do not have strong technical background. Democratization of computational systems demands equal access to people of different skills and backgrounds [127, 253, 261]. Therefore, the first goal of my thesis is to design and develop tools and mechanisms that can *enhance usability* of data systems.

At the core of all these data systems lies data, which is central to modern systems in a wide range of domains, including healthcare, transportation, and finance. The core of modern data-driven systems typically comprises of models learned from large datasets, and they are usually optimized to target particular data and workloads. While these data-driven

systems have seen wide adoption and success, their reliability and proper function hinge on the data's continued conformance to the systems' initial settings and assumptions. If the serving data (on which the system operates) deviates from the profile of the initial data (on which the system was trained), then system performance degrades and system behavior becomes unreliable. A mechanism to assess the trustworthiness of a system's inferences is paramount, especially for systems that perform safety-critical or high-impact operations. To achieve this, we need to first obtain data understanding, which can in turn help us detect when data has deviated to an extent that may compromise the trustworthiness of a system's inferences over the deviated data. Beyond achieving trust in data-driven machine learning, data understanding also helps us in other important tasks such as data-drift quantification and data cleaning. To this end, the second goal of my thesis is to design data-profiling primitives that can help users gain *data understanding*, which in turn helps in other important tasks such as achieving trustworthy machine learning, data-drift quantification, data cleaning, and so on.

When data systems do not behave as expected, we need mechanisms that can explain to us what is causing such unexpected behavior. Furthermore, application development in distributed settings brings forth important challenges that also need explanation. For example, failures in distributed applications are often triggered by concurrency bugs, such as interference and coordination issues. Such concurrency bugs are difficult to reproduce; therefore, the root causes of concurrent program failures are hard to identify, and even harder to explain. Without succinct explanation of how the root cause eventually triggers failure, the developer might fail to draw the causal connection from the root cause to the failure. Furthermore, as data is a central component of data-driven systems, the cause of system malfunction may reside in the data as well. Therefore, similar to software debugging, we also need to debug the data to identify potential sources of disconnect between the assumptions about the data and the systems that operate on that data. To this end, the

third goal of my thesis is to develop *explanation frameworks* to assist in debugging various components of data systems, including the data itself.

We now provide a brief overview of the work completed to achieve the three aforementioned goals. We outline our work towards enhancing usability of data systems in Section 1.1; our mechanisms to enable data understanding to achieve trusted machine learning, data-drift quantification, and data cleaning in Section 1.2; and our explanation frameworks to assist in debugging various components of data systems in Section 1.3.

1.1 Enhancing Usability: Example-Driven Intent Discovery

While using data systems, the traditional setting requires the users to precisely specify their intended task in a language that the systems understand. For example, a user has to compose a precise SQL query to retrieve data from a relational database or issue a well-formed natural language query to extract relevant data from a large document in a question-answering system. Such rigid mechanisms pose hurdles for nonexperts who lack technical expertise and are unfamiliar with the details of the data organization. However, a number of data-centric tasks can be simply expressed by *examples*. Such a *programming-by-example* mechanism significantly enhances usability of these systems for nonexperts [65, 126, 206, 285]. Towards enhancing usability of data systems, we focus on *example-driven* techniques to discover user intents. Next, we discuss our contributions in this area.

1.1.1 Querying Relational Databases by Example (SQuID)

Traditional relational data interfaces require precise structured queries over potentially complex schemas. These rigid data retrieval mechanisms pose hurdles for nonexpert users, who typically lack programming language expertise and are unfamiliar with the details of the schema. Existing tools assist in formulating queries through keyword search, query recommendation, and query auto-completion, but still require some technical expertise. An alternative method for accessing data is *query by example* (QBE), where users express their

data exploration intent simply by providing examples of their intended data and the system infers the intended query. However, existing QBE approaches focus on the structural similarity of the examples and ignore the richer context present in the data. As a result, they typically produce queries that are too general, and fail to capture the user's intent effectively.

To overcome these challenges, we develop SQUID [94, 95] a system that performs *semantic-similarity-aware* query intent discovery from user-provided example tuples. SQUID is an end-to-end system that automatically formulates select-project-join queries with optional group-by aggregation and intersection operators—a much larger class than what prior QBE techniques support—from user-provided examples, in an open-world setting. To express the problem of query intent discovery, we use a *probabilistic abduction model* that infers a query as the most likely explanation of the provided examples. To expedite query intent discovery, we introduce the notion of an *abduction-ready* database, which precomputes semantic properties and related statistics, allowing SQUID to achieve real-time performance.

Our extensive empirical evaluation on three real-world datasets, including user intent case studies, demonstrates that SQUID is efficient and effective, and outperforms machine learning methods, as well as the state of the art in the related query reverse engineering problem. To understand how effective SQUID is for real users over real-world data exploration tasks, we conducted comparative user studies contrasting SQUID with traditional SQL querying [93]. The user studies demonstrate that users with varying expertise are significantly more effective and efficient with SQUID than SQL. We find that SQUID eliminates the barriers in studying the database schema, formalizing task semantics, and writing syntactically correct SQL queries, and, thus, substantially alleviates the need for technical expertise in data exploration.

1.1.2 Summarizing Documents by Example (SUDOCU)

As a use case of by-example interaction beyond relational data domain, we apply the mechanism for example-driven user intent discovery to the area of text document summarization. Text document summarization refers to the task of producing a brief representation of a document for easy human consumption. Automatic text document summarization is a key natural language processing task. Existing text summarization techniques mostly focus on generic summarization, but users often require *personalized* summarization that targets their specific preferences and needs. However, precisely expressing preferences is challenging, and current methods are often ambiguous, outside the user's control, or require costly training data. We propose a novel and effective way to express summarization intent (preferences) via *examples*: the user provides a few example summaries for a small number of documents in a collection, and the system summarizes the rest. We develop SUDOCU, an example-based personalized DOCUment SUmmarization system [91]. Through a simple interface, SUDOCU allows the users to provide example summaries, learns the summarization intent from the examples, and produces summaries for new documents that reflect the user's summarization intent. SUDOCU further explains the captured summarization intent in the form of a *package query*, an extension of a traditional SQL query that handles complex constraints and preferences over answer sets. SUDOCU combines topic modeling, semantic similarity discovery, and in-database optimization in a novel way to achieve example-driven document summarization.

1.2 Data Understanding: Conformance Constraints

The reliability of inferences made by data-driven systems hinges on the data's continued conformance to the systems' initial settings and assumptions about the data. Data profiling refers to the task of extracting technical metadata that captures these assumptions about the data. Such metadata is also known as *profiles* and has numerous applications such as data understanding, validation, integration, and cleaning. While a number of data

profiling primitives exist in the literature, most of them are limited to categorical attributes. A few techniques consider numerical attributes; but, they either focus on simple relationships involving a pair of attributes (e.g., correlations) or convert the continuous semantics of numerical attributes to a discrete semantics, which results in information loss. To capture more complex relationships involving the numerical attributes, we design a new data-profiling primitive called *conformance constraints*, which can model linear *arithmetic* relationships involving multiple *numerical* attributes (called *projections*). Existing data profiling primitives such as functional dependencies and denial constraints cannot model such relationships. Beyond data understanding, there are a number of important applications of conformance constraints. In this work, we focus on a few of them: trusted machine learning, data-drift quantification, and data cleaning. Furthermore, we develop an explanation framework that can explain causes of tuple non-conformance guided by conformance constraints.

The reliability of inferences made by data-driven systems hinges on the data's continued conformance to the systems' initial settings and assumptions. When serving data (on which we want to apply inference) deviates from the profile of the initial training data, the outcome of inference becomes unreliable. Conformance constraints are tailored towards quantifying the degree of *non-conformance*, which can effectively characterize if inference over that tuple is *untrustworthy*. Our key finding is that projections that incur *low variance* on a dataset construct effective conformance constraints. This principle yields the surprising result that low-variance components of a principal component analysis, which are usually discarded for dimensionality reduction, generate stronger conformance constraints than the high-variance components. Based on this result, we provide a highly scalable and efficient technique—linear in data size and cubic in the number of attributes—for discovering conformance constraints for a dataset [100, 101]. To measure the degree of a tuple's non-conformance with respect to a dataset, we propose a *quantitative semantics* that captures how much a tuple violates the conformance constraints of that dataset. We demon-

strate the value of conformance constraints on two applications: *trusted machine learning* and *data drift*. We empirically show that conformance constraints offer mechanisms to (1) reliably detect tuples on which the inference of a machine-learned model should not be trusted, and (2) quantify data drift more accurately than the state of the art.

To allow interactive discovery and exploration of COnformance COnstraints for understanding trends involving the numerical attributes of a dataset, we develop COCO [98], with a particular focus on the application of data cleaning. COCO enables the user to guide conformance constraint discovery according to their preferences. The user can examine to what extent a new, possibly dirty, dataset satisfies or violates the discovered conformance constraints. Further, COCO provides useful suggestions for cleaning dirty data tuples, where the user can interactively alter cell values, and verify by checking change in conformance constraint violation due to the alteration. In summary, COCO can help in understanding trends in the data and assist the users in interactive data cleaning, using conformance constraints.

In data-driven systems, we often encounter tuples on which the predictions of a machine-learned model are untrustworthy. A key cause of such untrustworthiness is *non-conformance* of a new tuple with respect to the training dataset. To check conformance, we can use conformance constraints, which capture a set of implicit constraints that all tuples of a dataset satisfy: a test tuple is non-conforming if it violates the conformance constraints. Conformance constraints model complex relationships among multiple attributes; but do not provide interpretable explanations of non-conformance. To EXplaining causes of TUple Non-conformancE, we develop EXTUNE [99]. Based on the principles of causality, EXTUNE assigns *responsibility* to the attributes for causing non-conformance. The key idea is to observe change in constraint violation under *intervention* on attribute-values. EXTUNE produces a ranked list of the test tuples based on their degree of non-conformance and visualizes tuple-level attribute responsibility for non-conformance through heat maps. EXTUNE further visualizes attribute responsibility, aggregated over the test tuples. Through

real-world case studies, we show how EXTUNE can explain causes of tuple non-conformance and assist the users to make careful decisions towards achieving trusted machine learning.

1.3 Explanation Frameworks for Debugging Data Systems

When systems do not behave as expected, even expert users struggle to understand the causes of such unexpected behavior. They wonder why a system crashes intermittently, why an ML model fails for certain datasets while passes for others, whether there is a causal relationship between a pair of anomalous events, and so on. Explanation frameworks facilitate such understanding. Next, we discuss our two contributions in building explanation frameworks for (1) tracing issues within components and behaviors of data systems and (2) tracing issues related to the data itself.

1.3.1 Causality-Guided Adaptive Interventional Debugging (AID)

Runtime nondeterminism is a fact of life in modern database applications. Previous research has shown that nondeterminism can cause applications to *intermittently* crash, become unresponsive, or experience data corruption. We propose Adaptive Interventional Debugging (AID) for debugging such intermittent failures [96, 97]. AID combines existing statistical debugging, causal analysis, fault injection, and group testing techniques in a novel way to (1) pinpoint the root cause of an application's intermittent failure and (2) generate an explanation of how the root cause triggers the failure. AID works by first identifying a set of runtime behaviors (called predicates) that are strongly correlated to the failure. It then utilizes temporal properties of the predicates to (over)-approximate their causal relationships. Finally, it uses fault injection to execute a sequence of interventions on the predicates and discover their true causal relationships. This enables AID to identify the true root cause and its causal relationship to the failure. We theoretically analyze how fast AID can converge to the identification. We evaluate AID with six real-world applications that intermittently fail under specific inputs. In each case, AID was able to identify

the root cause and explain how the root cause triggered the failure, much faster than group testing and more precisely than statistical debugging. We also evaluate AID with many synthetically generated applications with known root causes and confirm that the benefits also hold for them.

1.3.2 Exposing Disconnect between Data and Systems (DATAEXPOSER)

As data is a central component of many modern systems, the cause of a system malfunction may reside in the data, and, specifically, particular properties of the data. For example, a health-monitoring system that is designed under the assumption that weight is reported in imperial units (lbs) will malfunction when encountering weight reported in metric units (kilograms). Similar to software debugging, which aims to find bugs in the mechanism (source code or runtime conditions), our goal is to debug the data to identify potential sources of *disconnect* between the assumptions about the data and the systems that operate on that data. Specifically, we seek which *properties* of the data cause a data-driven system to malfunction. We propose DATAEXPOSER [112], a framework to identify data properties, called *profiles*, that are the root causes of performance degradation, or failure, of a system that operates on the data. Such identification is necessary to repair the system and resolve the disconnect between data and system. Our technique is based on *causal reasoning* through *interventions*: when a system malfunctions for a dataset, DATAEXPOSER alters the data profiles and observes changes in the system's behavior due to the alteration. Unlike statistical observational analysis that reports mere correlations, DATAEXPOSER reports causally verified root causes—in terms of data profiles—of the system malfunction. We empirically evaluate DATAEXPOSER on three real-world and several synthetic data-driven systems that fail on datasets due to a diverse set of reasons. In all cases, DATAEXPOSER identifies the root causes precisely while requiring orders of magnitude fewer interventions than prior techniques.

1.4 Organization

To familiarize the reader with the prior work, Chapter 2 provides a literature survey pertinent to the work presented in this thesis. The remaining of the thesis is organized in three parts.

Part I, which contains Chapters 3 and 4, describes the systems involving example-driven user intent discovery. Chapter 3 describes the example-driven query intent discovery framework SQUID and presents our comparative user studies contrasting SQUID against traditional SQL querying. Chapter 4 describes the example-driven document summarization system SUDOCU along with the subjective summarization dataset SUBSUME.

Part II presents the data profiling primitive conformance constraints, including their applications. Chapter 5 provides the discovery algorithm for conformance constraints along with discussion on two application areas: trusted machine learning and data-drift quantification. It also includes discussion on a tool COCO that focuses on the application of data cleaning, and another tool EXTUNE that explains the causes of tuple non-conformance, guided by conformance constraints.

Part III, which contains Chapters 6 and 7, presents two explanation frameworks to assist in debugging various components of data systems. Chapter 6 describes AID that explains how a root cause triggers an intermittent failure in concurrent systems. Chapter 7 describes a framework that exposes which properties of a dataset cause a data-driven system to malfunction.

Finally, we conclude the thesis in Chapter 8 summarizing the work presented and highlight a few directions towards future work.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we provide an overview of the related literature. We start by providing a literature review on enhancing usability of data systems and contrast with those our proposed approaches (Section 2.1). Then we proceed to discuss existing literature on data profiling techniques and prior work that address the problem of trusted machine learning, data-drift quantification, data cleaning, and explaining data anomalies (Section 2.2). Finally, we review the literature related to our work on explanation frameworks to assist in debugging data systems (Section 2.3).

2.1 Enhancing System Usability

In this section, we provide an overview of the existing programming-by-example (PBE) and query-by-example (QBE) approaches, discuss alternative mechanisms that also aid users in data exploration, give a brief overview of other closely related work (e.g., query reverse engineering (QRE), set expansion, machine learning etc.), highlight prior user studies over other PBE approaches, and, finally, review the literature on text document summarization.

2.1.1 Programming by Example (PBE)

PBE is based on the intuitive premise that users who may lack or have low technical skills, but have expertise in a particular domain, can more easily express their computational desire by providing examples than by writing programs under strict language specifications. This is in contrast with traditional program synthesis [163, 271], which requires

a high-level formal specification (e.g., first-order logic) of the desired program. Example-driven program synthesis has been effectively used for a variety of tasks, such as code synthesis for data scientists [77]; data wrangling [125], integration [157], extraction [31, 198], transformation [124, 135], and filtering [326]; data structure transformation [104]; text processing [339] and normalization [186]; querying relational databases [294], and so on.

Many PBE approaches have been developed in the literature to aid novices or semi-experts in a variety of data-management tasks. The focus of PBE is to not only solve the task, but also provide the *mechanism* that can solve the task. To this end, all PBE tools learn from the user examples and synthesize programs that can produce the desired results. To help data scientists write complex code for data wrangling and data transformation, WREX [77] proposes an example-driven program synthesis approach. To enable integration of web data with spreadsheets, WebRelate [157] facilitates joining semi-structured web data with relational data in spreadsheets using input-output examples. FlashRelate [31] and FlashExtract [198] enable extraction of relational data from semi-structured spreadsheets, text files, and web pages, using examples. Data-transformation-by-example approaches [124, 135] led to the development of the FlashFill [106] feature in Microsoft Excel, which can learn the user's data transformation intent only from a few examples. Live programming [285] helps novice programmers to understand their code, where they can manipulate the input by directly editing the code and manipulate the output by providing examples of the desired output. Beyond computational tasks, PBE tools also support creative tasks such as music creation by example [109], where a software takes a song as an example and allows the user to interactively mix the AI-generated music.

2.1.2 Query by Example (QBE)

QBE was an early effort to assist users without SQL expertise in formulating SQL queries [352]. Existing QBE systems [260, 294] identify relevant relations and joins in situations where the user lacks schema understanding, but are limited to project-join queries.

These systems focus on the common structure of the example tuples, and do not try to learn the common semantics as SQuID does. QPlain [72] uses user-provided provenance of the example tuples to learn the join paths and improve intent inference. However, this assumes that the user understands the schema, content, and domain to provide these provenance explanations, which is often unrealistic for nonexperts.

2.1.3 Set Expansion

Set expansion is a problem corresponding to QBE in Knowledge Graphs [324, 331, 347]. SPARQLByE [73], built on top of a SPARQL QRE system [14], allows querying RDF datasets by annotated (positive/negative) example tuples. In semantic knowledge graphs, systems address the problem of entity set expansion using maximal-aspect-based entity model, semantic-feature-based graph query, and entity co-occurrence information [133, 160, 207, 232]. These approaches exploit the semantic context of the example tuples, but they cannot learn new semantic properties, such as aggregates involving numeric values, that are not explicitly stored in the knowledge graph, and they cannot express complex semantic properties without exploding the graph size.

2.1.4 Interactive Approaches

Interactive approaches rely on relevance feedback on system-generated tuples to improve query inference and result delivery [2, 44, 74, 117, 203]. Such systems typically expect a large number of interactions, and are often not suitable for nonexperts who may not be sufficiently familiar with the data to provide effective feedback.

2.1.5 Query Reverse Engineering (QRE)

QRE [27, 328] is a special case of QBE, which assumes that the provided examples comprise the complete output of the intended query. Because of this closed-world assumption, QRE systems can build data classification models on denormalized tables [312], labeling the provided tuples as positive examples and the rest as negative. Such methods

are not suitable for our setting, because we operate with few examples, under an open-world assumption. While few QRE approaches [175] relax the closed world assumption (known as the *superset QRE* problem) they are also limited to PJ queries similar to the existing QBE approaches. Most QRE methods are limited to narrow classes of queries, such as PJ [175, 345], aggregation without joins [307], or top-k queries [250]. REGAL+ [308] handles SPJA queries but only considers the schema of the example tuples to derive the joins and ignores other semantics. A few QRE methods target expressive SPJ queries [322, 346], but they only work for very small databases (< 100 cells), and do not scale to the datasets used in our evaluation. Moreover, the user needs to specify the data in their entirety, thus, expecting complete schema knowledge, while SCYTHE [322] also expects hints from the user towards precise discovery of the constants of the query predicates.

2.1.6 Alternative Approaches

Alternative approaches exist, beyond by-example methods, to aid novice users explore relational databases. Keyword-based search [8, 149, 344] allows accessing relational data without knowledge of the schema and SQL syntax, but does not facilitate search by examples. Other notable systems that aim to assist novice users in data exploration and complex query formulation are: QueRIE, a query recommendation based on collaborative filtering [83], SnipSuggest, a context-aware SQL autocompletion system [184], SQL-Sugg, a keyword-based query suggestion system [87], YmalDB, a "you-may-also-like"-style data exploration system [78], and SnapToQuery, an exploratory query specification assistance tool [165]. These approaches focus on assisting users in query formulation, but assume that the users have sufficient knowledge about the schema and the data. VIDA [199], ShapeSearch [298], and Zenvisage [297] are visual query systems that allow visual data exploration, but they require the user to be aware of the trend within the output. Some approaches exploit user interaction to assist users in query formulation and result delivery [2, 44, 74, 117, 203]. There, the user has to provide relevance feedback on system-

generated tuples. However, such highly interactive approaches are not suitable for data exploration as users often lack knowledge about the system-provided tuples, and, thus, fail to provide correct feedback reflecting their query intent. Moreover, such systems often require a large number of user interactions. User-provided examples and interactions appear in other problem settings, such as learning schema mappings [45, 266, 287]. The query likelihood model in IR [222] resembles our technique, but does not exploit the similarity of the input entities.

2.1.7 Machine Learning

Machine learning methods can model QBE settings as classification problems, and relational machine learning targets relational settings in particular [119]. However, while the provided examples serve as positive labels, QBE settings do not provide explicit negative examples. Semi-supervised statistical relational learning techniques [334] can learn from unlabeled and labeled data, but require unbiased sample of negative examples. There is no straightforward way to obtain such a sample in our problem setting without significant user effort.

Our problem setting is better handled by one-class classification [182, 221], more specifically, Positive and Unlabeled (PU) learning [34, 35, 84, 210, 237, 340], which learns from positive examples and unlabeled data in a semi-supervised setting [55]. Most PU-learning methods assume denormalized data, but relational PU-leaning methods do exist. However, all PU-learning methods rely on one or more strong assumptions [35] (e.g., all unlabeled entities are negative [244], examples are selected completely at random [33, 84], positive and negative entities are naturally separable [210, 302, 340], similar entities are likely from the same class [183]). These assumptions create a poor fit for our problem setting where the example set is very small, it may exhibit user biases, real-time response is required, and intents may involve complex semantic similarity.

2.1.8 User Study of PBE Approaches

Drosos et al. [77] present a comparative user study contrasting WREX against manual programming. The study results indicate that data scientists are more effective and efficient at data wrangling with WREX over manual programming. Mayer et al. [226] presents comparative study between two user interaction models—program navigation and conversational clarification—that can help resolve the ambiguities in the examples in by-example interaction models. Lee et al. [200] presents an online user study on how PBE systems help the users solve complex tasks. They identify seven types of mistakes commonly made by the users while using PBE systems, and also suggest an actionable feedback mechanism based on unsuccessful examples. Santolucito et al. [284] studied the impact of PBE on real-world users over a tool for shell scripting by example. Their study results indicate that while the users are quicker to solve the task using the PBE tool, they trust the traditional approach more. However, none of these studies focus on QBE in particular, which is a PBE system tailored towards data exploration over relational databases. The performance of a QBE tool is affected by additional factors, such as the subjectivity of the data exploration task and the domain knowledge of the user. Moreover, traditional data access and exploration methods pose hurdles not only to novices, but to expert users as well. These factors indicate the need for a new study that targets QBE systems in particular and are the motivation behind our comparative user studies involving SQUID.

2.1.9 Personalized Document Summarization

Document summarization is a well-studied problem in the area of natural language processing. Beyond generic document summarization, a special class of problem is intent-specific document summarization. Our focus is personalized, intent-specific document summarization. In *query-based* summarization, users specify their intent in the form of an unstructured query: typically, a natural language question. Some approaches use hints that represent user interest. Such hints take different forms, such as user-provided anno-

tations [238], vision-based eye-tracking [336], user history and collaborative social influences [273], and so on. Early approaches to sentence selection would score each sentence based on some criteria and return the top-k sentences as a summary. This would often lead to the inclusion of redundant sentences. To tackle the issue of redundancy, later work [130] followed an ad hoc iterative greedy approach, leading to suboptimal summaries. While some approaches try to iteratively refine the summary quality [11], they are mostly based on heuristic approaches, e.g., A* search, that still do not guarantee optimality. Lin and Bilmes [209] provide an integer-programming formulation with constraints and objectives involving general sentence score, diversity, and summary length, but with no connection to the user-provided examples. Also, because of the combinatorially large number of possible summaries, the formulation in [209] cannot generally scale to large dataset sizes.

2.1.10 Dataset for Subjective Document Summarization

Several datasets exist for generic summarization tasks, including the CNN/Daily Mail dataset [242] which contains 300,000 news article-summary pairs, Webis-TLDR-17, which contains three million document-summary pairs extracted from Reddit forums [321], Multi-News dataset, which is a multi-document summarization dataset containing over 50,000 articles-summary pairs [85], and the Gigaword [280] and X-Sum [243] datasets, both of which contain single-sentence summaries of news articles.

ScisummNet [338] is a manually annotated corpus for scientific papers on computational linguistics to generate summaries that include the articles' impacts on the research community. TalkSumm [202] is for scientific paper summarization based on conference talks. However, it requires additional information (corresponding video of the paper presentation) and does not consider personalization, where different audience might want different summaries of the same paper. In general, none of the above datasets are suitable for the task of subjective summarization, which is our focus.

A task close to ours is *query* or *topic-based* extractive summarization. Suitable datasets include DUC 2004, DUC 2005, and DUC 2006, which contain query-based multi-document summaries [82]. Webis-Snippet-20 is another dataset of 10 million web pages together with their query-based, abstractive snippets [57]. In these datasets, each document (or set of documents) has exactly one associated summary that corresponds to a single query. In contrast, SUBSUME contains multiple summaries of each document corresponding to different intents. Furthermore, each document, intent pair is summarized by multiple individuals.

Frermann et al. [108], in the context of "aspect-based" summarization, provide a dataset having multiple topic-focused summaries for each document. The dataset is synthetic, however, and does not involve human annotators. To the best of our knowledge, SUBSUME is the first human-generated dataset for subjective, extractive document summarization, where interpretation of intents vary across individuals.

2.2 Data Understanding: Conformance Constraints

We first discuss existing data profiling primitives and why they fall short in modeling arithmetic relationships involving numerical data attributes, which we use to solve the problem of trusted machine learning. We proceed to discuss other approaches that also target the problem of trusted machine learning and few other related work that address similar problems. We also discuss prior work that address the similar application areas of conformance constraints that we consider.

2.2.1 Data Profiling

There is extensive literature on data-profiling primitives [1] that model relationships among data attributes, such as unique column combinations [136], functional dependencies (FD) [251, 349] and their variants (metric [187], conditional [90], soft [155], approximate [150, 190], relaxed [53], etc.), differential dependencies [299], order dependencies [197, 305], inclusion dependencies [223, 252], denial constraints [42, 61, 215, 258],

and statistical constraints [337]. However, none of them focus on learning approximate arithmetic relationships that involve multiple numerical attributes in a noisy setting, which is the focus of our work. Soft FDs [155] model correlation and generalize traditional FDs by allowing uncertainty, but are limited in modeling relationships between only a pair of attributes. Metric FDs [187] allow small variations in the data, but the existing work focuses on verification only and not discovery of metric FDs. Some variants of FDs [53, 150, 187, 190] consider noisy setting, but they require the allowable noise parameters to be explicitly specified by the user. However, determining the right settings for these parameters is nontrivial. Most existing approaches treat constraint violation as Boolean, and do not measure the degree of violation. In contrast, we do not require any explicit noise parameter and provide a way to quantify the degree of violation of conformance constraints.

2.2.2 Trusted AI

The issue of trust, resilience, and interpretability of artificial intelligence (AI) systems has been a theme of increasing interest recently [162, 286, 319], particularly for high-stake and safety-critical data-driven AI systems [311, 320]. A standard way to decide whether to trust a classifier or not, is to use the classifier-produced confidence score. However, as a prior work [164] argues, this is not always effective since the classifier's confidence scores are not well-calibrated. Moreover, unlike classifiers, regressors lack a natural way to produce such confidence scores. While some recent techniques [71, 138, 164, 289] aim at validating the inferences made by machine-learned models on unseen tuples, they usually require knowledge of the inference task, access to the model, and/or expected cases of data shift, which we do not. Furthermore, they usually require costly hyper-parameter tuning and do not generate closed-form data profiles like conformance constraints. To evaluate model performance, regression diagnostics check if the assumptions made by the model

during training are still valid for the serving data. However, they require knowledge of the ground-truths for the serving data, which is often unavailable.

2.2.3 Data Drift

Prior work on data drift, change detection, and covariate shift [7, 50, 66, 68, 76, 86, 143, 146, 153, 180, 185, 292, 300] relies on modeling data distribution. However, data distribution does not capture constraints, which is the primary focus of our work. Instead of detecting drift globally, only a handful of work model local concept-drift [314] or drift for imbalanced data [323]. Few data-drift detection mechanisms rely on availability of classification accuracy [39, 111, 114, 281] or classification "blindspots" [293]. Some of these work focus on adapting change in data, i.e., learning in an environment where change in data is expected [39, 115, 248, 303, 341]. Such adaptive techniques are useful to obtain better performance for specific tasks; however, their goal is orthogonal to ours.

2.2.4 Data Cleaning

Integrity constraints and functional dependencies have long been used for error detection and data cleaning [43, 88, 89]. Holistic data cleaning [62] provides a unified framework to allow different types of user-provided constraints for data cleaning. Instead of relying on user-provided rules, a practical idea is to automatically discover them from clean data. ANMAT [264] exploits automatically discovered pattern functional dependencies for error detection, but it is limited to text attributes. In summary, none of the existing efforts in data cleaning consider automatically generated constraints involving linear arithmetic expressions over numerical attributes, which is the primary focus of conformance-constraint-driven data cleaning.

2.2.5 Explaining Outliers

Our explanation tool ExTuNe is similar to outlier-detection approaches [188] that define outliers as the ones that deviate from a generating mechanism such as local correla-

tions. Pattern-based outlier detection mechanisms [264] exploit patterns, similar to ours, to detect outliers, but they do not explain outliers. Scorpion [332] explains outliers through common explanation, but does not consider relationships among attributes. In general, no prior work on explaining outliers [233, 234, 332] addresses the question: "which attributes or attribute relationships are responsible for non-conformance?" Data drift and covariate shift [268] detection has connection to non-conformance detection. However, their techniques are based on multivariate distribution modeling, without emphasizing low-variance dimensions, and provide poor interpretability.

2.3 Explanation Frameworks for Debugging

We now provide an overview of prior art related to our explanation frameworks to assist in debugging various components of data systems.

2.3.1 Causal Inference

Causal inference has been long applied for root-cause analysis of program failures [18, 56, 128]. Attariyan et al. [16, 17] observe causality within application components through runtime control and data flow; but only report a list of root causes ordered by the likelihood of being faulty, without providing further causal connection between root causes and performance anomalies. Beyond statistical association (e.g., correlation) between root cause and failure, few techniques [19, 20, 105, 295] apply statistical causal inference on observational data towards software fault localization. However, observational data collected from program execution logs is often limited in capturing certain scenarios, and hence, observational study is ill-equipped to identify the intermediate explanation predicates. This is because observational data is not generated by randomized controlled experiments, and therefore, may not satisfy *conditional exchangeability* (data can be treated as if they came from a randomized experiment [161]) and *positivity* (all possible combinations of values for the variables are observed in the data), which are two key requirements for applying

causal inference on observational data [295]. While observational studies are extremely useful in many settings, AID's problem setting permits interventional studies, which offer increased reliability and accuracy.

Causal inference techniques are also used software testing [107, 121, 144, 169, 343]. However, they use a white-box strategy or are application-specific. Causal relational learning [282] infers causal relationships in relational data, but it does not seek mismatches between the data and the systems. Our work DATAEXPOSER shares similarity with BugEx [278], which generates test cases to isolate root causes. However, it assumes complete knowledge of the program and data-flow paths.

2.3.2 Explanation-centric Approaches

Explanation-centric approaches are relevant to AID as they also aim at generating informative, yet minimal, explanations of certain incidents, such as data errors [325], query results [24, 60], and binary outcomes [118]. Some work find causes of errors in data generation processes [325], while others discover relationships among attributes [24, 118], and across datasets [60]. These, however, do not focus on interventions. Viska [123] allows the users to perform intervention on system parameters to understand the underlying causes for performance differences across different systems. None of these systems are applicable for finding causally connected paths that explain intermittent failures due to concurrency bugs. ExceLint [29] exploits the spatial structure of spreadsheets to look for erroneous formulas. Unlikely interventional efforts, these approaches operate on observational data, and do not generate additional test cases.

Machine learning interpreters [276, 277] perturb testing data to learn a surrogate for models. Their goal is not to find mismatch between data and models. Debugging methods for ML pipelines are similar to data explanation [48, 51], where training data may cause model's underperformance. Varma et al. [318] and Kulesza et al. [192] discuss principled ways to find reasons of malfunctions. Wu et al. [333] allow users to complain about outputs

of SQL queries, and present data points whose removal resolves the complaints. Schelter et al. [289] validate when a model fails on certain datasets, but assume knowledge of the mechanism that corrupts the data. We aim to find discriminative predicates (profiles) among program executions (datasets) without any such knowledge.

2.3.3 Statistical Debugging

Statistical debugging approaches [59, 167, 179, 179, 205, 212, 310, 350] employ statistical diagnosis to rank program predicates based on their likelihood of being the root causes of program failures. However, all statistical debugging approaches suffer from the issue of not separating correlated predicates from the causal ones, and fail to provide contextual information regarding *how* the root causes lead to program failures.

2.3.4 Interventional Debugging

Despite using interventional approach to blame runtime conditions of a program for causing failure, AID [96] is limited to software bugs and does not intervene on datasets. BugDoc [216] finds parameter settings in a black-box pipeline as root causes of pipeline failure; but it only reports whether a dataset is a root cause and does not explain why a dataset causes the failure. CADET [158] uses causal inference to derive root causes of non-functional faults for hardware platforms focused on performance issues. Capuchin [283] casts fairness in machine learning as a database repair problem and adds or removes rows in the training data to simulate a *fair world*; but it does not aim to find cause of unfairness.

2.3.5 Data Debugging

Porting concepts of debugging from software to data has gained attention in data management community [47, 240]. Dagger [274, 275] provides data debugging primitives for white-box interactions with data-driven pipelines. CheckCell [30] ranks data cells that unusually affect output of a given target. However, it is not meant for large datasets where single cells are unlikely to causes malfunction. Moreover, CheckCell cannot expose combi-

nation of root causes. DATAEXPOSER is general-purpose, application-agnostic, and interventional, providing causally verified issues that cause mismatch between data and system.

2.3.6 Group Testing

Group testing [5, 21, 25, 79, 151, 178, 204] has been applied for fault diagnosis in prior literature [351]. Specifically, adaptive group testing is related to both AID and DATAEXPOSER's intervention algorithms. However, none of the existing work considers the scenario where a group test might reveal additional information and thus offers an inefficient solution for causal path discovery.

2.3.7 Extracting Predicates

To encode runtime events, AID uses predicates that are extracted from execution traces of the application. Ball et al. [26] provide algorithms for efficiently tracing execution with minimal instrumentation. While the authors had a different goal (i.e., path profiling) than ours, the traces can be used to extract AID predicates.

2.3.8 Fault Injection

Fault injection techniques [12, 134, 176, 224] intervene application runtime behavior with the goal to test if an application can handle the injected faults. In fault injection techniques, faults to be injected are chosen based on whether they can occur in practice. In contrast, AID intervenes with the goal of verifying (presence or absence of) causal relationship among runtime predicates, and faults are chosen based on if they can alter selected predicates.

2.3.9 Control-flow Graph

Control flow graph-based techniques [58, 166] aim at identifying bug signature for sequential programs using discriminative subgraphs within the program's control flow graph;

or generating faulty control flow paths that link many bug predictors. But these approaches do not consider causal connection among these bug predictors and program failure.

2.3.10 Differential Slicing

Differential slicing [170] aims towards discovering causal path of execution differences but requires complete program execution trace generated by execution indexing [335]. Dual slicing [327] is another program slicing-based technique to discover statement level causal paths for concurrent program failures. However, this approach does not consider compound predicates that capture certain runtime conditions observed in concurrent programs. Moreover, program slicing-based approaches cannot deal with a set of executions, instead they only consider two executions—one successful and one failed.

We now proceed to present the work accomplished in this thesis that build on and enrich the literature.

PART I: ENHANCING USABILITY: EXAMPLE-DRIVEN INTENT DISCOVERY

CHAPTER 3

QUERYING RELATIONAL DATABASES BY EXAMPLE: SEMANTIC-SIMILARITY-AWARE QUERY INTENT DISCOVERY (SQUID)

Like many computational systems, traditional database technology was not designed with the group of nonexpert users in mind, and, hence, poses hurdles to them. Traditional query interfaces allow data retrieval through well-structured queries, and to write such queries, one needs expertise in the query language (typically SQL) and knowledge of the potentially complex database schema. Unfortunately, nonexpert users typically lack both. *Programming by example* (PBE) has been explored as a method to bridge the usability gap of computational systems that typically require precise programs from users via example-based interactions. Under the PBE paradigm, instead of writing a precise program to specify their intent, users only need to provide a few examples of the result (i.e., program output) they desire [65, 126, 206, 285].

PBE has also been explored in the context of retrieving and exploring relational data, which led to the development of *query by example* (QBE) systems [72, 260, 294]. QBE offers an alternative data retrieval mechanism, where users specify their intent by providing example tuples for their query output [239]. Unfortunately, traditional QBE systems [72, 260, 294] for relational databases make a strong and oversimplified assumption in modeling user intent: they implicitly treat the structural similarity and data content of the example tuples as the only factors specifying query intent. As a result, they consider all queries that contain the provided example tuples in their result set as equally likely to represent the desired intent. This ignores the richer context in the data that can help identify the intended query more accurately. While more nuanced QBE systems exist, they typically

academics

id	name
100	Thomas Cormen
101	**Dan Suciu**
102	Jiawei Han
103	**Sam Madden**
104	James Kurose
105	Joseph Hellerstein

research

aid	interest
100	algorithms
101	data management
102	data mining
103	data management
103	distributed systems
104	computer networks
105	data management
105	distributed systems

Figure 3.1: Excerpt of two relations of the `CS Academics` database. `Dan Suciu` and `Sam Madden` (in bold), both have research interests in `data management`.

place additional requirements or significant restrictions over the supported class of queries (see Appendix A.1).

Example 3.1. *In Figure 3.1, the relations* `academics` *and* `research` *store information about CS researchers and their research interests. Given the user-provided set of examples* `{Dan Suciu, Sam Madden}`*, a human can posit that the user is likely looking for researchers in the area of* `data management`*. However, a QBE system that looks for queries only based on the structural similarity of the examples produces* `Q1` *to capture the query intent, which is too general:*

```
Q1: SELECT name FROM academics
```

In fact, the QBE system will generate the same generic query `Q1` *for any set of names from the relation* `academics`*. Even though the intended semantic context is present in the data (by associating academics with research interest information using the relation* `research`*), existing QBE systems fail to capture it. A more specific query that better represents the semantic similarity among the example tuples is* `Q2`*:*

```
Q2: SELECT name FROM academics, research
    WHERE research.aid = academics.id AND
          research.interest = 'data management'
```

Example 3.1 shows how reasoning about the semantic similarity of the example tuples can guide the discovery of the correct query structure (join of the `academics` and `research` tables), as well as the discovery of the likely intent (research interest in `data management`).

We can often capture semantic similarity through direct attributes of the example tuples. These are attributes associated with a tuple within the same relation, or through simple key-foreign key joins (such as research interest in Example 3.1). Direct attributes capture intent that is *explicit*, precisely specified by the particular attribute values. However, sometimes query intent is more vague, and is not expressible by explicit semantic similarity alone. In such cases, the semantic similarity of the example tuples is *implicit*, which can be captured through deeper associations with other entities in the data (e.g., genre and number of movies an actor appears in).

Example 3.2. *The IMDb dataset contains a wealth of information related to the film and entertainment industry. We query the IMDb dataset (Figure 3.2) with a traditional QBE system (e.g., [294]), using two different sets of examples:*

```
ET1 = {Arnold Schwarzenegger, Sylvester Stallone, Dwayne Johnson}
ET2 = {Eddie Murphy, Jim Carrey, Robin Williams}
```

`ET1` and `ET2` contain the names of three actors from two public lists of "physically strong" actors [259] and "funny" actors [110], respectively. While `ET1` and `ET2` represent different query intents ("strong" actors and "funny" actors, respectively), a standard QBE system produces the same generic query for both:

```
Q3: SELECT person.name FROM person
```

Explicit semantic similarity cannot capture these different intents, as there is no attribute that explicitly characterizes an actor as "strong" or "funny". Nevertheless, the database encodes these associations implicitly, in the number and genre of movies an actor appears in ("strong" actors frequently appear in action movies, and "funny" actors in comedies).

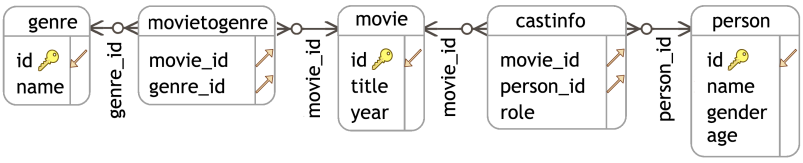


Figure 3.2: Partial schema of the IMDb database. The schema contains two entity relations: `movie` and `person`; and a semantic property relation: `genre`. The relations `castinfo` and `movietogenre` associate entities and semantic properties.

Standard QBE systems typically produce queries that are too general, and, thus, fail to capture nuanced query intents, such as the ones in Examples 3.1 and 3.2. Some prior approaches attempt to refine the queries based on additional external information, such as external ontologies [207], provenance information of the example tuples [72], and user feedback on multiple (typically a large number of) system-generated examples [44, 74, 203]. Other work relies on a *closed-world* assumption—where a tuple not specified as an example output is assumed to be excluded from the query result—to produce more expressive queries [203, 322, 346], and, thus, requires complete examples of input databases and output results. Providing such external information is typically complex and tedious for nonexpert users.

In contrast with prior approaches, in this thesis, we propose a method and present an end-to-end system for discovering query intent effectively and efficiently, in an open-world setting, without the need for any additional external information beyond the initial set of example tuples. While Figure A.1 provides a summary exposition of prior work, and contrasts with our contributions. SQuID, our semantic-similarity-aware query intent discovery framework, relies on two key insights: (1) It exploits the information and associations already present in the data to derive the explicit and implicit similarities among the provided examples. (2) It identifies the significant semantic similarities among them using *abductive reasoning*, a logical inference mechanism that aims to derive a query as the simplest and most likely explanation of the observation (example tuples). We proceed to explain how SQuID uses these insights to handle the challenging scenario of Example 3.2 next.

Example 3.3. *We query the IMDb dataset with* SQuID*, using the example tuples of* `ET2` *(Example 3.2).* SQuID *discovers the following semantic similarities among the examples: (1) all are* `Male`*, (2) all are* `American`*, and (3) all appeared in more than 40* `Comedy` *movies. Out of these properties,* `Male` *and* `American` *are very common in the IMDb database. In contrast, a very small fraction of persons in the dataset are associated with such a high number of* `Comedy` *movies; this means that it is unlikely for this similarity to be coincidental, as opposed to the other two. Based on abductive reasoning,* SQuID *selects the third semantic similarity as the best explanation of the observed example tuples, and produces the query:*

```
Q4: SELECT person.name
    FROM person, castinfo, movietogenre, genre
    WHERE person.id = castinfo.person_id AND
          castinfo.movie_id = movietogenre.movie_id AND
          movietogenre.genre_id = genre.id AND
          genre.name = 'Comedy'
    GROUP BY person.id
    HAVING count(*) >= 40
```

We make the following contributions:

- We design SQuID: an end-to-end system that automatically formulates select-project-join queries with optional group-by aggregation and intersection operators (SPJ_{AI}) based on few user-provided example tuples (Section 3.1). SQuID does not require the users to have any knowledge of the database schema or the query language. Unlike existing approaches, SQuID does not require any additional information from the user, beyond the example tuples.

- SQuID infers *semantic similarities* of the examples and models query intent using a collection of *basic* and *derived* semantic property *filters* (Section 3.2). While some

prior work explored the use of semantic similarity in knowledge graphs [160, 232, 347], they do not directly apply to the relational domain, as they do not model implicit semantic similarities derived from aggregating properties of affiliated entities (e.g., number of comedy movies an actor appears in).

- We express the problem of query intent discovery using a *probabilistic abduction model* (3.3). This model allows SQuID to identify the semantic property filters that represent the most likely intent, given the examples.

- SQuID achieves real-time performance through an offline strategy that precomputes semantic properties and related statistics to construct an *abduction-ready* database (Section 3.4). During the online phase, SQuID consults the abduction-ready database to derive relevant semantic property filters, based on the provided examples, and applies abduction to select the optimal set of filters towards query intent discovery (Section 3.5).

- Our empirical evaluation includes three real-world datasets, 41 queries covering a broad range of complex intents and structures, and three case studies (Section 3.6). We further compare SQuID with TALOS [312], a state-of-the-art query reverse engineering system that supports very expressive queries, but in a closed-world setting. We show that SQuID is more accurate at capturing intents and infers better queries, often reducing the number of predicates by orders of magnitude. We also empirically show that SQuID outperforms a semi-supervised positive and unlabeled learning system [84].

- We present results of two comparative user studies—a controlled experiment study and an interview study—contrasting SQuID with traditional SQL querying (Section 3.7). Our analysis of the controlled experiment study shows that participants were significantly more *effective* (achieved more accurate results) and *efficient* (required less time and fewer attempts) over a diverse set of data-exploration tasks us-

ing SQUID compared to SQL. Qualitative feedback from the interviewees confirms that SQUID eliminates SQL challenges and assists the users in effective data exploration. While our results validate some findings of prior studies over other PBE approaches [284], we contribute new empirical insights gained from our studies that indicate that even a limited level of domain expertise (knowledge of a small subset of the desired data) can substantially help overcome the lack of technical expertise (knowledge of SQL and schema) in data exploration.

3.1 SQUID Overview

In this section, we first discuss the challenges in example-driven query intent discovery and highlight the shortcomings of existing approaches. We then formalize the problem of query intent discovery using a probabilistic model and describe how SQUID infers the most likely query intent using abductive reasoning. Finally, we present the system architecture for SQUID, and provide an overview of our approach.

3.1.1 The Query Intent Discovery Problem

SQUID aims to address three major challenges that hinder existing QBE systems:

Large search space. Identifying the intended query, given a set of examples, can involve a huge search space of potential candidate queries. Aside from enumerating the candidate queries, validating them is expensive, as it requires executing the queries over potentially very large data. Existing approaches limit their search space in three ways: (1) They often focus on project-join (PJ) queries only. Unfortunately, ignoring selections severely limits the applicability of these solutions. (2) They assume that the user provides a large number of examples or interactions, which is often unreasonable in practice. (3) They make a closed-world assumption, thus needing complete sets of input data and output results. In contrast, SQUID focuses on a much larger and more expressive class of queries,

select-project-join queries with optional group-by aggregation and intersection operators (SPJ_{AI})[1], and is effective in the open-world setting with very few examples.

Distinguishing candidate queries. In most cases, a set of example tuples does not uniquely identify the target query, i.e., there are multiple valid queries that contain the example tuples in their results. Most existing QBE systems do not distinguish among the valid queries [294] or only rank them according to the degree of input containment, when the example tuples are not fully contained by the query output [260]. In contrast, SQuID exploits the semantic context of the example tuples and ranks the valid queries based on a probabilistic abduction model of query intent.

Complex intent. A user's information need is often more complex than what is explicitly encoded in the database schema (e.g., Example 3.2). Existing QBE solutions focus on the query structure, and, thus, are ill-equipped to capture nuanced intents. While SQuID still produces a structured query in the end, its objectives focus on capturing the semantic similarity of the examples, both explicit and implicit. SQuID thus draws a contrast between the traditional query-by-example problem, where the query is assumed to be the hidden mechanism behind the provided examples, and the *query intent discovery problem* that we focus on in this work.

We proceed to formalize the problem of query intent discovery. We use $\mathcal{D}$ to denote a database, and $Q(\mathcal{D})$ to denote the set of tuples in the result of query Q operating on $\mathcal{D}$. We aim to discover an SPJ_{AI} query Q that contains E within its result set and maximizes the query posterior, i.e., the conditional probability $Pr(Q \mid E)$.

Definition 3.1 (Query Intent Discovery)**.** *For a database $\mathcal{D}$ and a user-provided example tuple set E, the query intent discovery problem is to find an SPJ_{AI} query Q such that: (1) $E \subseteq Q(\mathcal{D})$, and (2) $Q = \text{argmax}_q \, Pr(q \mid E)$.*

[1] SQuID considers queries with key-foreign key joins, and conjunctive selection predicates of the form ⟨`attribute OP constant`⟩, where `OP` $\in \{=, \geq, \leq\}$.

3.1.2 Abductive Reasoning

SQUID solves the query intent discovery problem (Definition 3.1) using *abduction*. Abduction or abductive reasoning [15, 37, 174, 230] refers to the method of inference that finds the best explanation (query intent) of an often incomplete observation (example tuples). Unlike deduction, in abduction, the premises do not guarantee the conclusion. A deductive approach would produce all possible queries that contain the example tuples in their results, guaranteeing that the intended query is one of them. However, the set of valid queries can be extremely large, growing exponentially with the number of properties and the size of the data domain. Hence, we model query intent discovery as an abduction problem and apply abductive inference to discover the most likely query intent. For example, given two possible candidate queries, Q and Q', we infer Q as the intended query if $Pr(Q \mid E) > Pr(Q' \mid E)$.

Example 3.4. *In the scenario of Example 3.1,* SQUID *identifies that the two example tuples share the semantic context* `interest = 'data management'`. *While* `Q1` *and* `Q2` *both contain the examples tuples in their result set, the probability that two tuples drawn randomly from the output of* `Q1` *would display the identified semantic context is low (*$(\frac{3}{7})^2 \approx 0.18$ *in the data excerpt). In contrast, the probability that two tuples drawn randomly from the output of* `Q2` *would display the same semantic context is high (*1.0*). Assuming equal priors for* `Q1` *and* `Q2`*, from Bayes' rule:* $Pr(\texttt{Q2} \mid E) > Pr(\texttt{Q1} \mid E)$.

3.1.3 Solution Sketch

At the core of SQUID is an *abduction-ready database*, αDB (Figure 3.3). The αDB serves two purposes: (1) it increases SQUID's efficiency by storing precomputed associations and statistics, and (2) it simplifies the query model by reducing the extended family of SPJ_{AI} queries on the original database to equivalent SPJ queries on the αDB.

Example 3.5. *The IMDb database has, among others, relations* `person` *and* `genre` *(Figure 3.2).* SQUID*'s αDB stores a derived semantic property that associates the two en-*

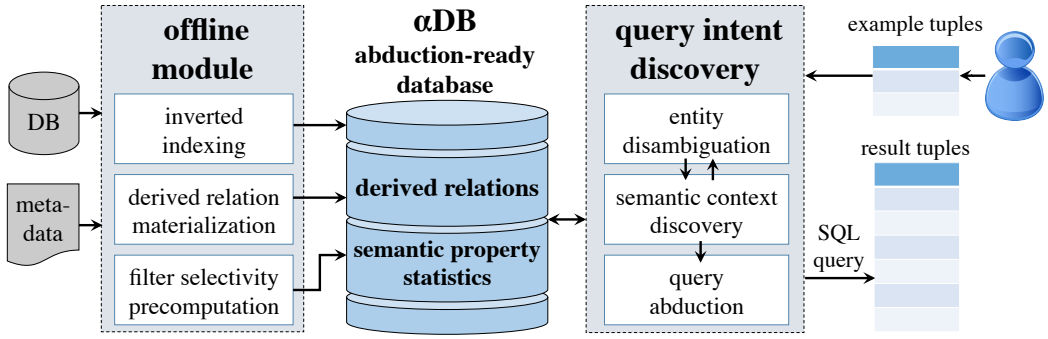


Figure 3.3: SQuID's operation includes an offline module, which constructs an *abduction-ready* database (αDB) and precomputes statistics of semantic properties. SQuID's query intent discovery module interacts with the αDB to identify the semantic context of the user-provided example tuples and abduces the most likely query intent.

tity types in a new relation, `persontogenre(person.id, genre.id, count)`*, which stores the number of movies of each genre each person appeared in.* SQuID *derives this relation through joins with* `castinfo` *and* `movietogenre`*, followed by aggregation (Figure 3.4). Then, the SPJ$_{AI}$ query* `Q4` *(Example 3.3) is equivalent to the simpler SPJ query* `Q5` *on the αDB:*

```
Q5: SELECT person.name
    FROM person, persontogenre, genre
    WHERE person.id = persontogenre.person_id AND
          persontogenre.genre_id = genre.id AND
          genre.name = 'Comedy' AND persontogenre.count >= 40
```

By incorporating aggregations in precomputed, derived relations, SQuID can reduce SPJ$_{\text{AI}}$ queries on the original data to SPJ queries on the αDB. SQuID starts by inferring a PJ query, Q^*, on the αDB as a query template; it then augments Q^* with selection predicates, driven by the semantic similarity of the examples.

Organization. Section 3.2 formalizes SQuID's model of query intent as a combination of the base query Q^* and a set of semantic property filters. Then, Section 3.3 analyzes the probabilistic abduction model that SQuID uses to solve the query intent discovery problem

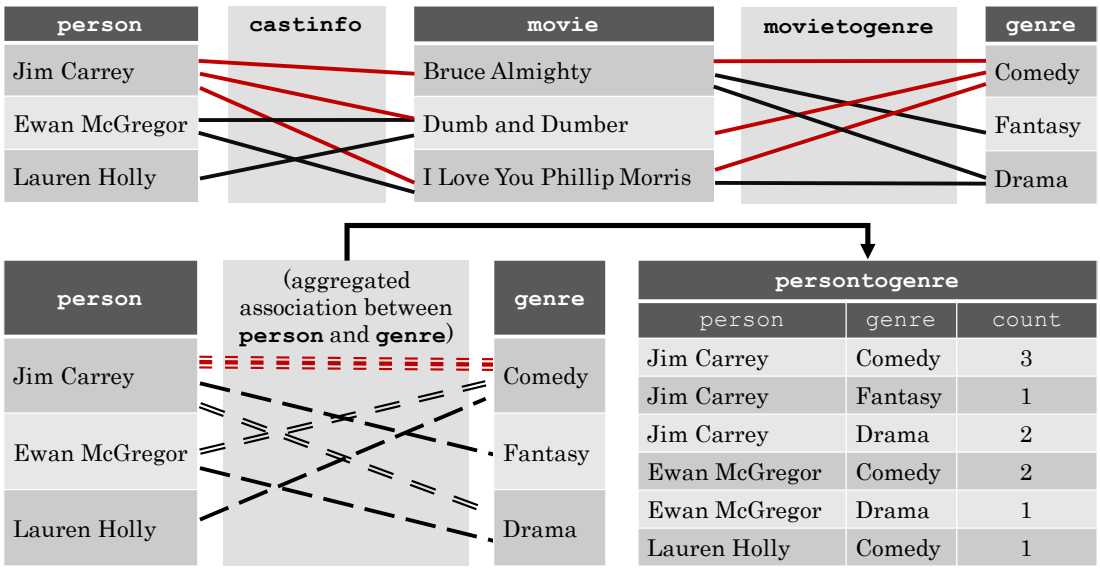


Figure 3.4: A `genre` value (e.g., `genre=Comedy`) is a basic semantic property of a `movie` (through the `movietogenre` relation). A `person` is associated with `movie` entities (through the `castinfo` relation); aggregates of basic semantic properties of movies are *derived semantic properties* of `person`, e.g., the number of comedy movies a person appeared in. The αDB stores the derived property in the new relation `persontogenre`. (For ease of exposition, we depict attributes `genre` and `person` instead of `genre.id` and `person.id`.)

(Definition 3.1). After the formal models, we describe the system components of SQUID. Section 3.4 describes the offline module, which is responsible for making the database abduction-ready, by precomputing semantic properties and statistics in derived relations. Section 3.5 describes the query intent discovery module, which abduces the most likely intent as an SPJ query on the αDB.

3.2 Modeling Query Intent

SQUID's core task is to infer the proper SPJ query on the αDB. We model an SPJ query as a pair of a base query and a set of semantic property filters: $Q^{\varphi}=(Q^{*}, \varphi)$. The *base query* Q^{*} is a project-join query that captures the structural aspect of the example tuples. SQUID can handle examples with multiple attributes, but, for ease of exposition, we focus on example tuples that contain a single attribute of a single entity (`name` of `person`).

Notation	Description
$p = \langle A, V, \theta \rangle$	Semantic property defined by attribute A, value V, and association strength θ
ϕ_p or ϕ	Semantic property filter for p
$\Phi = \{\phi_1, \phi_2, \dots\}$	Set of minimal valid filters
$Q^{\varphi} = (Q^*, \varphi)$	SPJ query with semantic property filters $\varphi \subseteq \Phi$ applied on base query Q^*
$x = (p, \lvert E \rvert)$	Semantic context of E for p
$\mathcal{X} = \{x_1, x_2, \dots\}$	Set of semantic contexts

Figure 3.5: Summary of notations.

In contrast to existing approaches that derive PJ queries from example tuples, the base query in SQuID does not need to be minimal with respect to the number of joins: while a base query on a single relation with projection on the appropriate attribute (e.g., `Q1` in Example 3.1) would capture the structure of the examples, the semantic context may rely on other relations (e.g., `research`, as in `Q2` of Example 3.1). Thus, SQuID considers any number of joins among αDB relations for the base query, but limits these to key-foreign-key joins. We discuss a simple method for deriving the base query in Section 3.5.2. SQuID's core challenge is to infer φ, which denotes a set of *semantic property filters* that are added as conjunctive selection predicates to Q^*. The base query and semantic property filters for `Q2` of Example 3.1 are:

Q^* =
```
SELECT name FROM academics, research
    WHERE research.aid = academics.id
```
φ = { `research.interest = 'data management'` }

3.2.1 Semantic Properties and Filters

Semantic properties encode characteristics of an entity. We distinguish semantic properties into two types. (1) A *basic semantic property* is affiliated with an entity directly. In the IMDb schema of Figure 3.2, "`gender = Male`" is a basic semantic property of a `person`. (2) A *derived semantic property* of an entity is an aggregate over a basic seman-

Sample database

id	name	gender	age
1	Tom Cruise	Male	50
2	Clint Eastwood	Male	90
3	Tom Hanks	Male	60
4	Julia Roberts	Female	50
5	Emma Stone	Female	29
6	Julianne Moore	Female	60

Example tuples

Column 1
Tom Cruise
Clint Eastwood

Figure 3.6: Sample database (left) with example tuples (right).

tic property of an associated entity. In Example 3.5, the number of movies of a particular genre that a person appeared in is a derived semantic property for `person`. We represent a semantic property p of an entity from a relation R as a triple $p = \langle A, V, \theta \rangle$. In this notation, V denotes a value or a value range for attribute A associated with entities in R.[2] The *association strength* parameter θ quantifies how strongly an entity is associated with the property. It corresponds to a threshold on derived semantic properties (e.g., the number of comedies an actor appeared in); it is not defined for basic properties ($\theta = \perp$).

A *semantic property filter* ϕ_p is a structured language representation of the semantic property p. In the data of Figure 3.6, the filters $\phi_{\langle \texttt{gender},\texttt{Male},\perp \rangle}$ and $\phi_{\langle \texttt{age},[\texttt{50},\texttt{90}],\perp \rangle}$ represent two basic semantic properties on `gender` and `age`, respectively. Expressed in relational algebra, filters on basic semantic properties map to standard selection predicates, e.g., $\sigma_{\texttt{gender}=\texttt{Male}}(\texttt{person})$ and $\sigma_{\texttt{50}\leq\texttt{age}\leq\texttt{90}}(\texttt{person})$. For derived properties, filters specify conditions on the association across different entities. In Example 3.5, for `person` entities, the filter $\phi_{\langle \texttt{genre},\texttt{Comedy},\texttt{30} \rangle}$ denotes the property of a person being associated with at least 30 movies with the basic property "`genre = Comedy`". In relational algebra, filters on derived properties map to selection predicates over derived relations in the αDB, e.g., $\sigma_{\texttt{genre}=\texttt{Comedy} \wedge \texttt{count} \geq \texttt{30}}(\texttt{persontogenre})$.

[2] SQuID can support disjunction for categorical attributes (e.g., "`gender = Male`" OR "`gender = Female`"), so V could be a set of values. However, for ease of exposition we keep our examples limited to properties without disjunction.

3.2.2 Filters and Example Tuples

To construct Q^{φ}, SQuID needs to infer the proper set of semantic property filters given a set of example tuples. Since all example tuples should be in the result of Q^{φ}, φ cannot contain filters that the example tuples do not satisfy. Thus, we only consider *valid* filters that map to selection predicates that *all* example tuples satisfy.

Definition 3.2 (Filter validity)**.** *Given a database $\mathcal{D}$, an example tuple set E, and a base query Q^*, a filter ϕ is valid if $Q^{\{\phi\}}(\mathcal{D}) \supseteq E$, where $Q^{\{\phi\}} = (Q^*, \{\phi\})$.*

Figure 3.6 shows a set of example tuples over the relation `person`. Given the base query Q^* = `SELECT name FROM person`, the filters $\phi_{\langle \texttt{gender},\texttt{Male},\perp \rangle}$ and $\phi_{\langle \texttt{age},[50,90],\perp \rangle}$ on relation `person` are valid, because all of the example entities of Figure 3.6 are `Male` and fall in the age range [50, 90].

Lemma 3.1. *(Validity of conjunctive filters). The conjunction $(\phi_1 \wedge \phi_2 \wedge \dots)$ of a set of filters $\Phi = \{\phi_1, \phi_2, \dots\}$ is valid, i.e., $Q^{\Phi}(\mathcal{D}) \supseteq E$, if and only if $\forall \phi_i \in \Phi$ ϕ_i is valid.*

Relaxing a filter (loosening its conditions) preserves validity. For example, if $\phi_{\langle \texttt{age},[50,90],\perp \rangle}$ is valid, then $\phi_{\langle \texttt{age},[40,120],\perp \rangle}$ is also valid. Among all valid filters, SQuID focuses on *minimal* valid filters, which have the tightest bounds. Bounds can be derived in other ways, e.g., informed by the result cardinality. However, we found the choice of the tightest bounds to work well in practice.

Definition 3.3 (Filter minimality)**.** *A basic semantic property filter $\phi_{\langle A,V,\perp \rangle}$ is* minimal *if it is valid, and $\forall V' \subset V$ $\phi_{\langle A,V',\perp \rangle}$ is invalid. A derived semantic property filter $\phi_{\langle A,V,\theta \rangle}$ is* minimal *if it is valid, and $\forall \epsilon > 0$ $\phi_{\langle A,V,\theta+\epsilon \rangle}$ is invalid.*

In the example of Figure 3.6, $\phi_{\langle \texttt{age},[50,90],\perp \rangle}$ is a minimal filter and $\phi_{\langle \texttt{age},[40,90],\perp \rangle}$ is not.

3.3 Probabilistic Abduction Model

We now revisit the problem of query intent discovery (Definition 3.1), and recast it based on our model of query intent (Section 3.2). Specifically, Definition 3.1 aims to dis-

cover an SPJ_{AI} query Q, which is then reduced to an equivalent SPJ query Q^{φ} on the αDB (as in Example 3.5). SQuID's task is to find the query Q^{φ} that maximizes the posterior probability $Pr(Q^{\varphi} \mid E)$, for a given set E of example tuples. In this section, we analyze the probabilistic model to compute this posterior, and break it down to three components.

3.3.1 Notations and Preliminaries

Semantic context $\boldsymbol{x}$. Observing a semantic property in a set of 10 examples is more significant than observing the same property in a set of 2 examples. We denote this distinction with the *semantic context* $x = (p, |E|)$, which encodes the size of the set $|E|$, where the semantic property p is observed. We denote with $\mathcal{X} = \{x_1, x_2, \dots\}$ the set of semantic contexts exhibited by the set of example tuples E.

Candidate SPJ query $\boldsymbol{Q^{\varphi}}$. Let $\Phi = \{\phi_1, \phi_2, \dots\}$ be the set of minimal valid filters, from hereon simply referred to as filters, where ϕ_i encodes the semantic context x_i. We omit $\langle A, V, \theta \rangle$ in the filter notation when the context in clear. Our goal is to identify the subset of filters in Φ that best captures the query intent. A set of filters $\varphi \subseteq \Phi$ defines a candidate query $Q^{\varphi} = (Q^*, \varphi)$, and $Q^{\varphi}(\mathcal{D}) \supseteq E$ (from Lemma 3.1).

Filter event $\boldsymbol{\widetilde{\phi}}$. A filter $\phi \in \Phi$ may or may not appear in a candidate query Q^{φ}. With slight abuse of notation, we denote the filter's presence ($\phi \in \varphi$) with ϕ and its absence ($\phi \notin \varphi$) with $\bar{\phi}$. We use $\widetilde{\phi}$ to represent the occurrence event of ϕ in Q^{φ}.

$$\text{Thus: } \widetilde{\phi} = \begin{cases} \phi & \text{if } \phi \in \varphi \\ \bar{\phi} & \text{if } \phi \notin \varphi \end{cases}$$

3.3.2 Modeling Query Posterior

We first analyze the probabilistic model for a *fixed base query* Q^* and then generalize the model. We use $Pr_*(a)$ as a shorthand for $Pr(a \mid Q^*)$. We model the query posterior $Pr_*(Q^{\varphi} \mid E)$, using Bayes' rule:

$$Pr_*(Q^\varphi \mid E) = \frac{Pr_*(E \mid Q^\varphi) \cdot Pr_*(Q^\varphi)}{Pr_*(E)}$$

By definition, $Pr_*(\mathcal{X} \mid E) = 1$; therefore:

$$Pr_*(Q^\varphi \mid E) = \frac{Pr_*(E, \mathcal{X} \mid Q^\varphi) \cdot Pr_*(Q^\varphi)}{Pr_*(E)} = \frac{Pr_*(E \mid \mathcal{X}, Q^\varphi) Pr_*(\mathcal{X} \mid Q^\varphi) \cdot Pr_*(Q^\varphi)}{Pr_*(E)}$$

Since $Pr_*(\mathcal{X} \mid E) = 1$ and applying Bayes' rule on the prior $Pr_*(E)$, we get:

$$Pr_*(Q^\varphi \mid E) = \frac{Pr_*(E \mid \mathcal{X}, Q^\varphi) Pr_*(\mathcal{X} \mid Q^\varphi) \cdot Pr_*(Q^\varphi)}{Pr_*(E \mid \mathcal{X}) \cdot Pr_*(\mathcal{X})}$$

Finally, E is conditionally independent of Q^φ given the semantic context $\mathcal{X}$, i.e., $Pr_*(E \mid \mathcal{X}, Q^\varphi) = Pr_*(E \mid \mathcal{X})$. Thus:

$$Pr_*(Q^\varphi \mid E) = \frac{Pr_*(\mathcal{X} \mid Q^\varphi) \cdot Pr_*(Q^\varphi)}{Pr_*(\mathcal{X})} \tag{3.1}$$

Equation 3.1 models the query posterior in terms of three components: (1) the semantic context prior $Pr_*(\mathcal{X})$, (2) the query prior $Pr_*(Q^\varphi)$, and (3) the semantic context posterior $Pr_*(\mathcal{X} \mid Q^\varphi)$. We proceed to analyze these components.

3.3.2.1 Semantic Context Prior

The semantic context prior $Pr_*(\mathcal{X})$ denotes the probability that any set of of example tuples of size $|E|$ exhibits the semantic contexts $\mathcal{X}$. This probability is not easy to compute analytically, as it involves computing a marginal over a potentially infinite set of candidate queries. In this work, we model the semantic context prior as proportional to the *selectivity* $\psi(\Phi)$ of $\Phi = \{\phi_1, \phi_2, \dots\}$, where $\phi_i \in \Phi$ is a filter that encodes context $x_i \in \mathcal{X}$:

$$Pr_*(\mathcal{X}) \propto \psi(\Phi) \tag{3.2}$$

Selectivity $\psi(\phi)$. Selectivity of filter ϕ denotes the portion of tuples from the result of the base query Q^* that satisfy ϕ, i.e., $\psi(\phi) = \frac{|Q^{\{\phi\}}(\mathcal{D})|}{|Q^*(\mathcal{D})|}$. Similarly, for a set of filters Φ,

$\psi(\Phi) = \frac{|Q^{\Phi}(\mathcal{D})|}{|Q^{*}(\mathcal{D})|}$. Intuitively, a selectivity value close to 1 means that the filter is not very selective and most tuples satisfy the filter; selectivity value close to 0 denotes that the filter is highly selective and rejects most of the tuples. For example, in Figure 3.6, $\phi_{\langle \texttt{gender},\texttt{Male},\perp \rangle}$ is more selective than $\phi_{\langle \texttt{age},[\texttt{50},\texttt{90}],\perp \rangle}$, with selectivities $\frac{1}{2}$ and $\frac{5}{6}$, respectively. Selectivity captures the rarity of a semantic context: uncommon contexts are present in fewer tuples, and, thus, appear in the output of fewer queries. Intuitively, a rare context has lower prior probability of being observed, which supports the assumption of Equation 3.2.

3.3.2.2 Query Prior

The query prior $Pr_{*}(Q^{\varphi})$ denotes the probability that Q^{φ} is the intended query, prior to observing the example tuples. We model the query prior as the joint probability of all filter events $\widetilde{\phi}$, where $\phi \in \Phi$. By further assuming filter independence,[3] we reduce the query prior to a product of probabilities of filter events:

$$Pr_{*}(Q^{\varphi}) = Pr_{*}(\textstyle\bigcap_{\phi \in \Phi} \widetilde{\phi}) = \prod_{\phi \in \Phi} Pr_{*}(\widetilde{\phi}) \tag{3.3}$$

The *filter event prior* $Pr_{*}(\widetilde{\phi})$ denotes the prior probability that filter ϕ is included in (if $\widetilde{\phi} = \phi$), or excluded from (if $\widetilde{\phi} = \bar{\phi}$), the intended query. We compute $Pr_{*}(\widetilde{\phi})$ for each filter as follows:

$$Pr_{*}(\phi) = \rho \cdot \delta(\phi) \cdot \alpha(\phi) \cdot \lambda(\phi) \;\text{ and }\; Pr_{*}(\bar{\phi}) = 1 - Pr_{*}(\phi)$$

Here, ρ is a base prior parameter, common across all filters, and represents the default value for the prior. The other factors (δ, α, and λ) reduce the prior, depending on characteristics of each filter. We describe these parameters next. (More discussion on the guidelines on how to choose the parameters are in Section 3.6.7 and Appendix A.4).

[3]Reasoning about database queries commonly assumes independence across selection predicates, which filters represent, although it may not hold in general.

Domain selectivity impact $\delta(\phi)$. Intuitively, a filter that covers a large range of values in an attribute's domain is unlikely to be part of the intended query. For example, if a user is interested in actors of a certain age group, that age group is more likely to be narrow ($\phi_{\langle \texttt{age},[41,45],\perp\rangle}$) than broad ($\phi_{\langle \texttt{age},[41,90],\perp\rangle}$). We penalize broad filters with the parameter $\delta \in (0, 1]$. The value $\delta(\phi)$ is 1 for filters that do not exceed a predefined ratio in the coverage of their domain, and decreases for filters that exceed this threshold. We use the notion of *domain coverage* of a filter $\phi_{\langle A,V,\theta\rangle}$ to denote the fraction of values of A's domain that V covers. For example, for attribute `age`, suppose that the domain consists of values in the range $[1, 100]$, then the filter $\phi_{\langle \texttt{age},[41,90],\perp\rangle}$ has 50% domain coverage and the filter $\phi_{\langle \texttt{age},[41,45],\perp\rangle}$ has 5% domain coverage. We use $\eta > 0$ to specify a threshold on domain coverage: any domain coverage lower than η does not reduce the domain selectivity impact δ. Once the coverage exceeds η, then δ decreases as domain coverage increases. We use another parameter $\gamma \geq 0$ that states how strongly we want to penalize a filter for having large domain coverage. The value of $\gamma = 0$ implies that we do not penalize at all, i.e., all filters will have $\delta(\phi) = 1$. As γ increases, we reduce δ more for larger domain coverage. Thus: $\delta(\phi_{\langle A,V,\theta\rangle}) = \frac{1}{\max(1, \frac{\texttt{domainCoverage}(V)}{\eta})^{\gamma}}$

Association strength impact $\alpha(\phi)$. Intuitively, a derived filter with low association strength is unlikely to appear in the intended query, as the filter denotes a weak association with the relevant entities. For example, $\phi_{\langle \texttt{genre},\texttt{Comedy},1\rangle}$ is less likely than $\phi_{\langle \texttt{genre},\texttt{Comedy},30\rangle}$ to represent a query intent. We use $\alpha(\phi) \in \{0, 1\}$ to characterize whether the filter ϕ should be considered as significant or not, based on the value of its association strength. For a derived filter $\phi_{\langle A,V,\theta\rangle}$ with θ as its association strength, if θ is lower than a threshold τ_α, we mark the filter ϕ as insignificant, and set $\alpha(\phi) = 0$. For all other filters, including basic filters, we set $\alpha(\phi) = 1$.

Outlier impact $\lambda(\phi)$. While $\alpha(\phi)$ characterizes the impact of association strength on a filter individually, $\lambda(\phi)$ characterizes its impact in consideration with other derived filters

Case A	
ϕ_1	$\phi_{\langle \texttt{genre}, \texttt{Comedy}, 30\rangle}$
ϕ_2	$\phi_{\langle \texttt{genre}, \texttt{SciFi}, 25\rangle}$
ϕ_3	$\phi_{\langle \texttt{genre}, \texttt{Drama}, 3\rangle}$
ϕ_4	$\phi_{\langle \texttt{genre}, \texttt{Action}, 2\rangle}$
ϕ_5	$\phi_{\langle \texttt{genre}, \texttt{Thriller}, 1\rangle}$

Case B	
ϕ_1	$\phi_{\langle \texttt{genre}, \texttt{Comedy}, 12\rangle}$
ϕ_2	$\phi_{\langle \texttt{genre}, \texttt{SciFi}, 10\rangle}$
ϕ_3	$\phi_{\langle \texttt{genre}, \texttt{Drama}, 10\rangle}$
ϕ_4	$\phi_{\langle \texttt{genre}, \texttt{Action}, 9\rangle}$
ϕ_5	$\phi_{\langle \texttt{genre}, \texttt{Thriller}, 9\rangle}$

Figure 3.7: Top two filters of Case A are interesting. No filter is interesting in Case B.

over the same attribute. Figure 3.7 demonstrates two cases of derived filters on the same attribute (`genre`), corresponding to two different sets of example tuples. In Case A, ϕ_1 and ϕ_2 have higher association strengths, and, thus are more significant than the other filters of the same family. Intuitively, this corresponds to the intent to retrieve actors who appeared in mostly `Comedy` and `SciFi` movies. In contrast, Case B does not have filters that stand out, as all have similar association strengths: The actors in this example set are not strongly associated with particular genres, and, thus, intuitively, this family of filters is not relevant to the query intent.

For a derived filter $\phi_{\langle A,V,\theta\rangle}$, we model its outlier impact $\lambda(\phi) \in \{0, 1\}$ using the *skewness* of the distribution of the association strengths (θ values) within the family of derived filters that share the same attribute A, but with different values for V and θ. Our assumption is that highly skewed, heavy-tailed distributions (Case A) are likely to contain the significant (intended) filters as *outliers* (ϕ_1 and ϕ_2). We set $\lambda(\phi) = 1$ for a derived filter ϕ whose association strength is an "outlier" in the association strength distribution of all filters of the same family. All other derived filters that have inlier θ values get $\lambda(\phi) = 0$. For example, the values 30 and 25 are outliers among the values $\{30, 25, 3, 2, 1\}$. Therefore, in Case A, we consider the two filters ϕ_1 and ϕ_2 as significant (i.e., $\lambda(\phi_1) = \lambda(\phi_2) = 1$) and the rest ($\phi_3$, ϕ_4, and ϕ_5) as insignificant (i.e., $\lambda(\phi_3) = \lambda(\phi_4) = \lambda(\phi_5) = 0$). For basic filters, we set $\lambda(\phi) = 1$, as no two basic filters share the same attribute. To compute the outlier impact of a filter $\phi_{\langle A,V,\theta\rangle}$, we use the skewness of the association strength distribution Θ_A that consists

of the θ values for all derived filters involving attribute A. For a skewed distribution, we consider a value to be an outlier if is more than k standard deviation larger than the mean.

3.3.2.3 Semantic Context Posterior

The semantic context posterior $Pr_*(\mathcal{X} \mid Q^\varphi)$ is the probability that a set of example tuples of size $|E|$, sampled from the output of a particular query Q^φ, exhibits the set of semantic contexts $\mathcal{X}$:

$$Pr_*(\mathcal{X} \mid Q^\varphi) = Pr_*(x_1, x_2, ..., x_n \mid Q^\varphi)$$

Two semantic contexts $x_i, x_j \in \mathcal{X}$ are conditionally independent, given Q^φ. Therefore:

$$Pr_*(\mathcal{X} \mid Q^\varphi) = \prod_{i=1}^{n} Pr_*(x_i \mid Q^\varphi) = \prod_{i=1}^{n} Pr_*(x_i \mid \widetilde{\phi}_1, \widetilde{\phi}_2, \dots)$$

Recall that ϕ_i encodes the semantic context x_i (Section 3.3.1). We assume that x_i is conditionally independent of any $\widetilde{\phi}_j, i \neq j$, given $\widetilde{\phi}_i$ (this always holds for $\widetilde{\phi}_i = \phi_i$):

$$Pr_*(\mathcal{X} \mid Q^\varphi) = \prod_{i=1}^{n} Pr_*(x_i \mid \widetilde{\phi}_i) \tag{3.4}$$

For each x_i, we compute $Pr_*(x_i \mid \widetilde{\phi}_i)$ based on the state of the filter event ($\widetilde{\phi}_i = \phi_i$ or $\widetilde{\phi}_i = \bar{\phi}_i$):

$\boldsymbol{Pr_*(x_i \mid \phi_i)}$: By definition, all tuples in $Q^{\{\phi_i\}}(\mathcal{D})$ exhibit the property of x_i. Hence, $Pr_*(x_i \mid \phi_i) = 1$.

$\boldsymbol{Pr_*(x_i \mid \bar{\phi}_i)}$: This is the probability that a set of $|E|$ tuples drawn uniformly at random from $Q^*(\mathcal{D})$ (ϕ_i is not applied to the base query) exhibits the context x_i. The portion of tuples in $Q^*(\mathcal{D})$ that exhibit the property of x_i is the selectivity $\psi(\phi_i)$. Therefore, $Pr_*(x_i \mid \bar{\phi}_i) \approx \psi(\phi_i)^{|E|}$.

Using Equations (3.1)–(3.4), we derive the final form of the query posterior (where K is a normalization constant):

$$\begin{aligned} Pr_*(Q^\varphi \mid E) &= \frac{K}{\psi(\Phi)} \prod_{\phi_i \in \Phi} \Big(Pr_*(\widetilde{\phi}_i) \cdot Pr_*(x_i \mid \widetilde{\phi}_i) \Big) \\ &= \frac{K}{\psi(\Phi)} \prod_{\phi_i \in \varphi} \Big(Pr_*(\phi_i) \cdot Pr_*(x_i \mid \phi_i) \Big) \prod_{\phi_i \notin \varphi} \Big(Pr_*(\bar{\phi}_i) \cdot Pr_*(x_i \mid \bar{\phi}_i) \Big) \end{aligned} \quad (3.5)$$

Generalization. So far, our analysis focused on a fixed base query. Given an SPJ query Q^φ, the underlying base query Q^* is deterministic, i.e., $Pr(Q^* \mid Q^\varphi) = 1$. Hence:

$$\begin{aligned} Pr(Q^\varphi \mid E) &= Pr(Q^\varphi, Q^* \mid E) \\ &= Pr(Q^\varphi \mid Q^*, E) \cdot Pr(Q^* \mid E) \\ &= Pr_*(Q^\varphi \mid E) \cdot Pr(Q^* \mid E) \end{aligned}$$

We assume $Pr(Q^* \mid E)$ to be equal for all valid base queries, where $Q^*(\mathcal{D}) \supseteq E$. Then we use $Pr_*(Q^\varphi \mid E)$ to find the query Q that maximizes the query posterior $Pr(Q \mid E)$.

3.4 Offline Abduction Preparation

In this section, we discuss system considerations to perform query intent discovery *efficiently*. SQUID employs an offline module that performs several precomputation steps to make the database *abduction-ready*. The abduction-ready database (αDB) augments the original database with derived relations that store associations across entities and precomputes semantic property statistics. Deriving this information is relatively straightforward; the contributions of this section lie in the design of the αDB, the information it maintains, and its role in supporting efficient query intent discovery. We describe the three major functions of the αDB.

3.4.1 Entity Lookup

SQUID's goal is to discover query intent based on the user-provided examples. To do that, it first needs to determine which entities in the database correspond to the examples. SQUID uses a *global inverted column index* [294], built over all text attributes, to perform fast lookups, matching the provided examples to database entities.

3.4.2 Semantic Property Discovery

To reason about intent, SQUID first needs to determine what makes the examples similar. SQUID looks for semantic properties within entity relations (e.g., `gender` appears in table `person`), other relations (e.g., `genre` appears in a separate table joining with `movie` through a key-foreign-key constraint), and other entities, (e.g., the number of movies of a particular `genre` that a `person` has appeared in). The αDB precomputes and stores such *derived relations* (e.g., `persontogenre`), as these frequently involve several joins and aggregations and performing them at runtime is prohibitive. E.g., SQUID computes the `persontogenre` relation (Figure 3.4), and stores it in the αDB, using `Q6`:

```
Q6: CREATE TABLE persontogenre as
    (SELECT person_id, genre_id, count(*) AS count
     FROM castinfo, movietogenre
     WHERE castinfo.movie_id = movietogenre.movie_id
     GROUP BY person_id, genre_id)
```

For the αDB construction, SQUID only relies on very basic information to understand the data organization: (1) the database schema, including the specification of primary- and foreign-key constraints, and (2) additional meta-data, which can be provided once by a database administrator, that specify which tables describe entities (e.g., `person`, `movie`), and which tables and attributes describe direct properties of entities (e.g., `genre`, `age`). SQUID then automatically discovers fact tables, which associate entities and properties, by exploiting the key-foreign-key relationships. SQUID also automatically discovers de-

rived properties up to a certain predefined depth, using paths in the schema graph that connect entities to properties. Since the number of possible values for semantic properties is typically very small and remains constant as entities grow, the αDB grows linearly with the data size. In our implementation, we restrict the derived property discovery to the depth of two fact tables (e.g., SQuID derives `persontogenre` through `castinfo` and `movietogenre`). SQuID can support deeper associations, but those are rare in practice. SQuID assumes that different entity types appear in different relations, which is the case in many commonly used schema types, such as star, galaxy, and fact-constellation schemas. SQuID can perform inference in a denormalized setting, but would not be able to produce and reason about derived properties in those cases.

3.4.3 Smart Selectivity Computation

For basic filters involving categorical values, SQuID stores the selectivity for each value. However, for numeric ranges, the number of possible filters can grow quadratically with the number of possible values. SQuID avoids wasted computation and space by only precomputing selectivities $\psi(\phi_{\langle A,[min_{V_A},v],\perp\rangle})$ for all $v \in V_A$, where V_A is the set of values of attribute A in the corresponding relation, and min_{V_A} is the minimum value in V_A. The αDB can derive the selectivity of a filter with any value range as:

$$\psi(\phi_{\langle A,(l,h],\perp\rangle}) = \psi(\phi_{\langle A,[min_{V_A},h],\perp\rangle}) - \psi(\phi_{\langle A,[min_{V_A},l],\perp\rangle})$$

In case of derived semantic properties, SQuID precomputes selectivities $\psi(\phi_{\langle A,v,\theta\rangle})$ for all $v \in V_A, \theta \in \Theta_{A,v}$, where $\Theta_{A,v}$ is the set of values of association strength for the property "$A = v$".

3.5 Query Intent Discovery

During normal operation, SQuID receives example tuples from a user, consults the αDB, and infers the most likely query intent (Definition 3.1). In this section, we describe

how SQuID resolves ambiguity in the provided examples, how it derives their semantic context, and how it finally abduces the intended query.

3.5.1 Entity and Context Discovery

SQuID's probabilistic abduction model (Section 3.3) relies on the set of semantic contexts $\mathcal{X}$ and determines which of these contexts are intended vs coincidental, by the inclusion or exclusion of the corresponding filters in the inferred query. To derive the set of semantic contexts from the examples, SQuID first needs to identify the entities in the αDB that correspond to the provided examples.

3.5.1.1 Entity Disambiguation

User-provided examples are not complete tuples, but often single-column values that correspond to an entity. As a result, there may be ambiguity that SQuID needs to resolve. For example, suppose the user provides the examples: `{Titanic, Pulp Fiction, The Matrix}`. SQuID consults the precomputed inverted column index to identify the attributes (`movie.title`) that contain all the example values, and classifies the corresponding entity (`movie`) as a potential match. However, while the dataset contains unique entries for `Pulp Fiction (1994)` and `The Matrix (1999)`, there are 4 possible mappings for `Titanic`: (1) a 1915 Italian film, (2) a 1943 German film, (3) a 1953 film by Jean Negulesco, and (4) the 1997 blockbuster film by James Cameron.

The key insight for resolving such ambiguities is that the provided examples are more likely to be alike. SQuID selects the entity mappings that maximize the semantic similarities across the examples. Therefore, based on the year and country information, it determines that `Titanic` corresponds to the 1997 film, as it is most similar to the other two (unambiguous) entities. In case of derived properties, e.g., nationality of actors appearing in a film, SQuID aims to increase the association strength (e.g., the number of such actors). Since the number of examples are typically small, SQuID can determine the right mappings by considering all combinations.

3.5.1.2 Semantic Context Discovery

Once SQuID identifies the right entities, it then explores all the semantic properties stored in the αDB that match these entities (e.g., `year`, `genre`, etc.). Since the αDB precomputes and stores the derived properties, SQuID can produce all the relevant properties using queries with at most one join. For each property, SQuID produces semantic contexts as follows:

Basic property on categorical attribute. If all examples in E contain value v for the property of attribute A, SQuID produces the semantic context $(\langle A, v, \perp\rangle, |E|)$. For example, a user provides three movies: `Dunkirk`, `Logan`, and `Taken`. The attribute `genre` corresponds to a basic property for movies, and all these movies share the values, `Action` and `Thriller`, for this property. SQuID generates two semantic contexts: $(\langle \texttt{genre}, \texttt{Action}, \perp\rangle, 3)$ and $(\langle \texttt{genre}, \texttt{Thriller}, \perp\rangle, 3)$.

Basic property on numerical attribute. If v_{min} and v_{max} are the minimum and maximum values, respectively, that the examples in E demonstrate for the property of attribute A, SQuID creates a semantic context on the range $[v_{min}, v_{max}]$: $(\langle A, [v_{min}, v_{max}], \perp\rangle, |E|)$. For example, if E contains three persons with ages 45, 50, and 52, SQuID will produce the context $(\langle \texttt{age}, [45, 52], \perp\rangle, 3)$.

Derived property. If all examples in E contain value v for the derived property of attribute A, SQuID produces the context $(\langle A, v, \theta_{min}\rangle, |E|)$, where θ_{min} is the minimum association strength for the value v among all examples. For example, if E contains two persons who appeared in 3 and 5 `Comedy` movies, SQuID will produce $(\langle \texttt{genre}, \texttt{Comedy}, 3\rangle, 2)$.

3.5.2 Query Abduction

SQuID starts abduction by constructing a base query that captures the structure of the example tuples. Once it identifies the entity and attribute that matches the examples (e.g., `person.name`), it forms the minimal PJ query (e.g., `SELECT name FROM person`).

Algorithm 1: QueryAbduction (E, Q^*, Φ)

Input: set of entities E, base query Q^*, set of minimal valid filters Φ
Output: Q^φ such that $Pr_*(Q^\varphi \mid E)$ is maximized

1 $\mathcal{X} = \{x_1, x_2, ...\}$ // semantic contexts in E
2 $\varphi = \emptyset$
3 **foreach** $\phi_i \in \Phi$ **do**
4 $\quad \texttt{include}_{\phi_i} = Pr_*(\phi_i) Pr_*(x_i \mid \phi_i)$ // from Equation (3.5)
5 $\quad \texttt{exclude}_{\phi_i} = Pr_*(\bar{\phi}_i) Pr_*(x_i \mid \bar{\phi}_i)$ // from Equation (3.5)
6 $\quad$ **if** $\texttt{include}_{\phi_i} > \texttt{exclude}_{\phi_i}$ **then**
7 $\quad\quad \varphi = \varphi \cup \{\phi_i\}$
8 **return** Q^φ

It then iterates through the discovered semantic contexts and appends the corresponding relations to the FROM clause and the appropriate key-foreign-key join conditions in the WHERE clause. Since the αDB precomputes and stores the derived relations, each semantic context will add at most one relation to the query.

The number of candidate base queries is typically very small. For each base query Q^*, SQuID abduces the best set of filters $\varphi \subseteq \Phi$ to construct SPJ query Q^φ, by augmenting the WHERE clause of Q^* with the corresponding selection predicates. SQuID also removes from Q^φ any joins that are not relevant to the selected filters φ. While the number of candidate SPJ queries grows exponentially in the number of minimum valid filters ($2^{|\Phi|}$), we prove that we can make decisions on including or excluding each filter independently. Algorithm 1 iterates over the set of minimal valid filters Φ and decides to include a filter only if its addition to the query increases the query posterior (lines 6-7). Our abduction algorithm has $O(|\Phi|)$ time complexity and is guaranteed to produce the query Q^φ that maximizes the query posterior.

Theorem 3.1. *Given a base query Q^*, a set of examples E, and a set of minimal valid filters Φ, Algorithm 1 returns the query Q^φ, where $\varphi \subseteq \Phi$, such that $Pr_*(Q^\varphi \mid E)$ is maximized. (Proof is in Appendix A.2)*

3.6 Experiments

We now present an extensive experimental evaluation of SQUID over three real-world datasets, with a total of 41 benchmark queries of varying complexities. Our evaluation shows that SQUID is scalable and effective, even with a small number of example tuples. Our evaluation extends to qualitative case studies over real-world, user-generated examples, which demonstrate that SQUID succeeds in inferring the query intent of real-world users. We further demonstrate that when used as a query reverse engineering system in a closed-world setting, SQUID outperforms the state of the art. Finally, we show that SQUID is superior to semi-supervised PU-learning in terms of both efficiency and effectiveness.

3.6.1 Experimental Setup

We implemented SQUID in Java and all experiments were run on a 12x2.66 GHz machine with 16GB RAM running CentOS 6.9 with PostgreSQL 9.6.6.

3.6.1.1 Datasets and Benchmark Queries

Our evaluation includes three real-world datasets and a total of 41 benchmark queries, designed to cover a broad range of intents and query structures. selection predicates (S)—and the result set cardinality. We summarize the datasets and benchmark queries below.

IMDb (633 MB): The IMDb dataset [156] is a well-known source of movie and entertainment facts and contains information regarding over 10 million personalities, along with their demographic information; and about 6 million movies and TV series, along with their genre, language, country, certificate, production company, cast and crew, etc. We stored the IMDb dataset using a schema that contains 15 relations. We designed a set of 16 benchmark queries (Figure A.2) ranging the number of joins (1 to 8 relations), the number of selection predicates (0 to 7), and the result cardinality (12 to 2512 tuples).

DBLP (22 MB): We used a subset of the DBLP data [6], with 14 relations, and 16 years (2000–2015) of top 81 conference publications. We designed 5 queries (Figure A.3) rang-

ing the number of joins (3 to 8 relations), the number of selection predicates (2 to 4), and the result cardinality (15 to 468 tuples).

Adult (4 MB): This is a single-relation dataset containing census data of people and their income brackets. We generated 20 queries (Figure A.4 and A.5), randomizing the attributes and predicate values, ranging the number of selection predicates (2 to 7) and the result cardinality (8 to 1404 tuples).

We collected the datasets from various sources and we provide them in Figure A.6. We provide the detailed description of the datasets in Figures A.7 and A.8. We mention the cardinalities of the large relations for providing a sense of the data size and associations among relations. For the scalability experiment, we generated 3 versions of the IMDb database (details are in Appendix A.3).

3.6.1.2 Case Study Data

We retrieved several public lists (sources are listed in Figure A.6) with human-generated examples, and identified the corresponding intent. For example, a user-created list of "115 funniest actors" reveals a query intent (funny actors), and provides us with real user examples (the names in the list). We used this method to design 3 case studies: funny actors (IMDb), 2000s Sci-Fi movies (IMDb), and prolific database researchers (DBLP).

3.6.1.3 Metrics

We report query discovery time as a metric of efficiency. We measure effectiveness using precision, recall, and f-score. If Q is the intended query, and Q' is the query inferred by SQuID, precision is computed as $\frac{Q'(\mathcal{D}) \cap Q(\mathcal{D})}{Q'(\mathcal{D})}$ and recall as $\frac{Q'(\mathcal{D}) \cap Q(\mathcal{D})}{Q(\mathcal{D})}$; f-score is their harmonic mean. We also report the total number of predicates in the produced queries and compare them with the actual intended queries.

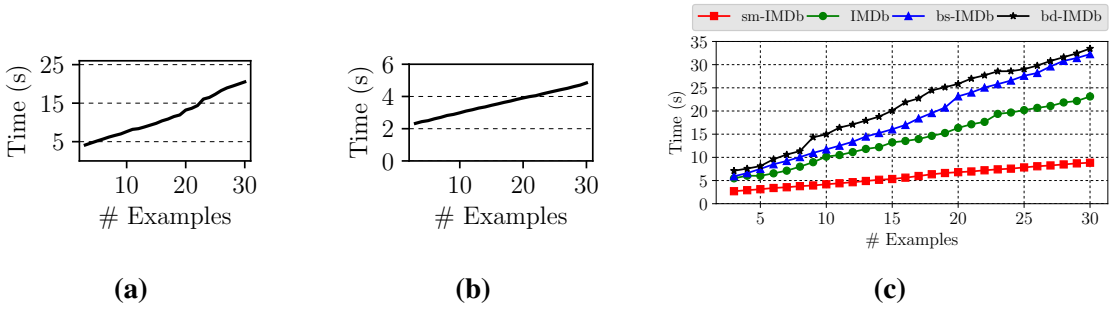


Figure 3.8: Average abduction time over the benchmark queries in (a) IMDb (b) DBLP and (c) 4 versions of the IMDb dataset.

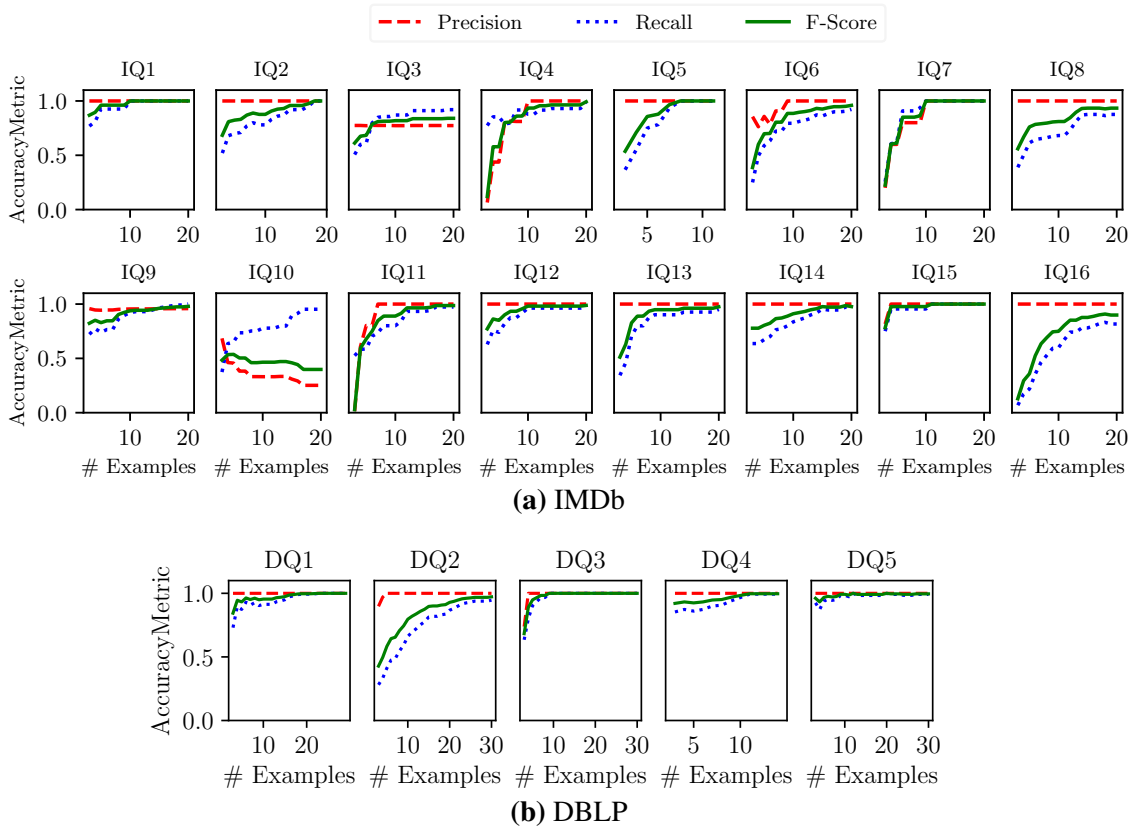
3.6.1.4 Comparisons

To the best of our knowledge, existing QBE techniques do not produce SPJ queries without (1) a large number of examples, or (2) additional information, such as provenance. For this reason, we cannot meaningfully compare SQuID with those approaches. Removing the open-world requirement, SQuID is most similar to the QRE system TALOS [312] with respect to expressiveness and capabilities (Figure A.1). We compare the two systems for query reverse engineering tasks in Section 3.6.5. We also compare SQuID against PU-learning methods [84] in Section 3.6.6.

3.6.2 Scalability

In our first set of experiments, we examine the scalability of SQuID against increasing number of examples and varied dataset sizes. Figures 3.8(a) and 3.8(b) display the abduction time for the IMDb and DBLP datasets, respectively, as the number of provided examples increases, averaged over all benchmark queries in each dataset. Since SQuID retrieves semantic properties and computes context for each example, the runtime increases linearly with the number of examples, which is what we observe.

Figure 3.8(c) extends this experiment to datasets of varied sizes. We generate three alternative versions of the IMDb dataset: (1) sm-IMDb (75 MB), a downsized version that keeps 10% of the original data; (2) bs-IMDb (1330 MB), doubles the entities of the origi-



(a) IMDb

(b) DBLP

Figure 3.9: SQuID achieves high accuracy with few examples (typically $\sim$ 5) in most benchmark queries.

nal dataset and creates associations among the duplicate entities (`person` and `movie`) by replicating their original associations; (3) bd-IMDb (1926 MB), is the same as bs-IMDb but also introduces associations between the original entities and the duplicates, creating denser connections. SQuID's runtime increases for all datasets with the number of examples, and, predictably, larger datasets face longer abduction times. Query abduction involves point queries to retrieve semantic properties of the entities, using B-tree indexes. As the data size increases, the runtime of these queries grows logarithmically. SQuID is slower on bd-IMDb than on bs-IMDb: both datasets include the same entities, but bd-IMDb has denser associations, which results in additional derived semantic properties.

3.6.3 Abduction Accuracy

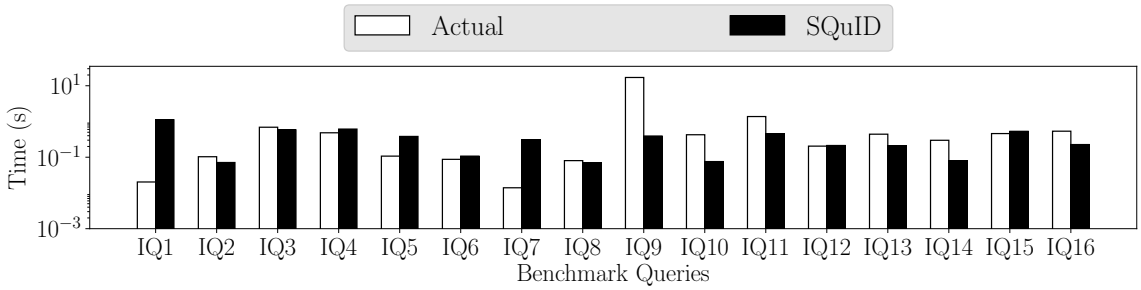
Intuitively, with a larger number of examples, abduction accuracy should increase: SQuID has access to more samples of the query output, and can more easily distinguish coincidental from intended similarities. Figure 3.9 confirms this intuition, and precision, recall, and f-score increase, often very quickly, with the number of examples for most of our benchmark queries. We discuss here a few particular queries.

`IQ4` & `IQ11`: These queries include a statistically common property (USA movies), and SQuID needs more examples to confirm that the property is indeed intended, not coincidental; hence, the precision converges more slowly.

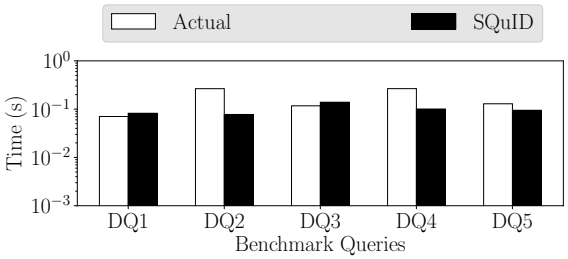
`IQ6`: In many movies where Clint Eastwood was a director, he was also an actor. SQuID needs to observe sufficient examples to discover that the property `role:Actor` is not intended, so recall converges more slowly.

`IQ10`: SQuID performs poorly for this query. The query looks for actors appearing in more than 10 *Russian* movies that were released *after 2010*. While SQuID discovers the derived properties "more than 10 Russian movies" and "more than 10 movies released after 2010", it cannot compound the two into "more than 10 Russian movies released after 2010". This query is simply outside of SQuID's search space, and SQuID produces a query with more general predicates than intended, which is why precision drops.

`IQ3`: The query is looking for actresses who are Canadian and were born after 1970. SQuID successfully discovers the properties `gender:Female`, `country:Canada`, and `birth year` $\geq$ `1970`; however, it fails to capture the property of "being an actress", corresponding to having appeared in at least 1 film. The reason is that SQuID is programmed to ignore weak associations (a person associated with only 1 movie). This behavior can be fixed by adjusting the association strength parameter to allow for weaker associations.



(a) IMDb



(b) DBLP

Figure 3.10: SQUID rarely produces queries that are slower than the original with respect to query runtime.

3.6.3.1 Execution Time

While the accuracy results demonstrate that the abduced queries are semantically close to the intended queries, SQUID could be deriving a query that is semantically close, but more complex and costly to compute. In Figures 3.10(a) and 3.10(b) we graph the average runtime of the abduced queries and the actual benchmark queries. We observe that in most cases the abduced queries and the corresponding benchmarks are similar in execution time. Frequently, the abduced queries are faster because they take advantage of the precomputed relations in the αDB. In few cases (`IQ1`, `IQ5`, and `IQ7`) SQUID discovered additional properties that, while not specified by the original query, are inherent in all intended entities. For example, in `IQ5`, all movies with `Tom Cruise` and `Nicole Kidman` are also English language movies and released between 1990 and 2014.

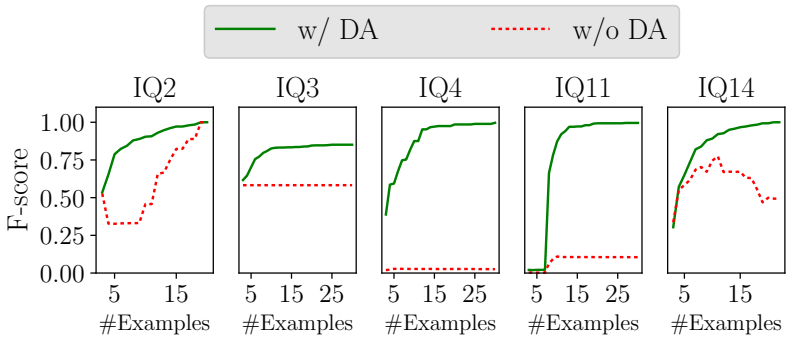


Figure 3.11: Effect of disambiguation on IMDb.

3.6.3.2 Effect of Entity Disambiguation

Finally, we found that entity disambiguation never hurts abduction accuracy, and may significantly improve it. Figure 3.11 displays the impact of disambiguation for five IMDb benchmark queries, where disambiguation significantly improves the f-score.

3.6.4 Qualitative Case Studies

We now present qualitative results on the performance of SQUID, through a simulated user study. We designed 3 case studies, by constructing queries and examples from human-generated, publicly available lists.

Funny actors (IMDb). We created a list of names of 211 "funny actors", collected from human-created public lists and Google Knowledge Graph (sources are in Figure A.6), and used these names as examples of the query intent "funny actors." Figure 3.12(a) demonstrates the accuracy of the abduced query over a varying number of examples. Each data point is an average across 10 different random samples of example sets of the corresponding size. For this experiment, we tuned SQUID to normalize the association strength, which means that the relevant predicate would consider the fraction of movies in an actor's portfolio classified as comedies, rather than the absolute number.

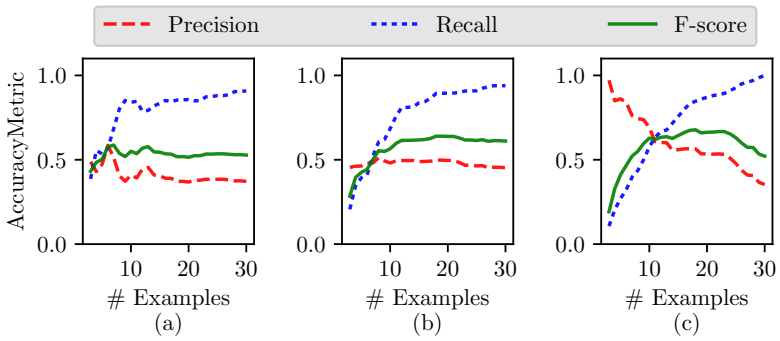


Figure 3.12: Precision, recall, and f-score for (a) Funny actors (b) 2000s Sci-Fi movies (c) Prolific DB researchers.

2000s Sci-Fi movies (IMDb). We used a user-created list of 165 Sci-Fi movies released in 2000s as examples of the query intent "2000s Sci-Fi movies". Figure 3.12(b) displays the accuracy of the abduced query, averaged across 10 runs for each example set size.

Prolific database researchers (DBLP). We collected a list of database researchers who served as chairs, group leaders, or program committee members in SIGMOD 2011–2015 and selected the top 30 most prolific. Figure 3.12(c) displays the accuracy of the abduced query averaged, across 10 runs for each example set size.

Analysis In our case studies there is no (reasonable) SQL query that models the intent well and produces an output that exactly matches our lists. Public lists have biases, such as not including less well-known entities even if these match the intent. To counter this bias, we use popularity masks (derived from public lists) to filter the examples and the abduced query outputs. In our prolific researchers use case, some well-known and prolific researchers may happen to not serve in service roles frequently, or their commitments may be in venues we did not sample. Therefore, it is not possible to achieve high precision, as the data is bound to contain and retrieve entities that do not appear on the lists, even if the query is a good match for the intent. For this reason, our precision numbers in the case studies are low. However our recall rises quickly with enough examples, which indicates that the abduced queries converge to the correct intent.

3.6.5 Query Reverse Engineering

We present an experimental comparison of SQUID with TALOS [312], a state-of-the-art query reverse engineering (QRE) system. We picked TALOS because other related methods either focus on more restricted query classes [175, 345] or do not scale to data sizes large enough for this evaluation [322, 346] (Figure A.1). Unlike SQUID, QRE systems operate in a closed-world setting, assuming that the provided examples comprise the entire query output. In the closed-world setting, SQUID is handicapped against a dedicated QRE system, as it does not take advantage of the closed-world constraint.

For this evaluation under the QRE setting, we use the IMDb and DBLP datasets, as well as the Adult dataset, on which TALOS was shown to perform well [312]. For each dataset, we provided the entire output of the benchmark queries as input to SQUID and TALOS. Since there is no need to drop coincidental filters for query reverse engineering, we set the parameters so that SQUID behaves optimistically (e.g., high filter prior, low association strength threshold, etc.). We adopt the notion of *instance equivalent query* (IEQ) from the QRE literature [312] to express that two queries produce the same set of results on a particular database instance. A QRE task is successful if the system discovers an IEQ of the original query (f-score=1). For the IMDb dataset, SQUID was able to successfully reverse engineer 11 out of 16 benchmark queries. Additionally, in 4 cases where exact IEQs were not abduced, SQUID queries generated output with ≥ 0.98 f-score. SQUID failed only for `IQ10`, which is a query that falls outside the supported query family, as discussed in Section 3.6.3. For the DBLP and Adult datasets, SQUID successfully reverse-engineered all benchmark queries. We compare SQUID to TALOS on three metrics: number of predicates (including join and selection predicates), query discovery time, and f-score.

Adult. Both SQUID and TALOS achieved perfect f-score on the 20 benchmark queries. Figure 3.13 compares the systems in terms of the number of predicates in the queries they produce (top) and query discovery time (bottom). SQUID almost always produces simpler queries, close in the number of predicates to the original query, while TALOS queries

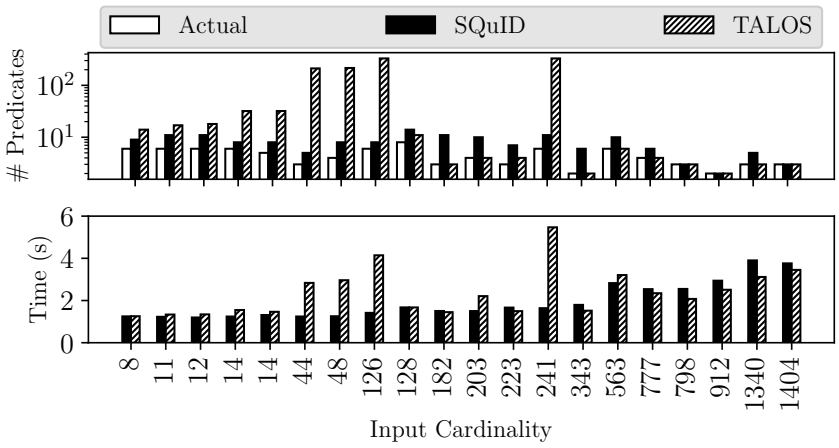
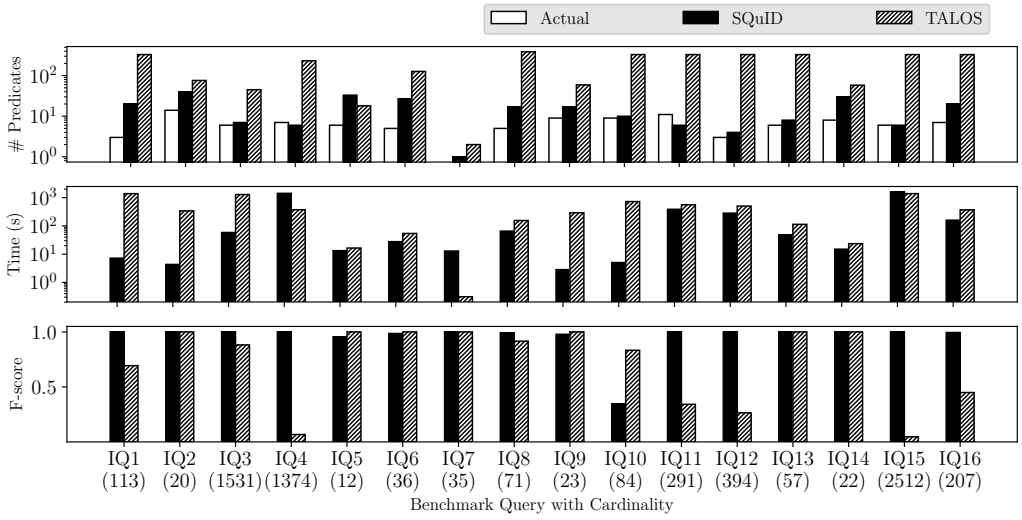


Figure 3.13: Both systems achieve perfect f-score on the Adult dataset (not shown). SQUID produces significantly smaller queries, often by orders of magnitude, and is often much faster.

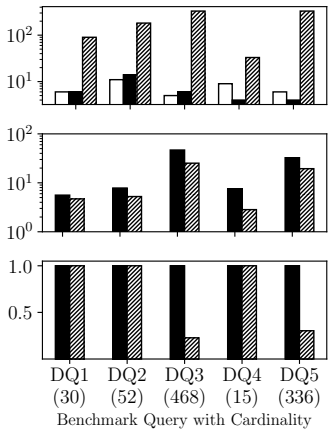
contain more than 100 predicates in 20% of the cases. SQUID is faster than TALOS when the input cardinality is low ($\sim$100 tuples), and becomes slower for the largest input sizes ($>$ 700 tuples). SQUID was not designed as a QRE system, and in practice, users rarely provide large example sets. SQUID's focus is on inferring simple queries that model the intent, rather than cover all examples with potentially complex and lengthy queries.

IMDb. Figure 3.14(a) compares the two systems on the 16 IMDb queries. SQUID generally produced better queries: in all cases, our abduced queries where significantly smaller, and our f-score is higher for most queries. SQUID was also faster than TALOS for most of the benchmark queries. We now delve deeper into some particular cases.

For `IQ1` (cast of `Pulp Fiction`), TALOS produces a query with f-score = 0.7. We attempted to provide guidance to TALOS through a system parameter that specifies which attributes to include in the selection predicates (which would give it an unfair advantage). TALOS first performs a full join among the participating relations (`person` and `castinfo`) and then performs classification on the denormalized table (with attributes person, movie, role). TALOS gives all rows referring to a cast member of `Pulp Fiction` a positive label (based on the examples), regardless of the movie that row refers to, and then



(a) IMDb



(b) DBLP

Figure 3.14: SQuID produces queries with significantly fewer predicates than TALOS and is more accurate on both IMDb and DBLP. SQuID is almost always faster on IMDb, but TALOS is faster on DBLP.

builds a decision tree based on these incorrect labels. This is a limitation of TALOS, which SQuID overcomes by looking at the semantic similarities of the examples, rather than treating them simply as labels.

SQuID took more time than TALOS in `IQ4`, `IQ7`, and `IQ15`. The result sets of `IQ4` and `IQ15` are large (> 1000), so this is expected. `IQ7` retrieves all movie genres, and,

thus, does not require any selection predicate. As a decision tree approach, TALOS has the advantage here, as it stops at the root and does not need to traverse the tree. In contrast, SQUID retrieves all semantic properties of the example tuples only to discover that either there is nothing common among them, or the property is not significant. While SQUID takes longer, it still abduces the correct query. These are not representative of QBE scenarios, as users are unlikely to provide large number of example tuples or have very general intents (PJ queries without selection).

DBLP. Figure 3.14(b) compares the two systems on the DBLP dataset. Here, SQUID successfully reverse engineered all five benchmark queries, but TALOS failed to reverse engineer two of them. TALOS also produced very complex queries, with 100 or more predicates for four of the cases. In contrast, SQUID's abductions were orders of magnitude smaller, on par with the original query. On this dataset, SQUID was slower than TALOS, but not by a lot.

3.6.6 Comparison with Learning Methods

Query intent discovery can be seen as a one-class classification problem, where the task is to identify the tuples that satisfy the desired intent. Positive and Unlabeled (PU) learning addresses this problem by learning a classifier from positive examples and unlabeled data in a semi-supervised setting. We compare SQUID against an established PU-learning method [84] on 20 benchmark queries over Adult. The setting of this experiment conforms with the technique's requirements [84]: the dataset comprises of a single relation and the examples are chosen uniformly at random from the positive data.

Figure 3.15 (a) compares the accuracy of SQUID and PU-learning using two different estimators, decision tree (DT) and random forest (RF). We observe that PU-learning needs a large fraction ($> 70\%$) of the query result to achieve f-score comparable to SQUID. PU-learning favors precision over recall, and the latter drops significantly when the number of examples is low. In contrast, SQUID achieves robust performance, even with few

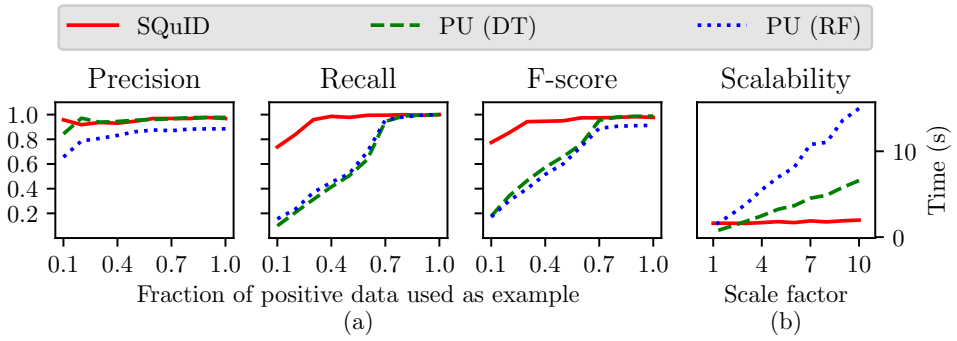


Figure 3.15: (a) PU-learning needs a large fraction ($> 70\%$) of the query results (positive data) as examples to achieve accuracy comparable to SQuID. (b) The total required time for training and prediction in PU-learning increases linearly with the data size. In contrast, abduction time for SQuID increases logarithmically.

examples, because it can encode problem-specific assumptions (e.g., that there exists an underlying SQL query that models the intent, that some filters are more likely than other filters, etc.); this cannot be done in straightforward ways for machine learning methods.

To evaluate scalability, we replicated the Adult dataset, with a scale factor up to 10x. Figure 3.15 (b) shows that PU-learning becomes significantly slower than SQuID as the data size increases, whereas SQuID's runtime performance remains largely unchanged. This is due to the fact that, SQuID does not directly operate on the data outside of the examples (unlabeled data); rather, it relies on the αDB, which contains a highly compressed summary of the semantic property statistics (e.g., filter selectivities) of the data. In contrast, PU-learning builds a new classifier over all of the data for each query intent discovery task.

3.6.7 SQuID Parameters

We further empirically evaluate how the parameters of SQuID contribute to its performance and found that SQuID does not depend heavily on the parameter values and its dependency on the parameters diminishes as the number of examples increases (details are in Appendix A.4).

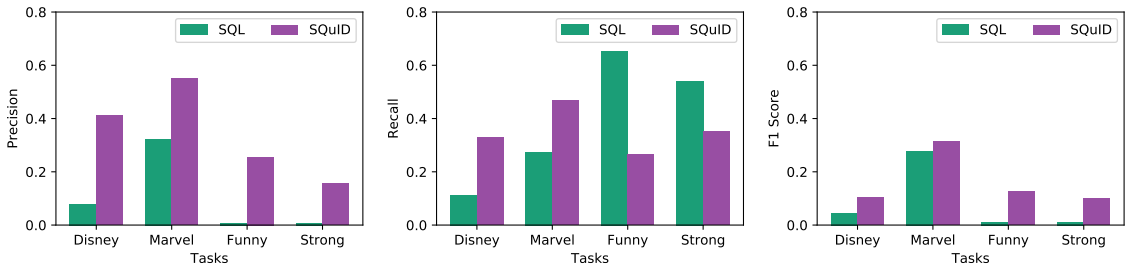
3.7 Comparative User Studies

In this section, we present findings from our comparative user studies over SQUID and the traditional SQL-based mechanism. We conducted two comparative user studies: (1) a controlled experiment study involving 35 participants, and (2) an interview study involving 7 interviewees to gain a richer understanding of users' issues and preferences. All participants and interviewees had varying levels of SQL expertise and experience, but were required to have at least basic SQL skills. Our studies focused on the task of data exploration over the IMDb dataset, and explored how SQUID compares against the traditional SQL querying mechanism, over four objective and subjective data exploration tasks. Specifically, our study aimed to identify the most critical issues users face when interacting with the traditional SQL querying mechanism, to what extent a QBE system like SQUID can alleviate these challenges, how effective SQUID is over a variety of data exploration tasks, and what the possible pain points of SQUID are. Details of the study design are in Appendix A.5.

3.7.1 Quantitative Results From Controlled Experiment

We now present the quantitative results of the controlled experiment study, summarizing our findings.

SQUID is generally more effective than SQL in generating accurate results. To quantitatively measure the quality of the results produced by both SQUID and SQL, we checked them against the ground-truth results (discussed in Section A.5.4.1). We used three widely used correctness metrics to quantify the result quality: precision, recall, and F1 score. On average, we found SQUID to be more effective in generating accurate results than SQL (Figure 3.16). For all four tasks, on average across participants, results obtained with SQUID achieved significantly higher precision than the results obtained with SQL. SQUID achieved higher recall than SQL for the two objective tasks (Disney and Marvel). While SQUID's recall for the subjective tasks (Funny and Strong) was lower than SQL,



(a) SQUID achieves higher precision than SQL.

(b) SQUID achieves higher recall than SQL in two tasks.

(c) SQUID achieves higher F1 score than SQL for all tasks.

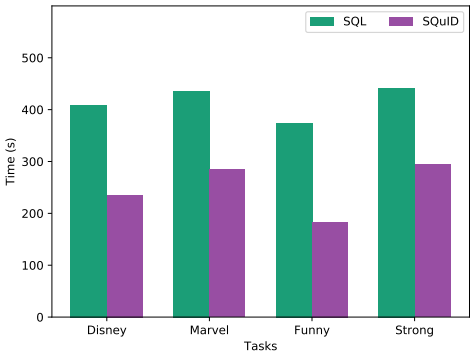
Figure 3.16: SQUID vs. SQL in terms of average precision, recall, and F1 score.

	Precision		Recall		F1 Score	
Task	p-value	t	p-value	t	p-value	t
Disney	**0.004**	3.0781	**0.0389**	2.1457	0.151	1.468
Marvel	0.1047	1.6669	0.0588	1.9554	0.7195	0.3621
Funny	**0.0001**	4.3845	**0.0042**	-3.0751	**0.0**	8.6225
Strong	**0.011**	2.6935	0.1751	-1.3859	**0.0**	6.4942

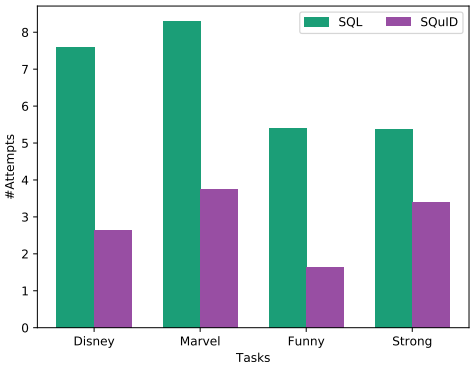
Figure 3.17: t test results for precision, recall, and F1 score. Out of 12 findings, 7 are statistically significant. In all cases, df = 33.

note that SQL's precision for those tasks was close to 0. This is simply because the SQL queries the participants wrote for those tasks were very imprecise and returned a very large number of results (e.g., all actors in the database). While such general queries can happen to contain a large portion of the correct results (hence the high recall), they contain an extremely large number of irrelevant results making them poorly suited for this retrieval task. In terms of F1 score, SQUID always achieved higher values than SQL implying its effectiveness over SQL for generating more accurate results. The result of t-tests for these findings are shown in Figure 3.17. Out of the 12 findings, 7 are statistically significant with a p-value less than 0.05.

Participants were more efficient with SQUID than SQL. SQUID helped the participants solve the tasks more quickly (Figure 3.18(a)) and with fewer attempts (Figure 3.18(b)) than SQL. On average, the participants were able to solve the tasks using SQUID about 200



(a) Time required to solve each task.



(b) Number of attempts required to solve each task.

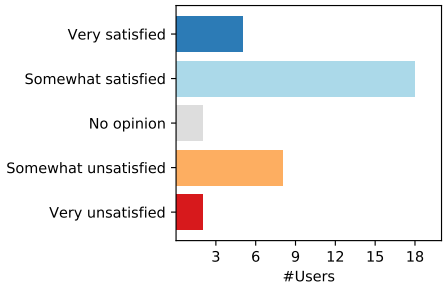
Figure 3.18: Comparison of SQL vs. SQuID in terms of effort (average time required and average number of attempts) for solving the same set of tasks.

	Task completion time		#Attempts	
Task	p-value	t	p-value	t
Disney	**0.0014**	-3.5	**0.0**	-4.7578
Marvel	**0.0146**	-2.5767	**0.0008**	-3.6985
Funny	**0.0008**	-3.7105	**0.0007**	-3.7441
Strong	**0.0132**	-2.6206	0.0595	-1.9518

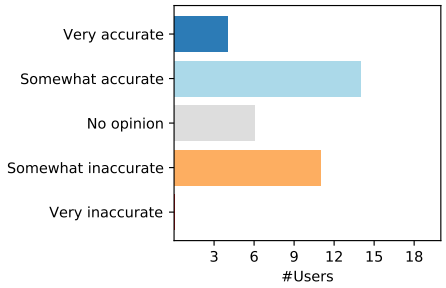
Figure 3.19: t test results for task completion time and number of attempts. 7/8 findings are statistically significant (df = 33).

seconds faster than when using SQL. Participants were also able to solve the tasks with about 4 fewer attempts while using SQuID compared to SQL. The results of t-test of these findings, shown in Figure 3.19, signify that most are statistically significant with a p-value less than 0.05.

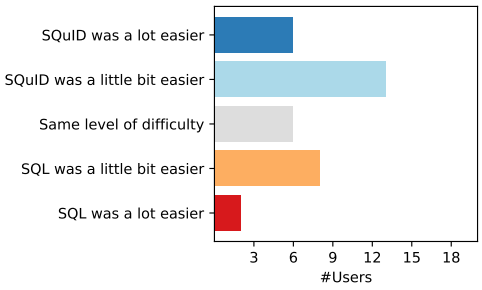
Participants generally found SQuID easier to use and more satisfying, but still preferred SQL. Figures 3.20(a) and 3.20(b) show self-reported overall satisfaction with the results produced by SQuID and SQL, respectively. Generally, participants found the results produced by SQuID more satisfying than the results produced by SQL. Out of the 35 participants, 23 were somewhat or very satisfied with SQuID. In contrast, 18 reported that



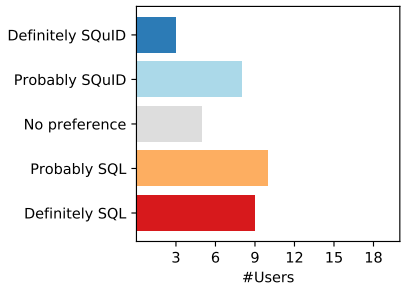
(a) Satisfaction with SQUID results



(b) Accuracy of SQL results



(c) Usability



(d) Preference

Figure 3.20: Comparison of SQUID vs. SQL in terms of various metrics (self-reported).

the results produced by SQL were somewhat or very accurate. However, we found that the self-reported satisfaction does not correlate with the actual correctness of the results (measured in terms of precision, recall, and F1 score), and in fact, the participants generally did better with SQUID than SQL, although they did not always realize it. Figure 3.20(c) shows self-reported overall evaluation comparing SQUID and SQL in terms of ease of use. Out of the 35 participants, 19 reported that SQUID was easier, 6 reported that they had the same level of difficulty, and 10 reported that SQL was easier.

However, despite reporting that SQUID was easier to use and the results were more satisfying, the participants were still leaning towards SQL as a preferred mechanism for data exploration. Figure 3.20(d) shows self-reported overall preference between SQUID and SQL, where 11 reported that they would prefer SQUID while 19 reported that they would prefer SQL. Five participants reported no preference.

3.7.2 Qualitative Results from Interview Study

We now report the key findings from our interview study. More details are in Appendix A.6.

- Studying the schema is challenging, even for SQL experts.
- SQL requires stricter syntax, which makes writing queries difficult.
- SQL requires parameter tuning for subjective tasks; SQUID alleviates this.
- SQUID produces precise results, which is preferred for data exploration.
- SQUID's interactivity helps users to enrich examples.
- Domain familiarity is crucial to evaluate the results, for both SQUID and SQL.

3.7.3 Limitations and Suggestions for Extension

Our study results indicate that SQUID effectively helped users with various levels of SQL familiarity perform their tasks faster and more efficiently. Additional work is needed to study the impact of QBE systems further. While our goal was to draw a comparison between traditional SQL querying and QBE systems, additional studies might investigate how complete novices (users with no SQL expertise) use QBE systems. Furthermore, one can expand the list of tasks to better tease apart the impact of using QBE systems for various task types. From the interviewees' feedback, we extracted a few suggestions to improve the user experience of QBE systems:

Exposing internal mechanism for explainability. SQUID can explain how the SQL queries were synthesized from the user examples by exposing the particular semantic similarities that the system discovers across the examples, and its confidence in each similarity being intended. This can also guide users in revising their examples to emphasize borderline semantic similarities that SQUID missed, or diversify examples to avoid coincidental similarities among the examples.

Tuple suggestions to enrich examples. A few interviewees reported that it would be helpful if SQUID could suggest a few tuples that the user may consider adding to the examples. Such a tuple-suggestion mechanism will help the users supply additional diversified examples, in case the users lack domain knowledge.

Interaction with the results for feedback. Another idea is to allow the users to interact with the results produced by QBE systems: the user will accept or reject a few result tuples which will act as feedback to the system. This will help QBE systems learn the user intent better.

Extensive user study. In general, more extensive user studies are needed to evaluate all these additional features and determine whether they contribute positively to the users' trust and satisfaction in QBE systems.

3.8 Summary and Future Work

In this chapter, we focused on the problem of query intent discovery from a set of example tuples. We presented SQUID, a system that performs query intent discovery effectively and efficiently, even with few examples in most cases. The insights of our work rely on exploiting the rich information present in the data to discover similarities among the provided examples, and distinguish between those that are coincidental and those that are intended. Our contributions include a probabilistic abduction model and the design of an abduction-ready database, which allow SQUID to capture both explicit and implicit semantic contexts. Our work includes an extensive experimental evaluation of the effectiveness and efficiency of our framework over three real-world datasets, case studies based on real user-generated examples and abstract intents, and comparison with the state-of-the-art query reverse engineering technique and with PU-learning. Our empirical results highlight the flexibility of our method, as it is extremely effective in a broad range of scenarios.

Our comparative user studies found that database users, with varied levels of prior SQL expertise, are significantly more effective and efficient at a variety of data exploration tasks with SQUID over the traditional SQL querying mechanism that requires database schema understanding and manual programming. Our results indicate that SQUID eliminates the barriers of familiarizing oneself with the database schema, formally expressing the semantics of an intended task, and writing syntactically correct SQL queries. The key takeaway of this work is that in a programming-by-example tool like SQUID, even a limited level of domain expertise (knowledge of a subset of the desired data) can substantially help overcome the lack of technical expertise (knowledge of SQL and schema) in data exploration and retrieval. This indicates that programming by example can lead to the democratization of complex computational systems and make these systems accessible to novice users while aiding expert users as well. Our studies validate some prior results over other PBE approaches but also contribute new empirical insights and suggest future directions for QBE systems to further increase system explainability and user trust.

Beyond the ones extracted from the user study, there are several possible improvements and research directions that can stem from our work, including smarter semantic context inference using log data, example recommendation to diversify the examples (which will improve abduction), techniques for adjusting the depth of association discovery, on-the-fly αDB construction, and efficient αDB maintenance for dynamic datasets.

CHAPTER 4

SUMMARIZING DOCUMENTS BY EXAMPLE (SUDOCU)[1]

To investigate the effectiveness of by-example interactions beyond relational databases, we explored the area of text document summarization by example. Document collections, such as Wikipedia, contain a wealth of information that can assist in many tasks. Yet, finding the right information quickly and easily is still a big challenge, despite all the advances in search engine technology, natural language processing, and machine learning. Consider the following scenario:

Example 4.1 (Trip planning)**.** *Arnob wants to plan visits to interesting places around the USA. She wants to know interesting locations and typical weather conditions for each state, but finding this information on the Web for 50 states is tedious and time-consuming. She knows that Wikipedia contains all the information she needs, but each page is large and full of facts that are not relevant to her intent (e.g., demographics, law, etc.). Arnob can manually extract relevant summaries of at most 3 pages, by selecting a small set of sentences that correspond to her specific information needs (interesting places and weather). But to thoroughly research her options, she needs an automated way to do this for the remaining 47 states.*

Surprisingly, today's technology cannot help Arnob! A search engine, like Google, is good at finding which web pages are likely to contain relevant information, but it would require many queries and Arnob would need to be very thoughtful about search keywords in order to collect the relevant information for all 50 states.

---

[1]The work for SUDOCU was done in collaboration with Ph.D. students Matteo Brucato and Nishant Yadav, and M.S. students Oscar Youngquist and Julian Killingback.

Arnob tried to use Natural Language Processing (NLP) and Machine Learning (ML) techniques and found that *text summarization* tools may be helpful. However, most text summarization tools are "generic": they produce summaries that are not tailored for her personal preferences and specific information needs. The summaries she obtained from these tools did not cover all important aspects of her task, but rather provided general information about the state's politics, law, education, etc. Arnob found that some summarization tools can be tailored with a user intent, and require a natural language question to express it. She picked a question answering system, like Alexa, and issued the following question: "What are some interesting places in Massachusetts and how extreme is the weather there?" Unfortunately, the system could not understand what Arnob meant by "interesting places"—since interestingness is a very personal concept—and returned her sentences about places of general interest: MIT, Harvard Square, and Boston Library.

Arnob is interested in natural sites: parks, lakes, mountains, seas, etc. While particular preferences may be hard to express precisely with a query, it is easy for Arnob to identify relevant sentences within a document. For example, Arnob selected the following sentences from Utah's Wikipedia page as most relevant to her needs:

Example 4.2 (Personalized summary of Utah)**.** *The state of Utah relies heavily on income from tourists and travelers visiting the state's parks and ski resorts. Today, Utah State Parks manages* 43 *parks and several undeveloped areas totaling over* $95,000$ *acres of land and more than* $1,000,000$ *acres of water. With five national parks (Arches, Bryce Canyon, Canyonlands, Capitol Reef, and Zion), Utah has the third most national parks of any state after Alaska and California. Temperatures dropping below* 0 *hould be expected on occasion in most areas of the state most years.*

She would like to extract something similar to the summary of Example 4.2 for each of the 50 states. Luckily, she can now use SUDOCU, a personalized DOCUment SUmmarization system, that enables users to specify their summarization intent by a few *example summaries* and produces *personalized* summaries for new documents. SUDOCU is an instance

of a *query-by-example* system [94], tailored for text document summarization. The key motivation of SUDOCU is that asking a user to provide examples of their desired answers, rather than vague questions, is a more effective way to learn the true intent, especially for a complex summarization intent involving multiple topics, e.g., interesting places *and* weather.

In this work, we consider extractive summarization [348], which involves selecting the set of sentences from a text document that best summarizes its information content. In many real-world applications, *generic* summaries [145], which are not focused on any specific topic, are insufficient, especially when users require summaries that are tailored to their specific information needs, i.e., their *intent*. Existing approaches for such applications rely on *query-based* summarization [67], where the sentences included in the summary are conditioned on a specific intent expressed as a query.

We posit that query-based summarization works best for *objective* or unambiguous intents, i.e., when there exists only one possible correct summary for a given query. For example, suppose that a user wants to summarize the Wikipedia page for the state of California. If they want to gather information about the weather, the intent is very objective and one can expect there would only be one correct summary. In this case, the user can easily issue a query such as "What is California weather like?"

However, queries are not always the best way to communicate *subjective* intents. For example, suppose that the user now issues the query "Which places are interesting to visit?" There are many ways to interpret what makes a place "interesting", which makes the query ambiguous, and, thus, the correct answer (summary) is very subjective. For example, one user may prefer natural places while another may prefer museums. In a query-based system, the user would need to devise very long and complex queries to clarify their subjective intent, but still may fail to communicate the intent effectively.

An alternative way to express a summarization intent is via user-constructed example summaries. *Example-driven* summarization seems especially well suited for expressing

subjective intents, as it is often easier to construct an example summary than it is to write a complex query. The summarization system can infer the user intent from the example summaries, and then summarize unseen documents based on the inferred intent.

We present SUDOCU [92], an end-to-end system that achieves *example-driven personalized* document summarization, that allows subjective summarization. The key idea is to view summarization as a combinatorial optimization problem where we want to extract an optimal set of sentences to form the summary, subject to the constraint that the summary's overall topic coverage should be *close* to that of the examples. We model topics of the documents using a standard LDA approach [41], adapt our prior work for example-driven semantic similarity discovery [94] to create the constraints, and solve the resulting integer linear program using our prior techniques for scalable package queries [49].

We proceed to discuss how SUDOCU contrasts with prior art (Section 4.1) and provide a solution sketch (Section 4.2)

4.1 Contrast with Prior Art

In *query-based* summarization, users specify their intent in the form of an unstructured query, typically, a natural language question. For example, the question "What are some interesting places?" is very subjective, as different people consider different places as interesting. For a nature enthusiast, parks, lakes, oceans, and mountains are interesting; for an art enthusiast, museums, concerts, and plays are interesting. SUDOCU allows the user to provide precise and concrete examples of the type of summaries they want, and does not require large training data. A possible way to adapt query-based summarization for example-driven summarization is to infer the underlying natural-language query from the example summaries, and then use an existing tool. However, computers understand structured queries with clear semantics, which can easily be constructed from examples, much better than natural language queries, so an example-based approach is both simpler and more accurate.

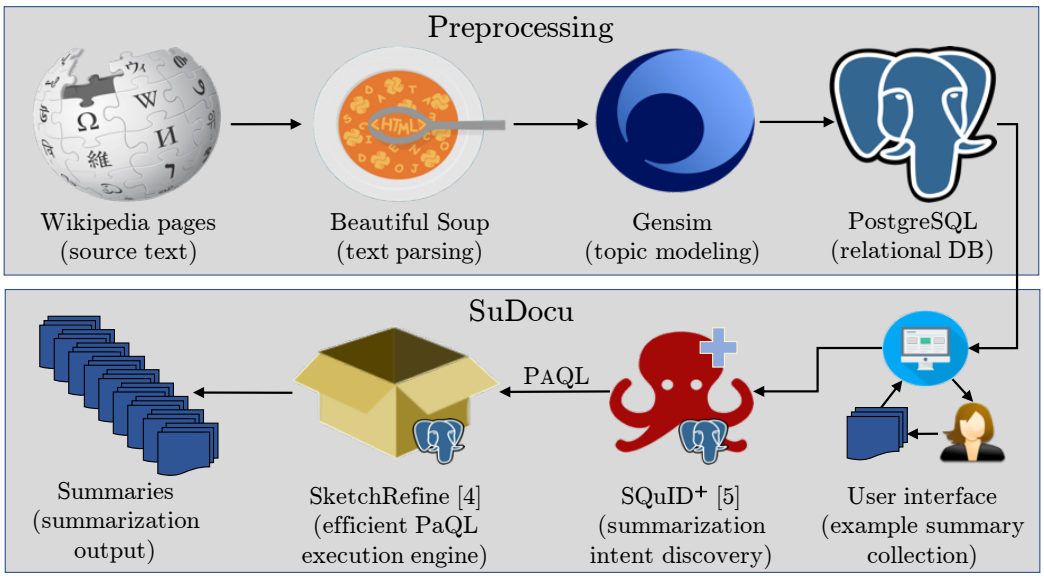


Figure 4.1: The SUDOCU architecture. SUDOCU combines SQUID$^+$ and SKETCHREFINE in a novel way to summarize documents by example.

A shortcoming of the foregoing sentence-selection approaches is that they consider candidate sentences in isolation, rather than trying to select a set of sentences that collectively form a good summary. The problem of selecting the best *set* of sentences can be formulated as an integer program. Our formulation can capture the summarization intent from the example summaries using constraints on how much each topic should be "covered" by the summary; roughly speaking, the coverage should resemble that of the user-provided examples. Also, because of the combinatorially large number of possible summaries, the formulation in [209] cannot generally scale to large dataset sizes. We use the SKETCHREFINE algorithm [49] to scale the resulting integer linear program to very large datasets.

4.2 SUDOCU Algorithms

Figure 4.3 provides a screenshot of the SUDOCU interface. We now provide the design and implementation details of SUDOCU. Figure 4.1 depicts SUDOCU's end-to-end pipeline. SUDOCU pre-processes a corpus of documents by extracting all the sentences, automatically identifying all the topics, and assigning topic scores to each sentence. After

ID	Topic	Top related words and their associated weights
1	*politics*	(governor, 0.015), (election, 0.013), (vote, 0.011), (democratic, 0.011), (majority, 0.009), (presidential, 0.008)
2	*legislature*	(century, 0.012), (passed, 0.011), (legislature, 0.010), (constitution, 0.009), (created, 0.007), (law, 0.006), (political, 0.006)
3	*urbanization*	(population, 0.077), (largest, 0.052), (city, 0.029), (percent, 0.019), (metropolitan, 0.012), (capital, 0.011), (people, 0.011)
4	*economy*	(major, 0.027), (economy, 0.018), (largest, 0.013), (industry, 0.013), (billion, 0.011), (production, 0.011), (oil, 0.009)
5	*demography*	(american, 0.029), (people, 0.021), (native, 0.018), (french, 0.015), (century, 0.015), (settlers, 0.012), (tribes, 0.010)
6	*climate*	(climate, 0.017), (feet, 0.011), (temperature, 0.010), (rail, 0.010), (forests, 0.009), (summer, 0.009), (winter, 0.009)
7	*location*	(north, 0.035), (west, 0.033), (south, 0.030), (east, 0.029), (southern, 0.022), (eastern, 0.020), (region, 0.020)
8	*taxes*	(tax, 0.056), (income, 0.030), (rate, 0.029), (ranked, 0.021), (nation, 0.021), (sales, 0.017), (average, 0.015), (capita, 0.014)
9	*education*	(government, 0.039), (school, 0.029), (county, 0.025), (public, 0.025), (federal, 0.023), (schools, 0.022), (law, 0.016)
10	*general*	(national, 0.007), (major, 0.006), (popular, 0.005), (system, 0.004), (founded, 0.004), (home, 0.004), (construction, 0.004)

Figure 4.2: Topics of Wiki pages of 50 states (extracted using topic modeling), their intuitive meaning, and top related words with associated weights.

preprocessing, the user can interact with SUDOCU's interface and issue example summaries to specify their intent. We first describe how we model the user intent, and then discuss preprocessing, summarization intent discovery, and summary generation.

4.2.1 Modeling Personalized Extractive Summaries

Following prior work on text summarization [209], we model the personalized summarization as an optimization problem. Given the example summaries, we define the optimal summary as the one that maximizes a user-defined merit score (discussed later) such that the topic-coverage of the summary is similar to that of the example summaries. In SUDOCU, we construct a linear constraint on topic-coverage for each topic, allowing scalable solution methods.

We express the optimization problem as a *package query* [49]. A *package* (summary) is a collection of tuples (sentences) from a relation (document) that (a) individually satisfy base predicates (traditional SQL selection predicates), and (b) collectively satisfy global predicates (package-specific predicates). A package query comprises base and global predicates that define the set of feasible packages and an objective function that defines a preference ranking among them. The *Package Query Language* (PaQL) is a simple extension to SQL that allows for the easy specification of global constraints and objectives.

4.2.2 Preprocessing

Preprocessing involves two steps: parsing the documents for extracting sentences, and topic modeling.

Sentence extraction. The first step of SUDOCU involves extracting sentences from documents. We use `Beautiful Soup` [32], a library for extracting content from HTML pages. We then identify all of the topics in the extracted sentences.

Topic modeling. We use *Latent Dirichlet Allocation (LDA)* topic model [41], in which a learned (latent) topic is represented as a set of weights assigned to the words in the vocabulary, and a sentence is viewed as a set of weights assigned to the topics. (Sentences here play the role of documents in [41].) The weight of a word (resp., topic) represents its relative importance to the topic (resp., sentence). For our implementation, we used `Gensim`, a standard NLP library that offers LDA-based topic modeling. Figure 4.2 shows the topics learned from the Wikipedia pages of 50 US states. In general, we can plug in any topic modeling technique into SUDOCU.

The LDA topic weight of a sentence scores the relevance of a particular sentence to a particular topic. For example, the first sentence from the Example Summary 4.2, "The state of Utah relies heavily on income from tourists and travelers visiting the state's parks and ski resorts", would score high on "economy" and low on "education". Once sentences are

encoded into the topic space, a sentence s (within document d) and its merit and topic-wise scores form a tuple of the form $\langle d, s, m\_score, s.T_1, s.T_2, \ldots, s.T_m \rangle$, where m_score is the merit score of s (see below) and $s.T_j$ denotes the score of s against topic T_j. We store these tuples into a PostgreSQL database.

4.2.3 Summarization Intent Discovery

To discover the summarization intent from example summaries, we extend the example-driven semantic similarity discovery approach of SQuID [94]; we call our extension SQuID$^+$. Whereas SQuID synthesizes SQL selection queries to retrieve tuples that are similar to user-specified example tuples, SQuID$^+$ synthesizes package queries to retrieve summaries (i.e., sets of tuples) that are similar to the user-specified example summaries. SQuID would treat a single sentence as an example tuple; in contrast, SQuID$^+$ considers a *set* of sentences (summary) as an example package. Further, it aims to retrieve the summary with the highest utility (maximizing total m_score) among these similar summaries.

To discover similarities among example summaries, we compute the topic-wise aggregate score for each example summary by summing the topic-wise scores of its sentences. That is, the score of example E_i against topic T_j is $E_i.T_j = \sum_{s \in E_i} s.T_j$. Now we specify the global topic-coverage predicate for T_j, given a set of examples $\{E_1, E_2, \ldots\}$, as follows:

$$\texttt{SUM}(T_j) \ \texttt{BETWEEN} \ \min_i E_i.T_j \ \texttt{AND} \ \max_i E_i.T_j$$

Thus the aggregate score for each topic T_j in the summary must lie between the minimum and maximum aggregate scores in the examples; i.e., viewing each E_i as a point in topic space, the summary must lie within the bounding hyperrectangle of the examples. One can further fine-tune the above constraint bounds: e.g., if most examples scored very high against a topic and only a few scored low, increase the minimum score threshold for that topic. In general, SuDocu can accept any package constraint derivation mechanism and is not limited to SQuID$^+$.

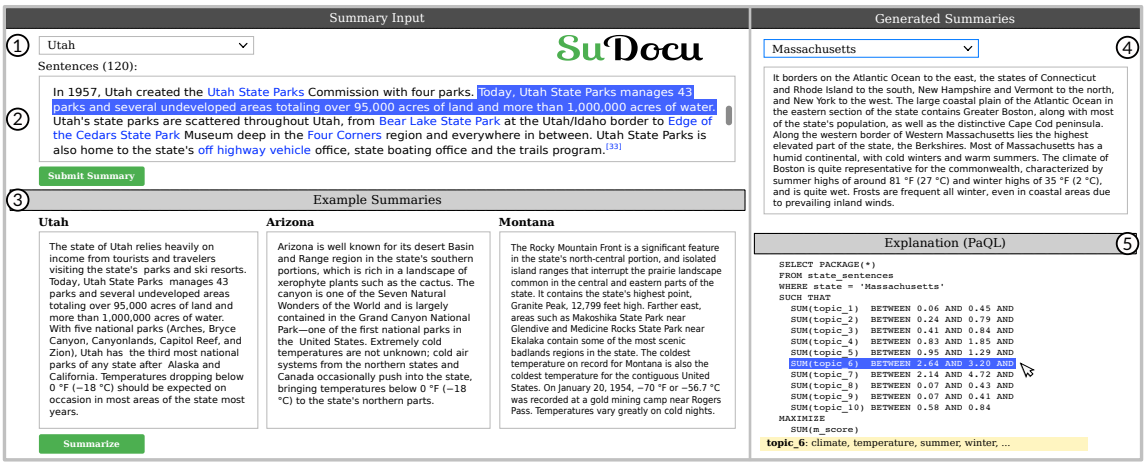


Figure 4.3: The SUDOCU interface: ① the user selects a document for manual summarization, ② the user selects sentences from the document to construct an example summary, ③ the user views the example summaries, edits them if necessary, and submits them to request for summarization intent discovery, ④ the user specifies a new document to summarize and SUDOCU produces a personalized summary of it, ⑤ PaQL query that captures the summarization intent.

From the set of "feasible" summaries that satisfy the topic constraints, we want to select the "best" one. More precisely, we aggregate a per-sentence, user-defined "merit" score over the sentences in a summary to obtain the summary's merit score; we then seek the feasible summary having the highest merit score. Different definitions of merit are possible. If, e.g., the merit score of every sentence is -1, then maximizing the merit is equivalent to finding the shortest feasible summary. In our implementation, the merit score $m\_score(s)$ of a sentence $s = (w_1, \ldots, w_J)$ comprising J words (with stop words excluded) is defined as $m\_score(s) = \sum_{j=1}^{J} F(w_j)$, where $F(w)$ is the normalized frequency of word w in the corpus. Thus the more "important" (high corpus-frequency) words that a sentence contains, the higher its merit score.

The complete PaQL query is formulated as in Figure 4.3. Each tuple of the input relation corresponds to a sentence, and the attributes comprise the sentence and document IDs, along with the merit and topic-wise scores. The objective function to be maximized is the summary merit score `SUM(m_score)`, and the `WHERE` clause ensures that only sentences from the document of interest are considered.

4.2.4 Efficient Summary Generation

Once the PaQL formulation of a package query is completed, the last step is to execute it. Package queries are combinatorial in nature, and solving them in general is NP-hard. If the problem is small enough, we can translate a package query directly into an equivalent integer linear program that can be solved with off-the-shelf softwares. For each tuple t_i in the input relation, the translation assigns a binary decision variable x_i corresponding to the inclusion/exclusion of t_i in the answer package. When there are so many candidate sentences that the solver either cannot load the problem in main memory or fails to find a solution, we apply the SKETCHREFINE algorithm [49], a divide-and-conquer approach that returns a near-optimal solution of the PaQL query having a provable approximation guarantee. SUDOCU then presents the optimal set of sentences as the summary to the user, along with the PaQL query that encodes the summarization intent.

4.3 SUBSUME: A Dataset for Subjective Summary Extraction from Wikipedia Documents

SUDOCU is particularly suitable for the task of *subjective* summarization that requires generation of summaries that are tailored to the user intent. Query-based methods approach this task by expressing the intent as a query, but fall short when query interpretation is user-dependent, and, thus, subjective. Several datasets exist for summarization with objective intents, where a single summary for each document can answer an intent. However, no datasets exist for subjective intents, where the summary of a document can vary across users having different intents.

Ground-truth data is crucial to rigorous development and comparison of methods for subjective summarization, but all prior datasets have been oriented toward objective summarization, and contain exactly one summary per document. To evaluate SUDOCU and compare it with other baselines, we constructed SUBSUME, the first comprehensive and large-scale, manually curated dataset for the task of SUBjective SUMmary Extraction.

Mostly Objective	
(I1)	How is the weather of the state?
(I2)	How is the government structured in this state?
(I3)	What is the state's policy regarding education?
(I4)	What are the available modes of transport in this state?
Balanced Subjective/Objective	
(I5)	What drives the economy in this state?
(I6)	What are the major historical events in this state?
Mostly Subjective	
(I7)	What about this state's arts and culture attracts you the most?
(I8)	Which places seem interesting to you for visiting in this state?
(I9)	What are some of the most interesting things about this state?
(I10)	What are the main reasons why you would like living in this state?

Figure 4.4: Intents in SUBSUME vary between mostly objective to mostly subjective.

SUBSUME contains 2,200 ⟨`document`, `intent`, `summary`⟩ triplets over 48 Wikipedia pages, with 10 intents of varying subjectivity, provided by 103 individuals over Mechanical Turk. We analyze the properties of SUBSUME and use it to evaluate several baselines, including SUDOCU, for subjective, extractive summarization, indicating room for improvement and motivating new research for the task of subjective document summarization.

We now describe our data collection process and design choices, and analyze some statistical properties of the dataset.

4.3.1 Intents

We devised 10 intents with different degrees of subjectiveness, ranging from mostly objective to mostly subjective, as shown in Figure 4.4.

4.3.2 Documents

As the source documents, we used English Wikipedia pages of 48 U.S. states.[2] We parsed the pages to get text content from paragraph tags, and extracted sentences using

[2]We removed Nebraska and Wyoming as their pages did not have enough content with respect to the intents in Figure 4.4.

Punkt sentence tokenizer from the NLTK library.[3] Our corpus includes homogeneous documents to allow summarization of all documents with respect to all intents. In particular, we chose the Wikipedia pages for the states in the USA because they are homogeneous and contain information on wide range of topics.

4.3.3 Interface and Tasks

We collected extractive summaries of the documents using a custom interface on Amazon Mechanical Turk (MTurk). Our interface allowed the workers to search the document for keywords, click on a sentence to include it to the summary, and remove a sentence from the summary.

Task. Each MTurk task (HIT) required a worker to extract sentences from 8 documents to best summarize them according to a given intent, resulting in 8 $\langle \texttt{document}, \texttt{intent}, \texttt{summary} \rangle$ triplets. To generate unique HITs, we partitioned the set of 48 documents into 6 disjoint sets, each containing 8 documents. We then paired each of the 6 sets with each of the 10 intents, resulting in 60 unique HITs. We repeated the above procedure 5 times to obtain a total of 300 HITs. Out of these 300 HITs, 25 were rejected upon manual inspection (due to poor-quality summaries). The remaining 275 HITs contained 8 summaries each, resulting in a total of 2,200 $\langle \texttt{document}, \texttt{intent}, \texttt{summary} \rangle$ triplets. We allowed workers to participate in multiple HITs as long as they were not identical: either the document-set or the intent was different.

4.3.4 Dataset

We now proceed to discuss the collected data. We start by presenting the data format with example summaries, and then provide analysis of the data collected.

---

[3] https://www.nltk.org/_modules/nltk/tokenize/punkt.html

Intent: Which places seem interesting to you for visiting in this state?

Summary of Colorado: The northwestern corner of Colorado [. . .] contains part of the noted Dinosaur National Monument, which not only is a paleontological area, but is also a scenic area of rocky hills, canyons, arid desert, and streambeds [. . .] There are also a number of established film festivals in Colorado, including Aspen Shortsfest, Boulder International Film Festival, Castle Rock Film Festival [. . .] The state's diverse geography and majestic mountains attract millions of tourists every year [. . .] Colorado is home to 4 national parks, 8 national monuments, 2 national recreation areas, 2 national historic sites, 3 national historic trails [. . .].

Interpretation: Things I would like to do includes nature spots I would like for outdoor recreation, cities for touring and special events.

Strategy: I looked for the capitol, any other interesting cities, nature, sports, cultural experiences and special events native to the state.

Keywords: tourism, national park, Denver, Mesa

Figure 4.5: A datapoint of SUBSUME.

Data format. We provide SUBSUME in a format to support both query-based and example-driven approaches, as described earlier. The result of each completed HIT gives us the following information and contributes to 8 data points in SUBSUME: (1) the intent text (one of I1–I10 in Figure 4.4), (2) one summary for each of the 8 documents in the HIT, (3) interpretation of the intent by the worker, (4) description of summarization strategy followed by the worker, (5) the keywords typed in the search box by the worker while selecting sentences, (6) time-stamps indicating when each sentence was added to the summary, (7) percentage of the document the worker viewed, and (8) optional demographic information of the worker. A datapoint, containing one of the 8 summaries within a HIT result, is shown in Figure 4.5; we omit (6)–(8) for brevity.

Dataset analysis. Figure 4.6 shows statistics of the dataset grouped by intents. We quantify the *subjectiveness* of an intent as follows: Let $\mathcal{S}_{i,d}$ be the set of summaries constructed by all different workers for an intent i and document d. We first compute pair-wise ROUGE-L F_1 scores (normalized between 0 and 100) for all pairs of summaries from $\mathcal{S}_{i,d}$. We define $\texttt{Sim}_{i,d}$ as the average of these scores, measuring the similarity of all pairs of

Statistic	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
# Summaries	240	216	232	240	232	224	192	200	208	216
Avg. # sentences per summary	11.4	12.7	8.6	10.5	10.8	13.7	11.3	9.3	13.4	11.2
Avg. # words per summary	314.8	285.8	227.2	278.6	288.9	380.1	319.5	274.6	375.1	304.6
Subjectiveness Score	22.7	34.2	35.0	35.6	47.4	58.7	55.7	56.9	74.3	73.2

Figure 4.6: Statistics of the SUBSUME dataset across 10 different intents.

Metric	Example-Driven (EX)				Query-Based (QB)		
	KEYWORD	TEXTRANK	SBERT	SUDOCU	KEYWORD	TEXTRANK	SBERT
ROUGE-1	31.20	25.30	51.17	33.28	30.50	25.06	41.73
ROUGE-2	7.88	12.59	34.21	16.25	10.09	7.89	21.91
ROUGE-L	17.09	15.57	38.35	21.02	17.18	13.24	28.035

Figure 4.7: Average F_1 scores over different ROUGE metrics for baseline techniques.

summaries for document d and intent i. We define the *subjectiveness score* (inverse of similarity) for intent i using the following formula:

$$\texttt{Subj}_i = 100 - \frac{\sum_d \texttt{Sim}_{i,d}}{\sum_d 1}$$

The higher the subjectiveness score for a given intent, the lower the similarity among summaries for that intent, thus indicating higher subjectiveness. Our classification of intents in Figure 4.4 aligns well with this subjectiveness score, as shown in Figure 4.6. For instance, "How is the weather of the state?" (I1) scores the lowest (22.7) and "What are some of the most interesting things about this state?" (I9) scores the highest (74.3).

4.3.5 Experiments

In this section, we present evaluation of a few baseline summarization techniques over the SUBSUME dataset in two settings: query-based (QB) and example-driven (EX). Recall that for every $(user, intent)$ pair, SUBSUME consists of summaries of 8 documents chosen uniformly at random from a pool of 48 documents. In the query-based setting, the baselines summarize the documents using only the query (intent text), and we evaluate on all 8 documents. In the example-driven setting, we use summaries of 5 documents, chosen

at random from the 8 summaries, as example summaries to learn the user's intent. We then evaluate the technique in terms of the generated summaries, according to the inferred intent, for the remaining 3 documents.

In both settings, we first compute average performance on test documents for each $(user, intent)$ pair and then average across all such pairs in the dataset. For each $(user, intent)$ pair, we get either 8 (QB) or 3 (EX) summaries. We report F_1 scores of the ROUGE-1, ROUGE-2, and ROUGE-L metrics [208] for all of the baselines in Figure 4.7.

4.3.5.1 Baselines

We implement the following unsupervised baselines, which summarize a new test document based on either the query text (QB) or the example summaries of other documents (EX), other than SUDOCU. For all baselines, we pre-processed the text within the documents by removing all stop-words and converting all characters to lower case; except for SBERT, which does not require stop-words to be removed. For these experiments, we set the value of $k = 10$ for the Keyword-based and SBERT-embeddings-based methods. Additionally, for the Keyword-based approach, we set the cut-off threshold to the average *keyword-coverage* score of all the sentences in the document. Lastly, the number of latent topics extracted by LDA for SUDOCU was limited to 10.

KEYWORD extracts keywords from the example summaries or query text (intent). Each sentence in the test document is scored based on the number of keywords contained in it followed by shortlisting sentences with score greater than a threshold (t_k). The summary is constructed using the top-k sentences from this candidate set ranked based on their TF-IDF scores.

TEXTRANK is an unsupervised, ranking-based, extractive summarization approach [235]. We modify it to use increased relative frequencies in its TF-IDF embeddings for keywords that appear in the user-provided example summaries or query text.

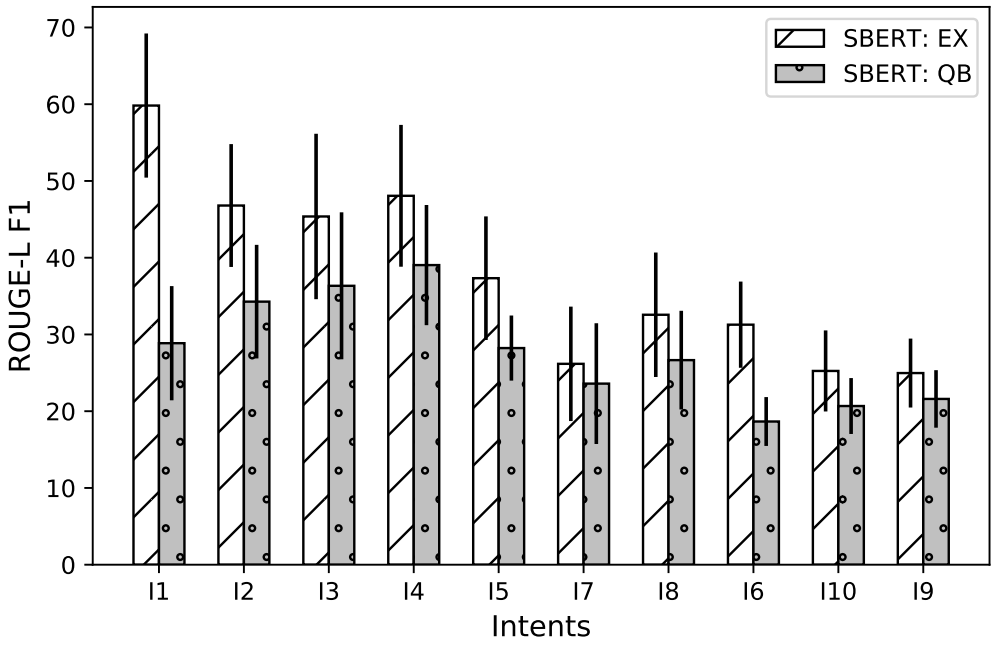


Figure 4.8: ROUGE-L F_1 for SBERT in example-driven (EX) and query-based (QB) settings for each intent. From left to right, intents are ordered from least subjective to most subjective according to their subjectiveness score shown in Figure 4.6. The correlation between the subjectiveness score and the F_1 score for SBERT EX and SBERT QB is -0.95 and -0.78 respectively.

SBERT embeds example summaries or query text and sentences in test documents using SBERT [272]. It scores each sentence based on its cosine similarity with average embedding of the example summaries or the query text and computes a summary using top-k high-scoring sentences in the document.

4.3.5.2 Results

Figure 4.7 shows the performance of each baseline averaged over all $(user, intent)$ pairs. Example-driven baselines consistently outperform their query-based counterparts with SBERT being the top performing method. This demonstrates effectiveness of example-driven approaches for subjective summarization over query-based approaches.

Figure 4.8 shows the average ROUGE-L F1-score SBERT achieves for each intent for both example-driven (EX) and query-based (QB) settings. For all intents, SBERT-

EX outperforms SBERT-QB. As we go from intents with low subjectiveness scores to intents with high subjectiveness scores, performance of SBERT decreases for both EX and QB. This shows how the summarization task becomes challenging with increase in subjectiveness of the intents, and indicates room for significant improvement for the task of subjective document summarization.

4.4 Summary and Future Work

In this chapter, we presented SUDOCU, a tool for personalized, subjective, and extractive text document summarization by example. We also presented a dataset SUBSUME for subjective summarization. Empirical evaluation suggests that SUDOCU, with its current form, does not outperform state of the art techniques for subjective summarization over the SUBSUME dataset. However, there is large room for improvement within different components of SUDOCU, and a direction for future work is to explore different variations of SUDOCU, such as fine tuning the bounds of topic constraints to model example similarities better and exploring other objective functions.

PART II: DATA UNDERSTANDING: CONFORMANCE CONSTRAINTS

CHAPTER 5

CONFORMANCE CONSTRAINTS AND THEIR APPLICATIONS

A machine-learned model typically works best if the serving dataset follows the profile of the dataset the model was trained on; when it doesn't, the model's inference can be unreliable. One can profile a dataset in many ways, such as by modeling the data distribution of the dataset, or by finding the (implicit) *constraints* that the dataset satisfies. Distribution-oriented approaches learn data likelihood (e.g., joint or conditional distribution) from the training data, and can be used to check if the serving data is unlikely. An unlikely tuple does not necessarily imply that the model would fail for it. The problem with the distribution-oriented approaches is that they tend to overfit, and thus are overly conservative towards unseen tuples, leading them to report many such false positives.

We argue that certain constraints offer a more effective and robust mechanism to quantify trust of a model's inference on a serving tuple. The reason is that learning systems implicitly exploit such constraints during model training, and build models that assume that the constraints will continue to hold for serving data. For example, when there exist high correlations among attributes in the training data, learning systems will likely reduce the weights assigned to redundant attributes that can be deduced from others, or eliminate them altogether through dimensionality reduction. If the serving data preserves the same correlations, such operations are inconsequential; otherwise, we may observe model failure.

We characterize datasets with a new data-profiling primitive, *conformance constraints*, and we present a mechanism to identify *strong* conformance constraints, whose violation indicates unreliable inference. Conformance constraints specify constraints over *arithmetic relationships* involving multiple numerical attributes of a dataset. We argue that a tuple's

	Departure Date	Departure Time [DT]	Arrival Time [AT]	Duration (min) [DUR]
t_1	May 2	14:30	18:20	230
t_2	July 22	09:05	12:15	195
t_3	June 6	10:20	12:20	115
t_4	May 19	11:10	13:05	117
t_5	April 7	22:30	06:10	458

Figure 5.1: Sample of the airlines dataset (details are in Section 5.5.2), showing departure, arrival, and duration only. The dataset does not report arrival date, but an arrival time earlier than departure time (e.g., last row), indicates an overnight flight. All times are in 24 hour format and in the same time zone. There is some noise in the values.

conformance to the conformance constraints is more critical for accurate inference than its conformance to the training data distribution. This is because any violation of conformance constraints is likely to result in a catastrophic failure of a learned model that is built upon the assumption that the conformance constraints will always hold. Thus, we can use a tuple's deviation from the conformance constraints as a proxy for the trust on a learned model's inference for that tuple. We proceed to describe a real-world example of conformance constraints, drawn from our case-study evaluation on *trusted machine learning* (TML).

Example 5.1. *We used a dataset with flight information that includes data on departure and arrival times, flight duration, etc. (Fig. 5.1) to train a linear regression model to predict flight delays. The model was trained on a subset of the data that happened to include only daytime flights (such as the first four tuples). In an empirical evaluation of the regression accuracy, we found that the mean absolute error of the regression output more than quadruples for overnight flights (such as the last tuple t_5), compared to daytime flights. The reason is that tuples representing overnight flights deviate from the profile of the training data that only contained daytime flights. Specifically, daytime flights satisfy the conformance constraint that "arrival time is later than departure time and their difference is very close to the flight duration", which does not hold for overnight flights. Note that this constraint is just based on the covariates (predictors) and does not involve the target attribute $delay$. Critically, although this conformance constraint is unaware of the regression task, it was still a*

good proxy of the regressor's performance. In contrast, approaches that model data likelihood may report long daytime flights as unlikely, since all flights in the training data (t_1–t_4) were also short flights, resulting in false alarms, as the model works very well for most daytime flights, regardless of the duration (i.e., for both short and long daytime flights).

Example 5.1 demonstrates that when training data has *coincidental* relationships (e.g., the one between AT, DT, and DUR for daytime flights), then ML models may *implicitly* assume them as *invariants*. Conformance constraints can capture such data invariants and flag non-conforming tuples (overnight flights) during serving.

Conformance constraints. Conformance constraints complement the existing data profiling literature, as the existing constraint models, such as functional dependencies and denial constraints, cannot model arithmetic relationships. For example, the conformance constraint of Example 5.1 is: $-\epsilon_1 \leq AT - DT - DUR \leq \epsilon_2$, where ϵ_1 and ϵ_2 are small values. Conformance constraints can capture complex linear dependencies across attributes within a *noisy* dataset. For example, if the flight departure and arrival data reported the hours and the minutes across separate attributes, the constraint would be on a different arithmetic expression: $(60 \cdot arrHour + arrMin) - (60 \cdot depHour + depMin) - duration$.

The core component of a conformance constraint is the arithmetic expression, called *projection*, which is obtained by a linear combination of the numerical attributes. There is an unbounded number of projections that we can use to form arbitrary conformance constraints. For example, for the projection AT, we can find a broad range $[\epsilon_3, \epsilon_4]$, such that all training tuples in Example 5.1 satisfy the conformance constraint $\epsilon_3 \leq AT \leq \epsilon_4$. However, this constraint is too inclusive and a learned model is unlikely to exploit such a weak constraint. In contrast, the projection $AT - DT - DUR$ leads to a stronger conformance constraint with a narrow range as its bounds, which is selectively permissible, and thus more effective.

Challenges and solution sketch. The principal challenge is to discover an *effective* set of conformance constraints that are likely to affect a model's inference implicitly. We first characterize "good" projections (that construct effective constraints) and then propose a method to discover them. We establish through theoretical analysis two important results: (1) A projection is good over a dataset if it is almost constant (i.e., has low variance) for all tuples in that dataset. (2) A set of projections, collectively, is good if the projections have small pair-wise correlations. We show that low variance components of a principal component analysis (PCA) on a dataset yield such a set of projections. Note that this is different from—and in fact completely opposite to—the traditional approaches (e.g., [263]) that perform multidimensional analysis based on the high-variance principal components, after reducing dimensionality using PCA.

Scope. Figure 5.2 summarizes prior work on related problems, but the scope of our setting differs significantly. Specifically, we can detect if a serving tuple is non-conforming with respect to the training dataset *only based on its predictor attributes*, and require no knowledge of the ground truth. This setting is essential in many practical applications when we observe *extreme verification latency* [69], where ground truths for serving tuples are not immediately available. For example, consider a self-driving car that is using a trained controller to generate actions based on readings of velocity, relative positions of obstacles, and their velocities. In this case, we need to determine, only based on the sensor readings (predictors), when the driver should be alerted to take over vehicle control.

Furthermore, we *do not assume access to the model*, i.e., model's predictions on a given tuple. This setting is also necessary for (1) safety-critical applications, where the goal is to alert the user as early as possible, without waiting for the availability of the prediction, (2) auditing and privacy-preserving applications where the prediction cannot be shared, and (3) when we are unaware of the detailed functionality of the system due to privacy concerns or lack of jurisdiction, but only have some meta-information such as the system trains some linear model over the training data.

Legend		constraints						violation		setting			technique				TML	
HP: Hyper Parameter FD: Functional Dependency DC: Denial Constraint ⊘: Does not require ⊥: Not applicable · Supports via extension !: Partially		parametric	arithmetic	approximate	conditional	notion of weight	interpretable	continuous	tuple-wise	noisy data	numerical attr.	categorial attr.	⊘ thresholds	⊘ distance metric	⊘ HP tuning	scalable	task agnostic	⊘ model access
Data Profiling	**Conformance Constraints**	●	●	●	●	●	⋆	●	●	●	●	●	●	●	●	●	●	●
	FD [251]						●					●	●	●	●		not addressed in prior work	
	Approximate FD [190]			●			●			●		●		●	●			
	Metric FD [187]			●			●			●	●	●			⊥	⊥		
	Conditional FD [90]	!			●		●		●	●		●		●	●			
	Pattern FD [265]	!					●		●	●		●		●	●			
	Soft FD [155]			●		●	●	●		●	●	●		●	●	●		
	Relaxed FD [53]			●			●			●	●	●			●			
	FDX [349]						●			●		●				●		
	Differential Dependency [299]						●				●	●			●			
	DC [42, 61]	!		●	●	●	●		●	●	●	●		●	●			
	Approximate DC [215, 258]	!		●	●	●	●		●	●	●	●		●	●			
	Statistical Constraint [337]			●			●	●		●	●	●		●	●	●		
Learning	Ordinary Least Square	●	●	●			⋆	●	●	●	●	⋆	●	●	●	●		
	Total Least Square	●	●	●			⋆	●	●	●	●	⋆	●	●	●	●	●	●
	Auto-encoder [71]	⊥						●	●	●	●	●				●	●	●
	Schelter et al. [289]+	⊥						●			●	●		●	●	●		
	Jiang et al. [164]	⊥						●	●	●	●	●	●			●		
	Hendrycks et al. [138]	⊥							●	●	●	●	●	●	●	●		
	Model's Prediction Probability	⊥						●	●	varies								

+ Requires additional information

Figure 5.2: Conformance constraints complement existing data profiling primitives and provide a mechanism to quantify trust in prediction, with minimal assumption on the setting. We provide an efficient and scalable technique to discover conformance constraints.

We focus on identifying *tuple-level* non-conformance as opposed to dataset-level non-conformance that usually requires observing entire data's distribution. However, our tuple-level approach trivially extends (by aggregation) to the entire dataset.

Contrast with prior art. We now discuss where conformance constraints fit with respect to the existing literature on data profiling and literature on modeling trust in data-driven inferences (Figure 5.2).

Data profiling techniques. Conformance constraints fall under the umbrella of data profiling, which refers to the task of extracting technical metadata about a given dataset [1]. A key task in data profiling is to learn relationships among attributes. Functional dependencies (FD) [251] and their variants only capture if a relationship exists between two sets of attributes, but do not provide a closed-form (parametric) expression of the relationship.

Using the FD $\{AT, DT\} \rightarrow \{DUR\}$ to model the constraint of Example 5.1 suffers from several limitations. First, since the data is noisy, in the traditional setting, no FD would be learned. Metric FDs [187] allow small variations in the data, but the users are required to supply appropriate distance metrics and thresholds. For example, if the attribute $time$ is split across two separate attributes ($hour$ and $minute$), the distance metric is nontrivial and involves multiple attributes: it needs to encode that $\langle hour = 4, min = 59\rangle$ and $\langle hour = 5, min = 1\rangle$ are similar, while $\langle hour = 4, min = 1\rangle$ and $\langle hour = 5, min = 59\rangle$ are not. In contrast, projections of conformance constraints can model the composite attribute: $60 \cdot hour + minute$. We also automatically discover the coefficients (60 and 1) for such a composite attribute.

Denial constraints (DC) [42, 61, 215, 258] encapsulate a number of different data-profiling primitives such as FDs and their variants (e.g, [90]). Exact DCs can adjust to noisy data by adding predicates until the constraint becomes exact over the entire dataset, but this can make the constraint extremely large and complex, which might even fail to provide the desired generalization. For example, a finite DC—whose language is limited to universally quantified first-order logic—cannot model the constraint of Example 5.1, which involves an arithmetic expression (addition and multiplication with a constant). Expressing conformance constraints requires a richer language that includes linear arithmetic expressions. Pattern functional dependencies [265] move towards addressing this limitation of DCs using regular expressions, but they focus on text attributes. Pattern functional dependencies (PFD) [265] move towards addressing this limitation of DCs, but they focus on text attributes: they are regex-based and treat digits as characters. However, modeling arithmetic relationships of numerical attributes requires interpreting digits as numbers.

To adjust for noise, FDs and DCs either relax the notion of constraint violation or allow a user-defined fraction of tuples to violate the (strict) constraint [53, 150, 155, 187, 190, 215, 258]. Some approaches [155, 337, 349] use statistical techniques to model other types of data profiles such as correlations and conditional dependencies. However, they

require additional parameters such as noise and violation thresholds and distance metrics. In contrast, conformance constraints do not require any parameter from the user and work on noisy datasets.

Existing data profiling techniques are not designed to learn what ML models exploit and they are sensitive to noise in the numerical attributes. Moreover, data constraint discovery algorithms typically search over an exponential set of candidates, and hence, are not scalable: their complexity grows exponentially with the number of attributes or quadratically with data size. In contrast, our technique for deriving conformance constraints is highly scalable (linear in data size) and efficient (cubic in the number of attributes). It does not explicitly explore the candidate space, as PCA—which lies at the core of our technique—performs the search *implicitly* by iteratively refining weaker constraints to stronger ones.

Learning techniques. While *ordinary least square (OLS)*—commonly known as linear regression—finds the lowest-variance projection, it minimizes observational error on only the target attribute, and thus does not apply to our setting. *Total least square (TLS)*—also known as orthogonal regression—offers a partial solution to our problem as it takes observational errors on all predictor attributes into account. However, TLS finds only one projection—the one with the lowest variance—that fits the data tuples best. But there may exist other projections with slightly higher variances and conformance constraints consider them all. As we show empirically in Section 5.5.3, constraints derived from multiple projections, collectively, capture various aspects of the data, and result in an effective data profile targeted towards certain tasks such as data-drift quantification.

Few work [71, 138] use autoencoder's [141, 279] input reconstruction error to determine if a new data point is out-of-distribution. Another mechanism [217] learns data *assertions* via autoencoders towards effective detection of invalid serving inputs. However, such an approach is task-specific and needs a specific system (e.g., a deep neural network) to begin with. We also share similarity with one-class-classification [309], where the training data contains tuples from only one class. However, conformance constraints differ from

these approaches as it performs under the additional requirement to generalize the data in a way that is exploited by a given class of ML models. In general, there is a clear gap between representation learning (that models data likelihood) [3, 141, 177, 279] and the (constraint-oriented) data-profiling techniques to address the problem of trusted AI. Our aim is to bridge this gap by introducing conformance constraints that are more abstract, yet informative, descriptions of data, tailored towards characterizing trust in ML predictions.

While some recent techniques [71, 138, 164, 289] aim at validating the inferences made by machine-learned models on unseen tuples, they usually require knowledge of the inference task and/or access to the model, which we do not. Furthermore, these techniques usually require costly hyper-parameter tuning and do not generate closed-form data profiles like conformance constraints (Figure 5.2).

Contributions. We make the following contributions:

- We ground the motivation of our work with two case studies on trusted machine learning (TML) and data drift. (Section 5.1)
- We introduce and formalize conformance constraints, a new data profiling primitive that specify constraints over arithmetic relationships among numerical attributes of a dataset. We describe a *conformance language* to express conformance constraints, and a *quantitative semantics* to quantify how much a tuple violates the conformance constraints. In applications of constraint violations, some violations may be more or less critical than others. To capture that, we consider a notion of constraint importance, and weigh violations against constraints accordingly. (Section 5.2)
- We formally establish that strong conformance constraints are constructed from projections with small variance and small mutual correlation on the given dataset. Beyond simple linear constraints (e.g., the one in Example 5.1), we derive *disjunctive* constraints, which are disjunctions of linear constraints. We achieve this by dividing

the dataset into disjoint partitions, and learning linear constraints for each partition. We provide an efficient, scalable, and highly parallelizable algorithm for computing a set of linear conformance constraints and disjunctions over them. We also analyze its runtime and memory complexity. (Section 5.3)

- We formalize the notion of *unsafe* tuples in the context of trusted machine learning and provide a mechanism to detect unsafe tuples using conformance constraints. (Section 5.4)

- We empirically analyze the effectiveness of conformance constraints in our two case-study applications—TML and data-drift quantification. We show that conformance constraints can reliably predict the trustworthiness of linear models and quantify data drift precisely, outperforming the state of the art. (Section 5.5)

- We demonstrate two tools CoCo and ExTuNe that are built on conformance constraints and can help in data understanding and data cleaning, and explaining causes of tuple non-conformance, respectively.

5.1 Case Studies

Like other data-profiling primitives, conformance constraints have general applicability in understanding and describing datasets. But their true power lies in quantifying the degree of a tuple's non-conformance with respect to a reference dataset. Within the scope of this work, we focus on two case studies to motivate our work. We provide an extensive evaluation over these applications in Section 5.5.

Trusted machine learning (TML) refers to the problem of quantifying trust in the inference made by a machine-learned model on a new serving tuple [164, 276, 286, 311, 319]. When a model is trained using a dataset, the conformance constraints for that dataset specify a safety envelope [311] that characterizes the tuples for which the model is expected

to make trustworthy predictions. If a serving tuple falls outside the safety envelope (violates the conformance constraints), then the model is likely to produce an untrustworthy inference. Intuitively, the higher the violation, the lower the trust. Some classifiers produce a confidence measure along with the class prediction, typically by applying a softmax function to the raw numeric prediction values. However, such a confidence measure is not well-calibrated [129, 164], and therefore, cannot be reliably used as a measure of trust in the prediction. Additionally, a similar mechanism is not available for inferences made by regression models.

In the context of TML, we formalize the notion of *unsafe tuples*, on which the prediction may be untrustworthy. We establish that conformance constraints provide a sound and complete procedure for detecting unsafe tuples, which indicates that the search for conformance constraints should be guided by the class of models considered by the corresponding learning system (Section 5.4).

Data drift [28, 115, 194, 263] specifies a significant change in a dataset with respect to a reference dataset, which typically requires that systems be updated and models retrained. To quantify how much a dataset D' drifted from a reference dataset D, our three-step approach is: (1) compute conformance constraints for D, (2) evaluate the constraints on all tuples in D' and compute their violations (degrees of non-conformance), and (3) finally, aggregate the tuple-level violations to get a dataset-level violation. If all tuples in D' satisfy the constraints, then we have no evidence of drift. Otherwise, the aggregated violation serves as the drift quantity.

5.2 Conformance Constraints

In this section, we first provide the general definition of conformance constraints. Then we propose a language for representing them. Finally, we define quantitative semantics over conformance constraints, which allows us to quantify their violation.

Basic notations. Let $\mathcal{R}(A_1, A_2, \ldots, A_m)$ denote a relation schema where A_i denotes the i^{th} attribute of $\mathcal{R}$. We use $\texttt{Dom}_i$ to denote the domain of attribute A_i. Then the set $\mathbf{Dom}^m = \texttt{Dom}_1 \times \cdots \times \texttt{Dom}_m$ specifies the domain of all possible tuples. We use $t \in \mathbf{Dom}^m$ to denote a tuple in the schema $\mathcal{R}$. A dataset $D \subseteq \mathbf{Dom}^m$ is a specific instance of the schema $\mathcal{R}$. For ease of notation, we assume some order of tuples in D and we use $t_i \in D$ to refer to the i^{th} tuple and $t_i.A_j \in \texttt{Dom}_j$ to denote the value of the j^{th} attribute of t_i.

5.2.1 Conformance Constraint

A conformance constraint Φ characterizes a set of allowable or conforming tuples and is expressed through a *conformance language* (Section 5.2.2). We write $\Phi(t)$ and $\neg\Phi(t)$ to denote that t satisfies and violates Φ, respectively.

Definition 5.1 (Conformance constraint)**.** *A conformance constraint for a dataset $D \subseteq \mathbf{Dom}^m$ is a formula $\Phi : \mathbf{Dom}^m \mapsto \{\texttt{True}, \texttt{False}\}$ such that $|\{t \in D \mid \neg\Phi(t)\}| \ll |D|$.*

The set $\{t \in D \mid \neg\Phi(t)\}$ denotes atypical tuples in D that do not satisfy the conformance constraint Φ. In our work, we do not need to know the set of atypical tuples, nor do we need to purge the atypical tuples from the dataset. Our techniques derive constraints in ways that ensure there are very few atypical tuples (Section 5.3).

5.2.2 Conformance Language

Projection. A central concept in our conformance language is *projection*. Intuitively, a projection is a derived attribute that specifies a "lens" through which we look at the tuples. More formally, a projection is a function $F : \mathbf{Dom}^m \mapsto \mathbb{R}$ that maps a tuple $t \in \mathbf{Dom}^m$ to a real number $F(t) \in \mathbb{R}$. In our language for conformance constraints, we only consider projections that correspond to linear combinations of the numerical attributes of a dataset. Specifically, to define a projection, we need a set of numerical coefficients for all attributes of the dataset and the projection is defined as a sum over the attributes, weighted by their corresponding coefficients. We extend a projection

F to a dataset D by defining $F(D)$ to be the sequence of reals obtained by applying F on each tuple in D individually.

Grammar. Our language for conformance constraints consists of formulas Φ generated by the following grammar:

$$\begin{aligned}
\phi &:= \texttt{lb} \leq F(\vec{A}) \leq \texttt{ub} \mid \wedge(\phi, \ldots, \phi) \\
\psi_A &:= \vee(\ (A = c_1) \rhd \phi,\ (A = c_2) \rhd \phi,\ \ldots) \\
\Psi &:= \psi_A \mid \wedge(\psi_{A_1}, \psi_{A_2}, \ldots) \\
\Phi &:= \phi \mid \Psi
\end{aligned}$$

The language consists of (1) bounded constraints $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$ where F is a projection on $\mathbf{Dom}^m$, $\vec{A}$ is the tuple of formal parameters $(A_1, A_2, \ldots, A_m)$, and $\texttt{lb}, \texttt{ub} \in \mathbb{R}$ are reals; (2) equality constraints $A = c$ where A is an attribute and c is a constant in A's domain; and (3) operators ($\rhd$, $\wedge$, and $\vee$,) that connect the constraints. Intuitively, $\rhd$ is a switch operator that specifies which constraint ϕ applies based on the value of the attribute A, $\wedge$ denotes conjunction, and $\vee$ denotes disjunction. Formulas generated by ϕ and Ψ are called *simple constraints* and *compound constraints*, respectively. Note that a formula generated by ψ_A only allows equality constraints on a single attribute, namely A, among all the disjuncts.

Example 5.2. *Consider the dataset D consisting of the first four tuples $\{t_1, t_2, t_3, t_4\}$ of Figure 5.1. A simple constraint for D is:*

$$\phi_1 : -5 \leq AT - DT - DUR \leq 5.$$

Here, the projection $F(\vec{A})$=$AT-DT-DUR$, with attribute coefficients $\langle 1, -1, -1\rangle$, $\texttt{lb}$=-5, *and* $\texttt{ub}$=5. *A compound constraint is:*

$$\psi_2 : M = \textit{“May”} \rhd -2 \leq AT - DT - DUR \leq 0$$
$$\vee \;\; M = \textit{“June”} \rhd \;\; 0 \leq AT - DT - DUR \leq 5$$
$$\vee \;\; M = \textit{“July”} \rhd -5 \leq AT - DT - DUR \leq 0$$

For ease of exposition, we assume that all times are converted to minutes (e.g., `06:10` $= 6{\times}60{+}10 = 370$*) and* M *denotes the departure month, extracted from* $Departure\ Date$*.*

5.2.3 Quantitative Semantics

Conformance constraints have a natural Boolean semantics: a tuple either satisfies a constraint or it does not. However, Boolean semantics is of limited use in practice, because it does not quantify the degree of constraint violation. We interpret conformance constraints using a quantitative semantics, which quantifies violations, and reacts to noise more gracefully than Boolean semantics.

The quantitative semantics $[\![\Phi]\!](t)$ is a measure of the violation of Φ on a tuple t—with a value of 0 indicating no violation and a value greater than 0 indicating some violation. In Boolean semantics, if $\Phi(t)$ is `True`, then $[\![\Phi]\!](t)$ will be 0; and if $\Phi(t)$ is `False`, then $[\![\Phi]\!](t)$ will be 1. Formally, $[\![\Phi]\!]$ is a mapping from $\mathbf{Dom}^m$ to $[0, 1]$.

Quantitative semantics of simple constraints. We build upon ϵ-insensitive loss [317] to define the quantitative semantics of simple constraints, where the bounds `lb` and `ub` define the ϵ-insensitive zone:

$$[\![\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}]\!](t) := \eta(\alpha \cdot \max(0, F(t) - \texttt{ub}, \texttt{lb} - F(t)))$$
$$[\![\wedge(\phi_1, \ldots, \phi_K)]\!](t) := \textstyle\sum_k^K \gamma_k \cdot [\![\phi_k]\!](t)$$

The quantitative semantics uses the following parameters:

Scaling factor $\alpha \in \mathbb{R}^+$. Projections are unconstrained functions and different projections can map the same tuple to vastly different values. We use a scaling factor α to standardize the values computed by a projection F, and to bring the values of different projections to the same comparable scale. The scaling factor is automatically computed as the inverse of the standard deviation: $\frac{1}{\sigma(F(D))}$. We set α to a large positive number when $\sigma(F(D)) = 0$.

Normalization function $\eta(.) : \mathbb{R} \mapsto [0, 1]$. The normalization function maps values in the range $[0, \infty)$ to the range $[0, 1)$. While any monotone mapping from $\mathbb{R}^{\geq 0}$ to $[0, 1)$ can be used, we pick $\eta(z) = 1 - e^{-z}$.

Importance factors $\gamma_k \in \mathbb{R}^+$, $\sum_k^K \gamma_k = 1$. The weights γ_k control the contribution of each bounded-projection constraint in a conjunctive formula. This allows for prioritizing constraints that are more significant than others. In our work, we derive the importance factor of a constraint automatically, based on its projection's standard deviation over D.

Our technique for deriving (unnormalized) importance factor γ_k, for bounded-projection constraint on projection F_k, uses the mapping $\frac{1}{\log(2+\sigma(F_k(D)))}$. This mapping correctly translates our principles for quantifying violation by putting high weight on conformance constraints constructed from low-variance projections, and low weight on conformance constraints constructed from high-variance projections. While this mapping works extremely well across a large set of applications (including the ones shown in our experimental results), our quantitative semantics are not limited to any specific mapping. In fact, the function to compute importance factors for bounded-projections can be user-defined (but we do not require it from the user). Specifically, a user can plug in any custom function to derive the (unnormalized) importance factors. Furthermore, our technique to compute the bounds `lb` and `ub` can also be customized (but we do not require it from the user either). Depending on the application requirements, one can apply techniques used in machine learning literature (e.g., cross-validation) to tighten or loosen the conformance constraints by tuning

these parameters. However, we found our technique—even without any cross-validation—for deriving these parameters to be very effective in most practical applications.

5.2.3.1 Quantitative Semantics of Compound Constraints

Compound constraints are first simplified into simple constraints, and they get their meaning from the simplified form. We define a function $\texttt{simp}(\psi, t)$ that takes a compound constraint ψ and a tuple t and returns a simple constraint:

$$\texttt{simp}(\vee(\ (A = c_1) \rhd \phi_1, (A = c_2) \rhd \phi_2, \ldots), t) := \phi_k \text{ if } t.A = c_k$$

$$\texttt{simp}(\wedge(\psi_{A_1}, \psi_{A_2}, \ldots), t) := \wedge(\texttt{simp}(\psi_{A_1}, t), \texttt{simp}(\psi_{A_2}, t), \ldots)$$

If the condition in the definition above does not hold for any c_k, then $\texttt{simp}(\psi, t)$ is undefined and $\texttt{simp}(\wedge(\ldots, \psi, \ldots), t)$ is also undefined. If $\texttt{simp}(\psi, t)$ is undefined, then $[\![\psi]\!](t) := 1$. When $\texttt{simp}(\psi, t)$ is defined, the quantitative semantics of ψ is given by:

$$[\![\psi]\!](t) := [\![\texttt{simp}(\psi, t)]\!](t)$$

Since compound constraints simplify to simple constraints, we mostly focus on simple constraints. Even there, we pay special attention to bounded-projection constraints (ϕ) of the form $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$, which lie at the core of simple constraints.

Example 5.3. *Consider the constraint ϕ_1 from Example 5.2. For $t \in D$, $[\![\phi_1]\!](t) = 0$ since ϕ_1 is satisfied by all tuples in D. The standard deviation of the projection F over D, $\sigma(F(D))$=$\sigma(\{0, -5, 5, -2\})$=3.6. Now consider the last tuple $t_5 \notin D$. $F(t_5) = (370 - 1350) - 458 = -1438$, which is way below the lower bound -5 of ϕ_1. Now we compute how much t_5 violates ϕ_1: $[\![\phi_1]\!](t_5) = [\![-5 \leq F(\vec{A}) \leq 5]\!](t_5) = \eta(\alpha \cdot \max(0, -1438 - 5, -5 + 1438)) = 1 - e^{-\frac{1433}{3.6}} \approx 1$. Intuitively, this implies that t_5 strongly violates ϕ_1.*

5.3 Conformance Constraint Synthesis

In this section, we describe our techniques for deriving conformance constraints. We start with the synthesis of simple constraints (the ϕ constraints in our language specification), followed by compound constraints (the Ψ constraints in our language specification). Finally, we analyze the time and memory complexity of our algorithm.

5.3.1 Simple Conformance Constraints

Synthesizing simple conformance constraints involves (a) discovering the projections, and (b) discovering the lower and upper bounds for each projection. We start by discussing the latter task (b). We then describe a principle for identifying effective projections, based on which we solve task (a).

5.3.1.1 Synthesizing Bounds for Projections

Fix a projection F and consider the bounded-projection constraint ϕ: $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$. Given a dataset D, a trivial choice for the bounds is: $\texttt{lb} = \min(F(D))$ and $\texttt{ub} = \max(F(D))$. However, this choice is very sensitive to noise: adding a single atypical tuple to D can produce very different constraints. Instead, we use a more robust choice as follows:

$$\texttt{lb} = \mu(F(D)) - C \cdot \sigma(F(D)), \quad \texttt{ub} = \mu(F(D)) + C \cdot \sigma(F(D))$$

Here, $\mu(F(D))$ and $\sigma(F(D))$ denote the mean and standard deviation of the values in $F(D)$, respectively, and C is some positive constant. With these bounds, $[\![\phi]\!](t) = 0$ implies that $F(t)$ is within $C \times \sigma(F(D))$ from the mean $\mu(F(D))$. In our experiments, we set $C = 4$, which ensures that in expectation, very few tuples in D will violate the constraint for many distributions of the values in $F(D)$. Specifically, if $F(D)$ follows a normal distribution, 99.99% of the population is expected to lie within 4 standard deviations from mean. Note that we make no assumption on the original data distribution of each attribute.

Setting the bounds lb and ub as $C \cdot \sigma(F(D))$-away from the mean, and the scaling factor α as $\frac{1}{\sigma(F(D))}$, guarantees the following property for our quantitative semantics:

Lemma 5.1. *Let D be a dataset and let ϕ_k be $\texttt{lb}_k \leq F_k(\vec{A}) \leq \texttt{ub}_k$ for $k = 1, 2$. Then, for any tuple t, if $\frac{|F_1(t)-\mu(F_1(D))|}{\sigma(F_1(D))} \geq \frac{|F_2(t)-\mu(F_2(D))|}{\sigma(F_2(D))}$, then $[\![\phi_1]\!](t) \geq [\![\phi_2]\!](t)$.*

This means that larger deviation from the mean (proportionally to the standard deviation) results in higher degree of violation under our semantics. The proof follows from the fact that the normalization function $\eta(.)$ is monotonically increasing, and hence, $[\![\phi_k]\!](t)$ is a monotonically non-decreasing function of $\frac{|F_k(t)-\mu(F_k(D))|}{\sigma(F_k(D))}$.

5.3.1.2 Principle for Synthesizing Projections

To understand how to derive the projections for effective conformance constraints, we need to first understand what makes a constraint more effective than others. Primarily, an effective constraint (1) should not overfit the data, but rather generalize by capturing the properties of the data, and (2) should not underfit the data, because it would be too permissive and fail to identify deviations effectively. Our flexible bounds (Section 5.3.1.1) serve to avoid overfitting. In this section, we focus on identifying the principles that help us avoid underfitting. We first describe the key technical ideas for characterizing effective projections through example and then proceed to formalization.

Example 5.4. *Let D be a dataset of three tuples {(1,1.1),(2,1.7),(3,3.2)} with two attributes X and Y. Consider two arbitrary projections: X and Y. For the projection X: $\mu(X(D)) = 2$ and $\sigma(X(D)) = 0.8$. So bounds for its conformance constraint are: $\texttt{lb} = 2 - 4 \times 0.8 = -1.2$ and $\texttt{ub} = 2 + 4 \times 0.8 = 5.2$. This gives us the conformance constraint: $-1.2 \leq X \leq 5.2$. Similarly, for projection Y, we get the conformance constraint: $-1.6 \leq Y \leq 5.6$. Figure 5.3(a) shows the conformance zone (clear region) defined by these two conformance constraints. The shaded region depicts non-conformance zone. The conformance zone is large and too permissive: it allows many atypical tuples with respect to D, such as $(0, 4)$ and $(4, 0)$.*

A natural question arises: are there other projections that can better characterize conformance with respect to the tuples in D? The answer is yes and next we show another pair of projections that shrink the conformance zone significantly.

Example 5.5. *In Figure 5.3(b), the clear region is defined by the conformance constraints* $-0.8 \leq X - Y \leq 0.8$ *and* $-2.8 \leq X + Y \leq 10.8$*, over projections* $X - Y$ *and* $X + Y$*, respectively. The clear region here is indeed much smaller than the one in Figure 5.3(a) and contains much fewer atypical tuples.*

How can we derive projection $X - Y$ from the projections X and Y, given D? Note that X and Y are highly correlated in D. In Lemma 5.2, we show that two highly correlated projections can be linearly combined to construct another projection with lower standard deviation that generates a *stronger* constraint. We proceed to formalize stronger constraint—which defines whether a constraint is more effective than another in quantifying violation—and *incongruous* tuples—which help us estimate the subset of the data domain for which a constraint is stronger than the others.

Definition 5.2 (Stronger constraint)**.** *A conformance constraint* ϕ_1 *is stronger than another conformance constraint* ϕ_2 *on a subset* $H \subseteq \mathbf{Dom}^m$ *if* $\forall t \in H,\ [\![\phi_1]\!](t) \geq [\![\phi_2]\!](t)$.

Given a dataset $D \subseteq \mathbf{Dom}^m$ and a projection F, for any tuple t, let $\Delta F(t) = F(t) - \mu(F(D))$. For projections F_1 and F_2, the correlation coefficient ρ_{F_1,F_2} (over D) is defined as $\frac{\frac{1}{|D|}\sum_{t\in D} \Delta F_1(t)\Delta F_2(t)}{\sigma(F_1(D))\sigma(F_2(D))}$.

Definition 5.3 (Incongruous tuple)**.** *A tuple* t *is* incongruous *w.r.t. a projection pair* $\langle F_1, F_2\rangle$ *on* D *if:* $\Delta F_1(t) \cdot \Delta F_2(t) \cdot \rho_{F_1,F_2} < 0$.

Informally, an incongruous tuple for a pair of projections does not follow the general trend of correlation between the projection pair. For example, if F_1 and F_2 are positively correlated ($\rho_{F_1,F_2} > 0$), an incongruous tuple t deviates in opposite ways from the mean of each projection ($\Delta F_1(t) \cdot \Delta F_2(t) < 0$). Our goal is to find projections that yield a conformance zone with very few incongruous tuples.

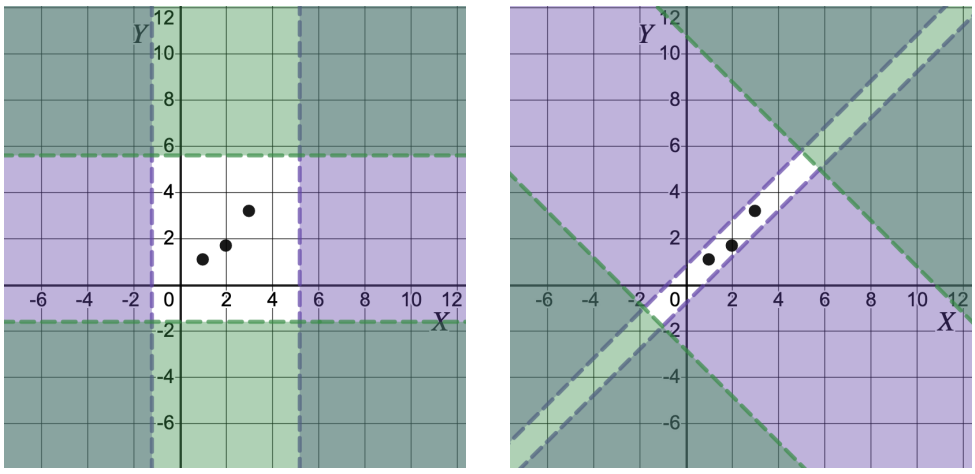


Figure 5.3: Clear and shaded regions depict conformance and non-conformance zones, respectively. (a) Correlated projections X and Y yield conformance constraints forming a large conformance zone, (b) Uncorrelated (orthogonal) projections $X - Y$ and $X + Y$ yield conformance constraints forming a smaller conformance zone.

Example 5.6. *In Example 5.4, X and Y are positively correlated with $\rho_{X,Y} \approx 1$. The tuple $t = (0, 4)$ is incongruous w.r.t. $\langle X, Y \rangle$, because $X(t) = 0 < \mu(X(D)) = 2$, whereas $Y(t) = 4 > \mu(Y(D)) = 2$. Intuitively, the incongruous tuples do not behave like the tuples in D when viewed through the projections X and Y. Note that the narrow conformance zone of Figure 5.3(b) no longer contains the incongruous tuple $(0, 4)$. In fact, the conformance zone defined by the conformance constraints derived from projections $X - Y$ and $X + Y$ are free from a vast majority of the incongruous tuples.*

We proceed to state Lemma 5.2, which informally says that: any two highly correlated projections can be linearly combined to construct a new projection to obtain a stronger constraint. We write ϕ_F to denote the conformance constraint $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$, synthesized from F (proof is in Appendix B.1).

Lemma 5.2. *Let D be a dataset and F_1, F_2 be two projections on D s.t. $|\rho_{F_1,F_2}| \geq \frac{1}{2}$. Then, $\exists \beta_1, \beta_2 \in \mathbb{R}$ s.t. $\beta_1^2 + \beta_2^2 = 1$ and for the new projection $F = \beta_1 F_1 + \beta_2 F_2$:*

(1) $\sigma(F(D)) < \sigma(F_1(D))$ and $\sigma(F(D)) < \sigma(F_2(D))$, and

(2) ϕ_F *is stronger than both* ϕ_{F_1} *and* ϕ_{F_2} *on the set of tuples that are incongruous w.r.t.* $\langle F_1, F_2 \rangle$.

We now extend the result to multiple projections in Theorem 5.1 (proof is in Appendix B.2).

Theorem 5.1 (Low Standard Deviation Constraints)**.** *Given a dataset* D*, let* $\mathcal{F}=\{F_1, \ldots, F_K\}$ *denote a set of projections on* D *s.t.* $\exists F_i, F_j \in \mathcal{F}$ *with* $|\rho_{F_i,F_j}| \geq \frac{1}{2}$*. Then, there exist a nonempty subset* $I \subseteq \{1, \ldots, K\}$ *and a projection* $F = \sum_{k \in I} \beta_k F_k$*, where* $\beta_k \in \mathbb{R}$ *s.t.*

(1) $\forall k \in I$, $\sigma(F(D)) < \sigma(F_k(D))$,

(2) $\forall k \in I$, ϕ_F *is stronger than* ϕ_{F_k} *on the subset* H*, where*
$H=\{t \mid \forall k \in I(\beta_k \Delta F_k(t) \geq 0) \vee \forall k \in I(\beta_k \Delta F_k(t) \leq 0)\}$*, and*

(3) $\forall k \notin I$, $|\rho_{F,F_k}| < \frac{1}{2}$.

The theorem establishes that to detect violations for tuples in H: (1) projections with low standard deviations define stronger constraints (and, thus, are preferable), and (2) a set of constraints with highly correlated projections is suboptimal (as they can be linearly combined to generate stronger constraints). Note that H is a conservative estimate for the set of tuples where ϕ_F is stronger than each ϕ_{F_k}; there exist tuples outside H for which ϕ_F is stronger.

5.3.1.3 PCA-inspired Projection Derivation

Theorem 5.1 sets the requirements for good projections (see also [194, 225, 315] that make similar observations in different ways). It indicates that we can start with any arbitrary projections and then iteratively improve them. However, we can get the desired set of best projections in one shot using an algorithm inspired by principal component analysis (PCA). PCA relies on computing eigenvectors. There exist different algorithms for computing eigenvectors (from the infinite space of possible vectors). The general mechanism

involves applying numerical approaches to iteratively converge to the eigenvectors (up to a desired precision) as no analytical solution exists in general. Our algorithm returns projections that correspond to the principal components of a slightly modified version of the given dataset. Algorithm 2 details our approach for discovering projections for constructing conformance constraints:

Line 1 Drop all non-numerical attributes from D to get the numeric dataset D_N. This is necessary because PCA only applies to numerical values. Instead of dropping, one can also consider embedding techniques to convert non-numerical attributes to numerical ones.

Line 2 Add a new column to D_N that consists of the constant 1, to obtain the modified dataset $D'_N := [\vec{1}; D_N]$, where $\vec{1}$ denotes the column vector with 1 everywhere. We do this transformation to capture the additive constant within principal components, which ensures that the approach works even for unnormalized data.

Line 3 Compute K eigenvectors of the square matrix ${D'_N}^T D'_N$, where K denotes the number of columns in D'_N. These eigenvectors provide coefficients to construct projections.

Lines 5–6 Remove the first element (coefficient for the newly added constant column) of all eigenvectors and normalize them to generate projections. Note that we no longer need the constant element of the eigenvectors since we can appropriately adjust the bounds, `lb` and `ub`, for each projection by evaluating it on D_N.

Line 7 Compute importance factor for each projection. Since projections with smaller standard deviations are more discerning (stronger), as discussed in Section 5.2.3, we assign each projection an importance factor (γ) that is inversely proportional to its standard deviation over D_N.

Line 8 Return the projections with corresponding normalized importance factors.

Algorithm 2: Procedure to generate linear projections.

Inputs : A dataset $D \subset \mathbf{Dom}^m$

Output : A set $\{(F_1, \gamma_1), \ldots, (F_K, \gamma_K)\}$ of projections and importance factors

$D_N \leftarrow D$ after dropping non-numerical attributes

$D'_N \leftarrow [\vec{1}; D_N]$

$\{\vec{w}_1, \ldots, \vec{w}_K\} \leftarrow$ eigenvectors of ${D'_N}^T D'_N$

foreach $1 \leq k \leq K$ **do**

  $\vec{w}'_k \leftarrow \vec{w}_k$ with first element removed

  $F_k \leftarrow \lambda \vec{A} : \frac{\vec{A}^T \vec{w}'_k}{||\vec{w}'_k||}$

  $\gamma_k \leftarrow \frac{1}{\log(2+\sigma(F_k(D_N)))}$

return $\{(F_1, \frac{\gamma_1}{Z}), \ldots, (F_K, \frac{\gamma_K}{Z})\}$, where $Z = \sum_k \gamma_k$

We now claim in Theorem 5.2 that the projections returned by Algorithm 2 include the projection with minimum standard deviation and the correlation between any two projections is 0 (proof is in Appendix B.3). This indicates that we cannot further improve the projections: they are optimal.

Theorem 5.2 (Correctness of Algorithm 2)**.** *Given a numerical dataset D, let $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ be the set of linear projections returned by Algorithm 2. Let $\sigma^* = \min_k^K \sigma(F_k(D))$. Then,*

(1) $\sigma^* \leq \sigma(F(D))$ for every possible linear projection F, and

(2) $\forall F_j, F_k \in \mathcal{F}$ s.t. $F_j \neq F_k$, $\rho_{F_j, F_k} = 0$.

Using projections $F_1, \ldots, F_K$, and importance factors $\gamma_1, \ldots, \gamma_K$, returned by Algorithm 2, we generate the simple (conjunctive) constraint with K conjuncts: $\bigwedge_k \texttt{lb}_k \leq F_k(\vec{A}) \leq \texttt{ub}_k$. We compute the bounds $\texttt{lb}_k$ and $\texttt{ub}_k$ following Section 5.3.1.1 and use the importance factor γ_k for the k^{th} conjunct in the quantitative semantics.

Example 5.7. *Algorithm 2 finds the projection of the conformance constraint of Example 5.1, but in a different form. The actual airlines dataset has an attribute $distance$ (DIS)*

that represents miles travelled by a flight. In our experiments, we found the following conformance constraint[1] *over the dataset of daytime flights:*

$$0.7 \times AT - 0.7 \times DT - 0.14 \times DUR - 0.07 \times DIS \approx 0 \tag{5.1}$$

This constraint is not quite interpretable by itself, but it is in fact a linear combination of two expected and interpretable constraints:

$$AT - DT - DUR \approx 0 \tag{5.2}$$

$$DUR - 0.12 \times DIS \approx 0 \tag{5.3}$$

Here, (5.2) is the one mentioned in Example 5.1 and (5.3) follows from the fact that average aircraft speed is about 500 *mph implying that it requires* 0.12 *minutes per mile.* $0.7 \times$ *(5.2)* + $0.56 \times$ *(5.3) yields:*

$$\begin{aligned} & 0.7 \times (AT - DT - DUR) + 0.56 \times DUR - 0.56 \times 0.12 \times DIS \approx 0 \\ \implies & 0.7 \times AT - 0.7 \times DT - 0.14 \times DUR - 0.07 \times DIS \approx 0 \end{aligned}$$

Which is exactly the conformance constraint (5.1). Algorithm 2 found the optimal projection of (5.1), which is a linear combination of the projections of (5.2) and (5.3). The reason is: there is a correlation between the projections of (5.2) and (5.3) over the dataset (Theorem 5.1). One possible explanation of this correlation is: whenever there is an error in the reported duration of a tuple, it violates both (5.2) and (5.3). Due to this natural correlation, Algorithm 2 returned the optimal projection of (5.1), that "covers" both projections of (5.2) or (5.3).

[1]For ease of exposition, we show conformance constraint of the form $\epsilon_1 \leq F(\vec{A}) \leq \epsilon_2$ in the form $F(\vec{A}) \approx 0$, where $|\epsilon_1| \approx 0$ and $|\epsilon_2| \approx 0$.

5.3.2 Compound Conformance Constraints

The quality of our PCA-based simple linear constraints relies on how many low variance linear projections we are able to find on the given dataset. For many datasets, it is possible we find very few, or even none, such linear projections. In these cases, it is fruitful to search for compound constraints; we first focus on *disjunctive constraints* (defined by ψ_A in our language grammar).

Example 5.8. *PCA-based approach fails in cases where there exist different piecewise linear trends within the data. If we apply PCA to learn conformance constraints on the entire dataset of Figure 5.4(a), it will learn two low-quality constraints, with very high variance. In contrast, partitioning the dataset into three partitions (Figure 5.4(b)), and then learning constraints separately on each partition, will result in significant improvement of the learned constraints.*

A disjunctive constraint is a compound constraint of the form $\bigvee_k((A = c_k) \rhd \phi_k)$, where each ϕ_k is a constraint for a specific partition of D. Finding disjunctive constraints involves horizontally partitioning the dataset D into smaller disjoint datasets $D_1, D_2, \ldots, D_L$. Our strategy for partitioning D is to use categorical attributes with a small domain in D; in our implementation, we use those attributes A_j for which $|\{t.A_j | t \in D\}| \leq 50$. If A_j is such an attribute with values $v_1, v_2, \ldots, v_L$, we partition D into L disjoint datasets $D_1, D_2, \ldots, D_L$, where $D_l = \{t \in D | t.A_j = v_l\}$. Let $\phi_1, \phi_2, \ldots, \phi_L$ be the L simple conformance constraints we learn for $D_1, D_2, \ldots, D_L$ using Algorithm 2, respectively. We compute the following disjunctive conformance constraint for D:

$$((A_j = v_1) \rhd \phi_1) \vee ((A_j = v_2) \rhd \phi_2) \vee \cdots \vee ((A_j = v_L) \rhd \phi_L)$$

We repeat this process and partition D across multiple attributes and generate a compound disjunctive constraint for each attribute. Finally, we generate a compound conjunctive conformance constraint (Ψ), which is the conjunction of all these compound disjunc-

tive constraints, as the final conformance constraint for D. Intuitively, the final compound conformance constraint forms a set of *overlapping* hyper-boxes around the data tuples.

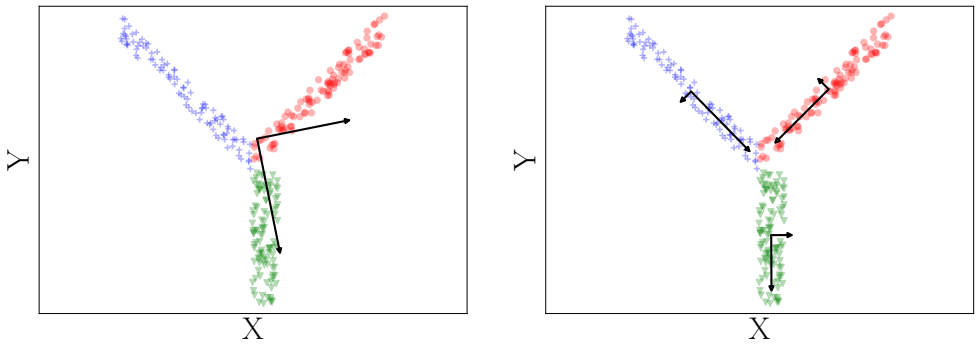
5.3.3 Complexity Analysis

Runtime Complexity. Computing simple constraints involves two computational steps: (1) computing X^TX, where X is an $n \times m$ matrix with n tuples (rows) and m attributes (columns), which takes $\mathcal{O}(nm^2)$ time, and (2) computing the eigenvalues and eigenvectors of an $m \times m$ positive definite matrix, which has complexity $\mathcal{O}(m^3)$ [249]. Once we obtain the linear projections using the above two steps, we need to compute the mean and variance of these projections on the original dataset, which takes $\mathcal{O}(nm^2)$ time. In summary, the overall procedure is cubic in the number of attributes and linear in the number of tuples. For computing disjunctive constraints, we greedily pick attributes that take at most L (typically small) distinct values, and then run the above procedure for simple constraints at most L times. This adds just a constant factor overhead per attribute.

Memory Complexity. The procedure can be implemented in $\mathcal{O}(m^2)$ space. The key observation is that X^TX can be computed as $\sum_{i=1}^{n} t_i t_i^T$, where t_i is the i^{th} tuple in the dataset. Thus, X^TX can be computed incrementally by loading only one tuple at a time into memory, computing $t_i t_i^T$, and then adding that to a running sum, which can be stored in $\mathcal{O}(m^2)$ space. Note that instead of such an incremental computation, this can also be done in an embarrassingly parallel way where we horizontally partition the data (row-wise) and each partition is computed in parallel.Due to such low time and memory complexity, our approach scales gracefully to large datasets.

5.4 Trusted Machine Learning

In this section, we provide a theoretical justification of why conformance constraints are effective in identifying tuples for which learned models are likely to make incorrect predictions. To that end, we define *unsafe* tuples and show that an "ideal" conformance con-



(a) PCA **(b)** Disjoint PCA

Figure 5.4: Learning PCA-based constraints globally results in low quality constraints when data satisfies strong local constraints.

straint provides a sound and complete mechanism to detect unsafe tuples. In Section 5.3, we showed that low-variance projections construct strong conformance constraints, which yield a small conformance zone. We now make a similar argument, but in a slightly different way: we show that projections with zero variance give us equality constraints that are useful for trusted machine learning. We start with an example to provide the intuition.

Example 5.9. *Consider the airlines dataset D and assume that* all *tuples in D satisfy the equality constraint $AT - DT - DUR = 0$ (i.e.,* `lb` $=$ `ub` $= 0$*). Note that for equality constraint, the corresponding projection has* zero *variance—the lowest possible variance. Now, suppose that the task is to learn some function $f(AT, DT, DUR)$. If the above constraint holds for D, then the learning procedure can instead learn the function $g(AT, DT, DUR) = f(DT + DUR, DT, DUR)$. g will perform just as well as f on D: in fact, it will produce the same output as f on D. If a new serving tuple t satisfies the constraint, then $g(t) = f(t)$, and the prediction will be correct. However, if t does not satisfy the constraint, then $g(t)$ will likely be significantly different from $f(t)$. Hence, violation of the conformance constraint is a strong indicator of performance degradation of the learned prediction model. Note that f need not be a linear function: as long as g is also in the class of models that the learning procedure is searching over, the above argument holds.*

We proceed to formally define unsafe tuples. We use $[D; Y]$ to denote the *annotated dataset* obtained by appending the target attribute Y to a dataset D, and $\texttt{coDom}$ to denote Y's domain.

Definition 5.4 (Unsafe tuple)**.** *Given a class $\mathcal{C}$ of functions with signature $\mathbf{Dom}^m \mapsto \texttt{coDom}$, and an annotated dataset $[D; Y] \subset (\mathbf{Dom}^m \times \texttt{coDom})$, a tuple $t \in \mathbf{Dom}^m$ is unsafe w.r.t. $\mathcal{C}$ and $[D; Y]$, if $\exists f, g \in \mathcal{C}$ s.t. $f(D) = g(D) = Y$ but $f(t) \neq g(t)$.*

Intuitively, t is unsafe if there exist two different predictor functions f and g that agree on all tuples in D, but disagree on t. Since, we can never be sure whether the learning procedure learned f or g, we should be cautious about the prediction on t.

Example 5.9 suggests that t can be unsafe when all tuples in D satisfy the equality conformance constraint $f(\vec{A}) - g(\vec{A}) = 0$ but t does not. Hence, we can use the following conformance-constraint-centric approach for trusted machine learning:

1. Learn conformance constraints Φ for the dataset D.
2. Declare t as unsafe if t does not satisfy Φ.

The above approach is sound and complete for characterizing unsafe tuples, thanks to the following proposition.

Proposition 5.1. *There exists a conformance constraint Φ for D s.t. the following statement is true: "$\neg\Phi(t)$ iff t is unsafe w.r.t. $\mathcal{C}$ and $[D; Y]$ for all $t \in \mathbf{Dom}^m$".*

Proof. We show that the conformance constraint $\Phi := \forall f, g \in \mathcal{C} : f(D) = g(D) \Rightarrow f(t) - g(t) = 0$, is the required conformance constraint over D to detect tuples that are unsafe with respect to $\mathcal{C}$ and $[D; Y]$.

First, we claim that Φ is a conformance constraint for D. For this, we need to prove that every tuple in D satisfies Φ. Consider any $t' \in D$. We need to prove that $f(t') = g(t')$ for all $f, g \in \mathcal{C}$ s.t. $f(D) = g(D) = Y$. Since $t' \in D$, and since $f(D) = g(D)$, it follows that $f(t') = g(t')$. This shows that Φ is a conformance constraint for every tuple in D.

Next, we claim that Φ is *not satisfied* by exactly tuples that are unsafe w.r.t. $\mathcal{C}$ and $[D; Y]$. Consider any t' such that $\neg\Phi(t')$. By definition of Φ, it follows that there exist $f, g \in \mathcal{C}$ s.t. $f(D) = g(D) = Y$, but $f(t') \neq g(t')$. This is equivalent to saying that t' is unsafe, by definition. □

The required conformance constraint Φ is: $\forall f, g \in \mathcal{C} : f(D) = g(D) = Y \Rightarrow f(\vec{A}) - g(\vec{A}) = 0$, where A denotes the set of predictor attributes in D. Intuitively, the proposition states that when all possible pairs of functions that agree on D also agree on t, only then the prediction on t can be (fully) trusted.

5.4.1 Applicability

Generalization to noisy setting. While our analysis and formalization for using conformance constraints for TML focused on the noise-free setting, the intuition generalizes to noisy data. Specifically, suppose that f and g are two possible functions a model may learn over D; then, we expect that the difference $f - g$ will have small variance over D, and thus would be a good conformance constraint. In turn, the violation of this constraint would mean that f and g diverge on a tuple t (making t unsafe); since we are oblivious of the function the model learned, prediction on t is untrustworthy.

False positives. Conformance constraints are designed to work in a model-agnostic setting. Although this setting is of great practical importance, designing a perfect mechanism for quantifying trust in ML model predictions, while remaining completely model-agnostic, is challenging. It raises the concern of *false positives*: conformance constraints may incorrectly flag tuples for which the model's prediction is in fact correct. This may happen when the model ignores the trend that conformance constraints learn. Since we are oblivious of the prediction task and the model, it is preferable that conformance constraints behave rather *conservatively* so that the users can be cautious about potentially unsafe tuples. Moreover, if a model ignores some attributes (or their interactions) during training, it

is still necessary to learn conformance constraints over them. Particularly, in case of concept drift [313], the ground truth may start depending on those attributes, and by learning conformance constraints over all attributes, we can better detect potential model failures.

False negatives. Another concern involving conformance constraints is of *false negatives*: linear conformance constraints may miss nonlinear constraints, and thus fail to identify some unsafe tuples. However, the linear dependencies modeled in conformance constraints persist even after sophisticated (nonlinear) attribute transformations. Therefore, violation of conformance constraints is a strong indicator of potential failure of a possibly nonlinear model.

Modeling nonlinear constraints. While linear conformance constraints are the most common ones, we note that our framework can be easily extended to support nonlinear conformance constraints using *kernel functions* [290]—which offer an efficient, scalable, and powerful mechanism to learn nonlinear decision boundaries for support vector machines (also known as "kernel trick"). Briefly, instead of explicitly augmenting the dataset with transformed nonlinear attributes—which grows exponentially with the desired degree of polynomials—kernel functions enable *implicit* search for nonlinear models. The same idea also applies for PCA called kernel-PCA [13, 164]. While we limit our evaluation to only linear kernel, polynomial kernels—e.g., radial basis function (RBF) [181]—can be plugged into our framework to model nonlinear conformance constraints.

In general, our conformance language is not guaranteed to model all possible functions that an ML model can potentially learn, and thus is not guaranteed to find the best conformance constraint. However, our empirical evaluation on real-world datasets shows that our language models conformance constraints effectively.

5.5 Experimental Evaluation

We now present experimental evaluation to demonstrate the effectiveness of conformance constraints over our two case-study applications (Section 5.1): trusted machine learning and data drift. Our experiments target the following research questions:

- How effective are conformance constraints for trusted machine learning? Is there a relationship between constraint violation score and the ML model's prediction accuracy? (Section 5.5.2)
- Can conformance constraints be used to quantify data drift? How do they compare to other state-of-the-art drift-detection techniques? (Section 5.5.3)

Efficiency. In all our experiments, our algorithms for deriving conformance constraints were extremely fast, and took only a few seconds even for datasets with 6 million rows. The number of attributes were reasonably small ($\sim$40), which is true for most practical applications. As our theoretical analysis showed (Section 5.3.3), our approach is linear in the number of data rows and cubic in the number of attributes. Since the runtime performance of our techniques is straightforward, we opted to not include further discussion of efficiency here and instead focus this empirical analysis on the techniques' effectiveness.

5.5.1 Experimental Settings

We now describe our experimental settings, including implementation details and the datasets we used for the experiments.

5.5.1.1 Implementation: CCSynth

We created an open-source implementation of conformance constraints and our method for synthesizing them, CCSynth, in Python 3. Experiments were run on a Windows 10 machine (3.60 GHz processor and 16GB RAM).

5.5.1.2 Datasets

Airlines [9] contains data about flights and has 14 attributes: year, month, day, day of week, departure time, arrival time, carrier, flight number, elapsed time, origin, destination, distance, diverted, and arrival delay. We used a subset of the data containing all flight information for year 2008. In this dataset, most of the attributes follow uniform distribution (e.g., month, day, arrival and departure time, etc.); elapsed time and distance follow skewed distribution with higher concentration towards small values (implying that shorter flights are more common); arrival delay follows a slightly skewed gaussian distribution implying most flights are on-time, few arrive late and even fewer arrive early. The training and serving datasets contain 5.4M and 0.4M rows, respectively. We use this dataset for a regression task of predicting the arrival delay in the trusted machine learning problem setting.

Human Activity Recognition (HAR) [306] is a real-world dataset about activities for 15 individuals, 8 males and 7 females, with varying fitness levels and BMIs. We use data from two sensors—accelerometer and gyroscope—attached to 6 body locations—head, shin, thigh, upper arm, waist, and chest. We consider 5 activities—lying, running, sitting, standing, and walking. The dataset contains 36 numerical attributes (2 sensors $\times$ 6 body-locations $\times$ 3 co-ordinates) and 2 categorical attributes—activity-type and person-ID. We pre-processed the dataset to aggregate the measurements over a small time window, resulting in 10,000 tuples per person and activity, for a total of 750,000 tuples.

Extreme Verification Latency (EVL) [69] is a widely used benchmark to evaluate drift-detection algorithms in non-stationary environments under extreme verification latency. It contains 16 synthetic datasets with incremental and gradual concept drifts. The number of attributes of these datasets vary from 2 to 6 and each of them has one categorical attribute.

5.5.2 Trusted Machine Learning

We now demonstrate the applicability of CCSYNTH in the trusted machine learning problem. We show that, serving tuples that violate the conformance constraints derived

from the training data are unsafe, and therefore, a machine-learned model is more likely to perform poorly on those tuples.

Airlines. For the airlines dataset, we design a regression task of predicting the arrival delay and train a linear regression model for the task. Our goal is to observe whether the mean absolute error of the predictions (positively) correlates to the constraint violation for the serving tuples. In a process analogous to the one described in Example 5.1, our training dataset (`Train`) comprises of a subset of daytime flights, flights that have arrival time later than the departure time (in 24 hour format). We design three serving sets: (1) `Daytime`: similar to `Train`, but another subset, (2) `Overnight`: flights that have arrival time earlier than the departure time (the dataset does not explicitly report the date of arrival), and (3) `Mixed`: a mixture of `Daytime` and `Overnight`. A few sample tuples of this dataset are in Fig. 5.1.

Our experiment involves the following steps: (1) CCSYNTH computes conformance constraints Φ over `Train`, while *ignoring* the target attribute `delay`. (2) We compute average constraint violation for all four datasets—`Train`, `Daytime`, `Overnight`, and `Mixed`—against Φ (first row of Fig. 5.5). (3) We train a linear regression model over `Train`—including `delay`—that learns to predict arrival delay. (4) We compute mean absolute error (MAE) of the prediction accuracy of the regressor over the four datasets (second row of Fig. 5.5). We find that constraint violation is a very good proxy for prediction error, as they vary in a similar manner across the four datasets. The reason is that the model implicitly assumes that the constraints (e.g., $AT - DT - DUR \approx 0$) derived by CCSYNTH will always hold, and thus, deteriorates when the assumption no longer holds.

To observe the rate of false positives and false negatives, we investigate the relationship between constraint violation and prediction error at tuple-level granularity. We sample 1000 tuples from `Mixed` and organize them by decreasing order of violations (Fig. 5.6). For all the tuples (on the left) that incur high constraint violations, the regression model incurs high error for them as well. This implies that CCSYNTH reports no false positives.

	Train	Serving		
		Daytime	Overnight	Mixed
Average violation	0.02%	0.02%	27.68%	8.87%
MAE	18.95	18.89	80.54	38.60

Figure 5.5: Average constraint violation (in percentage) and MAE (for linear regression) of four data splits on the airlines dataset. The constraints were learned on `train`, excluding the target attribute, delay.

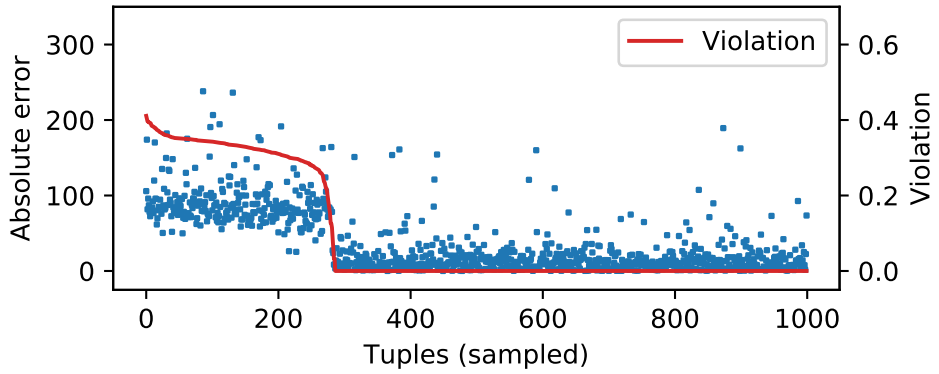


Figure 5.6: Constraint violation strongly correlates with the absolute error of delay prediction of a linear regression model.

There are some false negatives (right part of the graph), where violation is low, but the prediction error is high. Nevertheless, such false negatives are very few.

HAR. On the HAR dataset, we design a supervised classification task to identify persons from their activity data that contains 36 numerical attributes. We construct `train_x` with data for sedentary activities (lying, standing, and sitting), and `train_y` with the corresponding person-IDs. We learn conformance constraints on `train_x`, and train a Logistic Regression (LR) classifier using the annotated dataset [`train_x`; `train_y`]. During serving, we mix mobile activities (walking and running) with held-out data for sedentary activities and observe how the classification's mean accuracy-drop (i.e., how much the mean prediction accuracy decreases compared to the mean prediction accuracy over the training data) relates to average constraint violation. To avoid any artifact due to sampling bias, we repeat this experiment 10 times for different subsets of the data by randomly sam-

pling 5000 data points for each of training and serving. Fig. 5.7(a) depicts our findings: classification degradation has a clear positive correlation with violation (pcc = 0.99 with p-value = 0).

Noise sensitivity. Intuitively, noise weakens conformance constraints by increasing variance in the training data, which results in reduced violations of the serving data. However, this is desirable: as more noise makes machine-learned models less likely to overfit and thus, more robust. In our experiment for observing noise sensitivity of conformance constraints, we use only mobile activity data as the serving set and start with sedentary data as the training set. Then we gradually introduce noise in the training set by mixing mobile activity data. As Fig. 5.7(b) shows, when more noise is added to the training data, conformance constraints start getting weaker; this leads to reduction in violations. However, the classifier also becomes robust with more noise, which is evident from gradual decrease in accuracy-drop (i.e., increase in accuracy). Therefore, even under the presence of noise, the positive correlation between classification degradation and violation persists (pcc = 0.82 with p-value = 0.002).

The key takeaway is that CCSYNTH derives conformance constraints whose violation is a strong proxy of model prediction accuracy.

5.5.3 Data Drift

We now present results of using CCSYNTH as a drift-detection tool; specifically, for *quantifying* drift in data. Given a baseline dataset D, and a new dataset D', the drift is measured as average violation of tuples in D' on constraints learned for D.

HAR. We perform three drift-quantification experiments on the HAR dataset which we discuss next.

Gradual drift. For observing how CCSYNTH detects gradual drift, we introduce drift in an organic way. The initial training dataset contains data of exactly one activity for each person. This is a realistic scenario as one can think of it as taking a snapshot of what a group

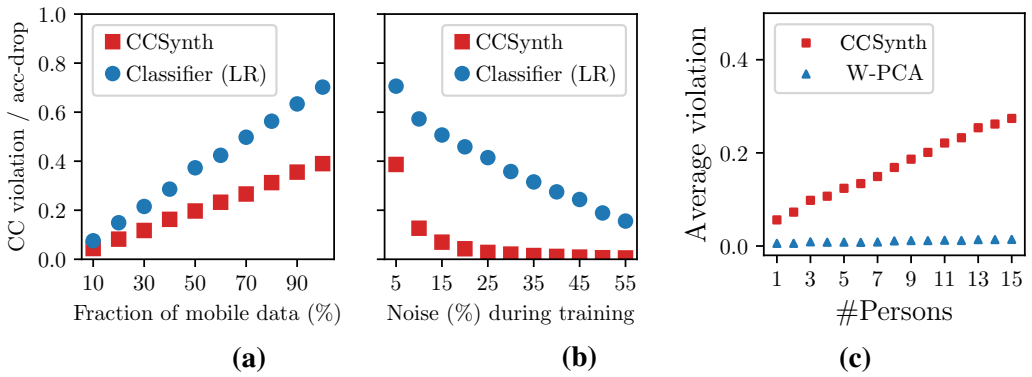


Figure 5.7: (a) As a higher fraction of mobile activity data is mixed with sedentary activity data, conformance constraints are violated more, and the classifier's mean accuracy-drop increases. (b) As more noise is added during training, conformance constraints get weaker, leading to less violation and decreased accuracy-drop. (c) CCSYNTH detects the gradual local drift on the HAR dataset as more people start changing their activities. In contrast, weighted-PCA (W-PCA) fails to detect drift in absence of a strong global drift.

of people are doing during a reasonably small time window. We introduce gradual drift to the initial dataset by altering the activity of one person at a time. To control the amount of drift, we use a parameter K. When $K = 1$, the first person switches their activity, i.e., we replace the tuples corresponding to the first person performing activity A with new tuples that correspond to the same person performing another activity B. When $K = 2$, the second person switches their activity in a similar fashion, and so on. As we increase K from 1 to 15, we expect a gradual increase in the drift magnitude compared to the initial training data. When $K = 15$, all persons switch their activities, and we expect to observe maximum drift. We repeat this experiment 10 times, and display the average constraint violation in Figure 5.7(c). We note that the drift magnitude (violation) indeed increases as more people alter their activities.

In contrast, the baseline weighted-PCA (W-PCA) method fails to detect this drift. This is because W-PCA does not model local constraints (who is doing what), and learns some weaker global constraints. Thus, it fails to detect the gradual local drift, as the global situation "a group of people are performing some activities" is not changing. In contrast,

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	Fitness	BMI	Gender
p1	0	0.3	0.4	0.3	0.3	0.3	0.3	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.3	Moderate	Underweight	Female
p2	0.4	0	0.4	0.3	0.3	0.3	0.3	0.5	0.3	0.3	0.4	0.3	0.4	0.4	0.3	Moderate	Normal	Male
p3	0.5	0.5	0	0.4	0.5	0.5	0.5	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	Moderate	Overweight	Male
p4	0.2	0.2	0.3	0	0.3	0.2	0.3	0.5	0.2	0.3	0.2	0.3	0.3	0.3	0.3	Moderate	Normal	Male
p5	0.2	0.3	0.4	0.2	0	0.2	0.2	0.4	0.2	0.3	0.3	0.3	0.3	0.3	0.3	Moderate	Normal	Male
p6	0.3	0.3	0.4	0.3	0.3	0	0.3	0.4	0.2	0.2	0.2	0.2	0.3	0.2	0.4	High	Normal	Female
p7	0.3	0.3	0.4	0.3	0.3	0.3	0	0.4	0.3	0.2	0.3	0.3	0.3	0.3	0.3	Moderate	Overweight	Male
p8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.4	0.5	0.4	0.5	0.5	0.5	0.5	Low	Obese	Female
p9	0.4	0.4	0.5	0.4	0.3	0.3	0.5	0.5	0	0.3	0.4	0.4	0.4	0.4	0.5	High	Overweight	Male
p10	0.4	0.3	0.4	0.3	0.3	0.3	0.4	0.5	0.3	0	0.3	0.3	0.3	0.4	0.3	Moderate	Obese	Male
p11	0.4	0.4	0.5	0.3	0.4	0.3	0.4	0.5	0.3	0.3	0	0.3	0.3	0.3	0.4	Moderate	Normal	Female
p12	0.3	0.3	0.4	0.3	0.3	0.3	0.3	0.5	0.3	0.3	0.3	0	0.3	0.4	0.4	Moderate	Normal	Female
p13	0.3	0.3	0.4	0.3	0.3	0.2	0.3	0.4	0.2	0.3	0.2	0.3	0	0.3	0.4	Moderate	Normal	Female
p14	0.3	0.3	0.4	0.3	0.3	0.2	0.3	0.4	0.2	0.3	0.3	0.3	0.3	0	0.3	High	Normal	Male
p15	0.4	0.4	0.5	0.4	0.4	0.5	0.4	0.5	0.5	0.4	0.4	0.4	0.5	0.5	0	Low	Normal	Female

Figure 5.8: Inter-person constraint violation heat map. Each person has a very low self-violation.

CCSYNTH learns disjunctive constraints that encode which person is performing which activity, and hence, is capable to detect drift when individuals switch activities.

Inter-person drift. The goal of this experiment is to observe how effectively conformance constraints can model the representation of an entity and whether such learned representations can be used to accurately quantify drift between two entities. We use half of each person's data to learn the constraints, and compute violation on the held-out data. CCSYNTH learns disjunctive constraints for each person over all activities, and then we use the violation w.r.t. the learned constraints to measure how much the other persons drift. While computing drift between two persons, we compute activity-wise constraint violation scores and then average them out. In Fig. 5.8, the violation score at row `p1` and column `p2` denotes how much `p2` drifts from `p1`. As one would expect, we observe a very low self-drift across the diagonal. Interestingly, our result also shows that some people are more different from others, which appears to have some correlation with (the hidden ground truth) fitness and BMI values. This asserts that the constraints we learn for each person are

	lying	standing	sitting	walking	running
lying	0.05	0.41	0.57	0.68	0.78
standing	0.62	0.02	0.51	0.56	0.71
sitting	0.57	0.23	0.04	0.59	0.72
walking	0.21	0.01	0.06	0	0.25
running	0.12	0	0.03	0.02	0.01

Figure 5.9: Inter-activity constraint violation heat map. Mobile activities violate the constraints of the sedentary activities more.

an accurate abstraction of that person's activities, as people do not deviate too much from their usual activity patterns.

Inter-activity drift. Similar to inter-person constraint violation, we also compute inter-activity constraint violation. Figure 5.9 shows our findings. Note the asymmetry of violation scores between activities, e.g., `running` is violating the constraints of `standing` much more than the other way around. A close observation reveals that, all mobile activities violate all sedentary activities more than the other way around. This is because, the mobile activities behave as a "safety envelope" for the sedentary activities. E.g., while a person walks, she also stands (for a brief moment); but the opposite does not happen.

EVL. We now compare CCSYNTH against other state-of-the-art drift detection approaches on the EVL benchmark.

Baseline Approaches. In our experiments, we use two drift-detection approaches as baselines which we describe below:

(1) PCA-SPLL [194] [2], similar to us, also argues that principal components with lower variance are more sensitive to a general drift, and uses those for dimensionality reduction. It then models multivariate distribution over the reduced dimensions and applies semi-parametric log-likelihood (SPLL) to detect drift between two multivariate distributions.

[2] SPLL source code: `github.com/LucyKuncheva/Change-detection`

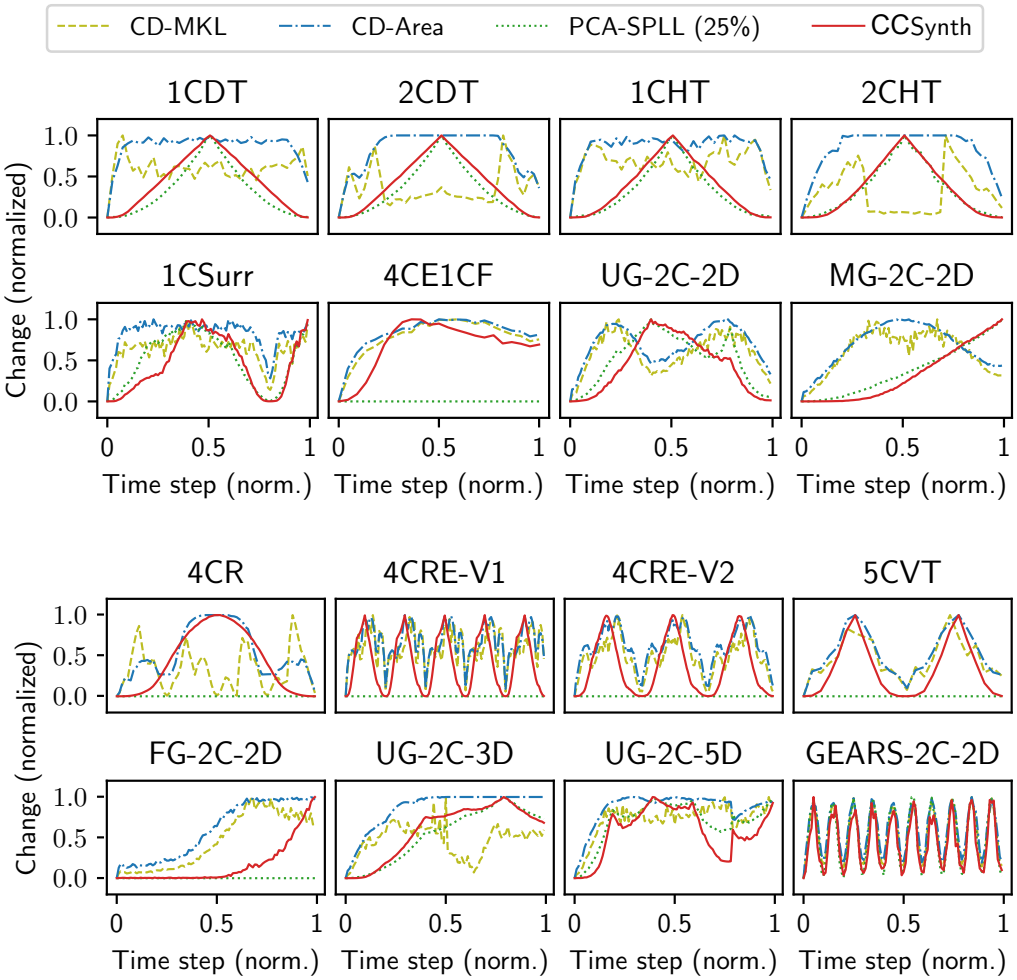


Figure 5.10: In the EVL benchmark, CCSYNTH quantifies drift correctly for all cases, outperforming other approaches. PCA-SPLL fails to detect drift in a few cases by discarding all principal components; CD-MKL and CD-Area are too sensitive to small drift and detect spurious drifts.

However, PCA-SPLL discards all high-variance principal components and does not model disjunctive constraints.

(2) CD (Change Detection) [263] [3] is another PCA-based approach for drift detection in data streams. But unlike PCA-SPLL, it ignores low-variance principal components. CD projects the data onto top k high-variance principal components, which results into multiple univariate distributions. We compare against two variants of CD: CD-Area, which uses the

[3] CD source code: `mine.kaust.edu.sa/Pages/Software.aspx`

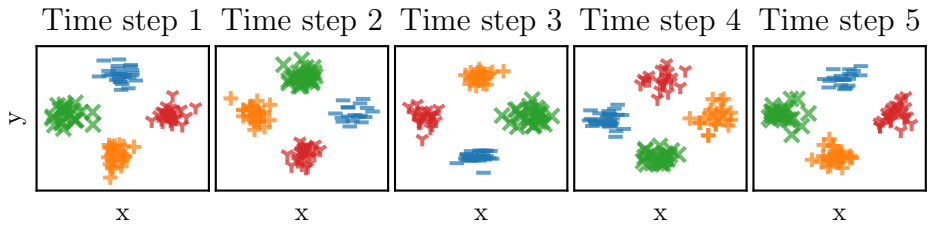


Figure 5.11: Snapshots over time for 4CR dataset with local drift. It reaches maximum drift from the initial distribution at time step 3 and goes back to the initial distribution at time step 5.

intersection area under the curves of two density functions as a divergence metric, and CD-MKL, which uses Maximum KL-divergence as a symmetric divergence metric, to compute divergence between the univariate distributions.

Figure 5.10 depicts how CCSYNTH compares against CD-MKL, CD-Area, and PCA-SPLL, on 16 datasets in the EVL benchmark. For PCA-SPLL, we retain principal components that contribute to a cumulative explained variance below 25%. Beyond drift detection, which just detects if drift is above some threshold, we focus on drift quantification. A tuple (x, y) in the plots denotes that drift magnitude for dataset at x^{th} time window, w.r.t. the dataset at the first time window, is y. Since different approaches report drift magnitudes in different scales, we normalize the drift values within $[0, 1]$. Additionally, since different datasets have different number of time windows, for the ease of exposition, we normalize the time window indices. Below we state our key findings from this experiment:

CCSYNTH*'s drift quantification matches the ground truth.* In all of the datasets in the EVL benchmark, CCSYNTH is able to correctly quantify the drift, which matches the ground truth exceptionally well.[4] In contrast, as CD focuses on detecting the drift point, it is ill-equipped to precisely quantify the drift, which is demonstrated in several cases (e.g., 2CHT), where CD fails to distinguish the deviation in drift magnitudes. In contrast, both PCA-SPLL and CCSYNTH correctly quantify the drift. Since CD only retains high-variance principal components, it is more susceptible to noise and considers noise in the

[4]EVL video: `sites.google.com/site/nonstationaryarchive/home`

dataset as significant drift, which leads to incorrect drift quantification. In contrast, PCA-SPLL and CCSYNTH ignore the noise and only capture the general notion of drift. In all of the EVL datasets, we found CD-Area to work better than CD-MKL, which also agrees with the authors' experiments. Since CD only retains high-variance principal components, it is more susceptible to noise and considers noise in the dataset as significant drift, which leads to incorrect drift quantification. In contrast, PCA-SPLL and CCSYNTH ignore the noise and only capture the general notion of drift.

CCSYNTH *models local drift.* When the dataset contains instances from multiple classes, the drift may be just local, and not global. Figure 5.11 demonstrates such a scenario for the 4CR dataset. If we ignore the color/shape of the tuples, we will not observe any significant drift across different time steps. In such cases, PCA-SPLL fails to detect drift (4CR, 4CRE-V2, and FG-2C-2D). In contrast, CCSYNTH learns disjunctive constraints and quantifies local drifts accurately.

The key takeaway is that CCSYNTH can effectively detect data drift, both global and local, is robust across drift patterns, and significantly outperforms the state of the arts.

5.6 Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning (COCO)

Traditionally, integrity constraints are specified along with the schema to keep the "integrity" of new data over that schema, e.g., to prevent erroneous tuple insertion. Like other data profiles, conformance constraints can also be specified during schema design. However, it is difficult to come up with the right set of conformance constraints manually. First, real-world data is often noisy and pre-specified constraints can be too strict, resulting in unwanted conservativeness during future data operations. Second, figuring out the right set of conformance constraints requires complete understanding of the domain and semantics of each attribute (e.g., `duration` includes both the flight time and the `delay`). Third, finding the coefficients manually is tedious: it requires knowledge of the measurement units of

the attributes (e.g., `distance` is in miles). Thus, it is preferable to have a mechanism for automatic discovery of conformance constraints from a given dataset.

Interactive exploration of conformance constraints. A shortcoming of our approach for conformance constraint discovery is that it prioritizes *effectiveness* (finds the strongest conformance constraints) over *interpretability* (may involve a large number of numerical attributes). For example, if there exists a correlation between the two constraints over the data, i.e., all tuples tend to incur similar violation scores against both, then they can be merged to derive a *stronger* constraint. However, such an increase in strength comes at the cost of interpretability, as a conformance constraint that involves too many numerical attributes is less interpretable. To address this issue, CoCo allows the user to tune the parameters for conformance constraint discovery: they can specify the attributes that are of interest and the maximum number of attributes they prefer within the conformance constraints. While this might produce suboptimal constraints, it nonetheless is valuable because it gives users more control and confidence. In particular, CoCo displays the strength of each discovered conformance constraint and also shows the top 15 most violating tuples, which helps the user judge the effectiveness of the constraints.

Conformance constraints for data cleaning. An obvious application of conformance constraints is data cleaning. The idea is to first learn conformance constraints over a clean (reference) dataset and then consult the learned constraints for data cleaning. Specifically, violation of the learned constraints by a new tuple indicates that the tuple may be dirty. Furthermore, the closed form expression of conformance constraints also allows us to provide suggestions regarding valid values for each cell (Figure 5.12), which can guide the user throughout the data cleaning process. CoCo provides an interactive data cleaning solution where the user can edit a cell within a tuple and gets immediate feedback about the corresponding change in constraint violation: reduction or removal of constraint violation confirm a correct cleaning operation (Figure 5.13).

We now present CoCo, a tool that discovers *interpretable* conformance constraints, based on user preferences, and how violation of conformance constraints can facilitate in interactive data cleaning.

5.6.1 Interpretable Conformance Constraints

Involving all attributes during learning yields the strongest set of conformance constraints. However, it may result in poor interpretability. CoCo allows the user to specify a parameter K that denotes the maximum number of attributes preferred within a conformance constraint. Additionally, CoCo allows the user to tune a second parameter $\mathcal{A}$ which denotes a set of attributes over which conformance constraint discovery should be limited. With K and $\mathcal{A}$ specified by the user, CoCo learns constraints on different vertical partitions of the reference dataset, with each partition limited to a subset of K attributes from $\mathcal{A}$. Although, theoretically, this results in combinatorial explosion, the set $\mathcal{A}$ is expected to be small and the value K must be small (≤ 5) for ensuring interpretability. Therefore, in practice, such a runtime complexity is acceptable.

CoCo preprocesses the discovered conformance constraints before presenting them to the user. The preprocessing involves (1) removing attributes that are associated with very small weights within a constraint, as this improvs interpretability, and (2) removing redundant constraints that involve the same attributes (and, thus, are equivalent) by keeping only one of the redundant constraints. E.g., the constraints $-2 \leq X+Y \leq 2$ and $-4 \leq 2X+2Y \leq 4$ are equivalent and keeping only one of them is sufficient.

5.6.2 Generating Suggestions for Data Cleaning

For data cleaning, we use a reasonably clean dataset as a reference data. Discovery of conformance constraints requires only a small amount of data that are reasonably clean (number of tuples should be more than the number of attributes for PCA to work). However, our technique for conformance constraint discovery is robust to uniformly distributed outliers across all projections, and straightforward modification in bound computation (tight-

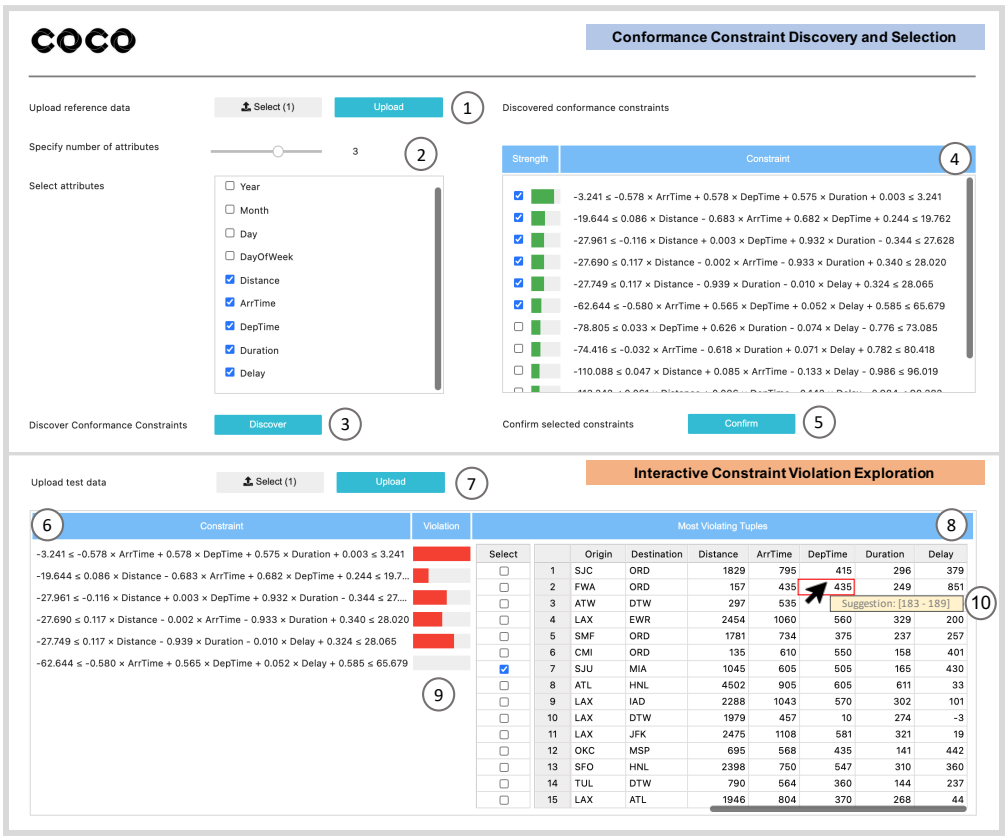


Figure 5.12: User scenario for CoCo: ① upload reference (clean) data, ② select relevant attributes and specify the maximum number of attributes desired within the conformance constraints, ③ discover conformance constraints, ④ view the discovered constraints along with their strengths, ⑤ select a subset of conformance constraints for further exploration, ⑥ view the selected conformance constraints, ⑦ upload test (unclean) data, ⑧ view top 15 most violating tuples and select a tuple for checking its violations, ⑨ view constraint-wise violations for the selected tuple, ⑩ hover on a cell to get suggestion on how to alter its value to satisfy the conformance constraints.

ening) can adjust to noisy data. For a tuple t, all of whose attributes are correct, except the j^{th} attribute, we can generate a range of valid values for the j^{th} attribute to fix it by exploiting a given conformance constraint C. When we have a set of such constraints C_1, C_2, ..., we can generate a range of valid values for each C_i and take their intersection to find a range that will satisfy *all* of the constraints. Using this mechanism, CoCo generates suggestions on how to alter value of a single cell of a tuple to fix it. However, we note that when mul-

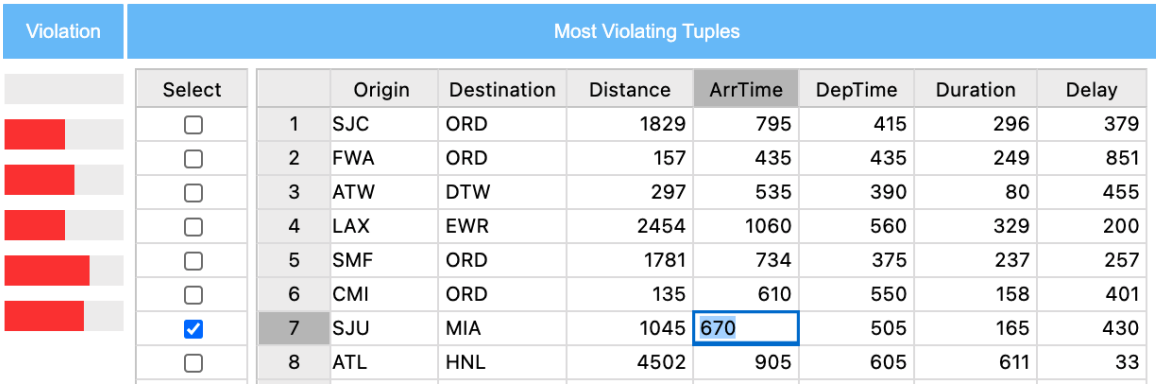


Figure 5.13: Interactive data cleaning: the user edits a cell in-place and views changes in constraint violation. Changing `ArrTime` from 605 to 670 for the 7^{th} tuple reduces its violation against the first constraint, but increases violation against the 6^{th} constraint.

tiple attributes are incorrect, such a suggestion may not be helpful. Figures 5.12 and 5.13 show two screenshots of CoCo.

5.7 Explaining Tuple Non-conformance (ExTuNe)

In data-driven systems, we often encounter tuples on which the predictions of a machine-learned model are untrustworthy. Early detection of such tuples is necessary to ensure Trusted Machine Learning (TML) [164]. Since data is inherently an incomplete specification for any task, invariably there will exist multiple different models that can be learned from the given training dataset. This in return introduces uncertainty in predictions made using any specific model learned from the dataset. Motivated by the issue of trusting the predictions made by machine learning, we define *non-conforming tuples*—which are tuples on which a machine-learned model makes untrustworthy predictions. One might see non-conforming tuples as outliers. However, traditional definition of outlier overlooks the fact that some outliers are non-conforming, for certain tasks, while others are not.

Example 5.10. *Consider a dataset with three training tuples with predictor attributes* x_1 *and* x_2*:* $\{(1, 10), (2, 20), (4, 40)\}$*. We consider a task-agnostic setting and hence omit the target attribute. Now consider two new tuples:* $t_1 = (3, 12)$ *and* $t_2 = (10, 100)$*. A*

traditional distance-based outlier detector will mark t_2 as an outlier and possibly t_1 as an inlier. However, ML models often exploit the consistent relationship observed between the attribute pairs ($x_2 = 10x_1$) within the training tuples, and assume it as an invariant or constraint. Since t_1 violates this constraint, a model that uses $10x_1$ instead of x_2 is likely to make inaccurate prediction on t_1, if x_2 is the true predictor. Here, t_1 is non-conforming and t_2 is conforming with the constraint.

Example 5.10 shows the shortcoming of distance-based outliers in capturing the notion of non-conformance. Identification of non-conforming tuples is a two-step process: (1) learning *conformance constraints* from the training dataset, and (2) checking for violation of the learned conformance constraints by the test tuples; violation of conformance constraints indicates non-conformance. We present EXTUNE, which (1) detects non-conforming tuples and quantifies the degree of non-conformance based on [101], and (2) extends [101] to explain the cause of non-conformance by assigning degree of responsibility to tuple attributes.

EXTUNE's attribute responsibility has connection to one class classification (OCC) and feature importance and feature selection for binary classification. However, EXTUNE works in an OCC setting, where data from other classes (counter-examples) are unavailable during training. Unlike binary classifiers, which start with the knowledge of tuples from both classes, EXTUNE's goal is to (1) predict which tuples fall in the negative class (i.e., non-conforming), and (2) assign responsibility to the attributes for non-conformance. Feature importance for OCC fails here too, as it only considers the training data and overlooks the test tuples. Responsibility computation in EXTUNE is generic and can be applied to any technique (distance- or pattern-based outlier detectors or classifiers) that distinguishes two classes (representing conforming and non-conforming tuples); however, EXTUNE applies it to non-conformance based on conformance constraints due to our interest in TML.

We present how EXTUNE detects the non-conforming tuples and provides real-time explanation for non-conformance. We proceed to discuss, at a high level, the solution sketch of EXTUNE, provide a user scenario, and conclude by some evaluation results.

5.7.1 Solution Sketch

The key component of EXTUNE is a conformance constraint learner, which learns conformance constraints from the training tuples. When a new test tuple arrives, EXTUNE checks if the test tuple satisfies the conformance constraint, and if it does not, EXTUNE quantifies the degree of non-conformance. To generate explanation for non-conformance, EXTUNE uses an intervention-centric approach that alters values of attributes, and observes change in conformance constraint violation. EXTUNE then assigns degree of responsibility to the attributes for non-conformance (Section 5.7.1.1).

5.7.1.1 Responsibility for Non-conformance

Responsibility quantifies how much attributes of a tuple contribute for causing non-conformance. We adapt Halpern and Pearl's [132] definition of causality. To measure the degree of responsibility of attributes of tuples that violate conformance constraints, we adapt the notion of *degree of responsibility* from Meliou et al. [227]. We reason about causality by *intervening* on attribute-values: we alter value of an attribute to the attribute-mean over the training dataset, and observe how it affects the tuple's conformance constraint violation.

Counterfactual cause. By definition, C is a counterfactual cause of an event E if E would not occur unless C occurs. In our case, an event is the violation of conformance constraint I by a tuple $\vec{x} = \langle x_1, x_2, \ldots, x_i, \ldots, x_m \rangle$, i.e., $\vec{x} \nvdash I$. The fact $A_i = x_i$ is a counterfactual cause for the violation if intervening on the value of A_i *prevents* the conformance constraint violation: $\langle x_1, x_2, \ldots, \mu_i, \ldots x_m \rangle \vdash I$, where μ_i denotes the attribute-mean of A_i. In such case, we assign responsibility 1 to attribute A_i.

Actual cause. C is an actual cause of E if E counterfactually depends on C under some *permissive contingency* [227]. To determine causality for an attribute-value that is not a counterfactual cause, we allow alteration of other attributes *that are not counterfactual causes* as permissive contingency. However, we only allow alterations that involve changing the value of an attribute to its attribute-mean. We define *minimum support* to be the minimum number of attribute-value alterations required within a contingency, among all possible contingencies, to achieve counterfactual causality. Contingencies with minimum support are the *minimal contingencies*. We assign responsibility $\frac{1}{M+1}$ to an attribute with minimum support M.

Example 5.11. *Consider a tuple $\vec{x}$=$\langle x_1, x_2, \ldots, x_m\rangle$ s.t. $\vec{x} \nvdash I$. Suppose that, individually, none of $A_1 = x_1$ and $A_2 = x_2$ are counterfactual causes for the violation, but $\langle \mu_1, \mu_2, x_3, \ldots, x_m\rangle \vdash I$. Since $A_1 = x_1$ is a counterfactual cause for the violation under the contingency $A_2 = \mu_2$, it is an actual cause with minimum support 1. So, we assign responsibility $\frac{1}{1+1} = \frac{1}{2}$ to A_1. Using a symmetric argument, A_2 also gets responsibility $\frac{1}{2}$.*

Approximating minimal contingency. Finding the minimal contingency for an actual cause is NP-hard [227]. Hence, we follow a greedy approach to find an approximate minimal contingency. While this approach does not guarantee optimality and might even fail to identify an actual cause, it works well in practice, particularly when responsibility is aggregated over a large set of non-conforming tuples.

The greedy approach iteratively selects attributes to alter based on their contribution to the conformance constraint violation. For example, consider the conformance constraint $-3 \leq F(\vec{A}) \leq 3$ where $F(\vec{A}) = 2A_1{+}4A_2{+}7A_3{+}5A_4{+}4A_5$ and attribute-mean is 0 for all attributes. For a tuple $\vec{x} = \langle 6, -5, 2, 3, 2\rangle$, $F(\vec{x}) = 12 + (-20) + 14 + 15 + 8 = 29$. None of the attribute-values are counterfactual in this case. Now, suppose that we are looking for minimal contingency (if one exists) of $A_1 = 6$. Clearly, $\vec{x}$ violated the conformance constraint due to 29 being too high than the upper bound 3 of the conformance constraint.

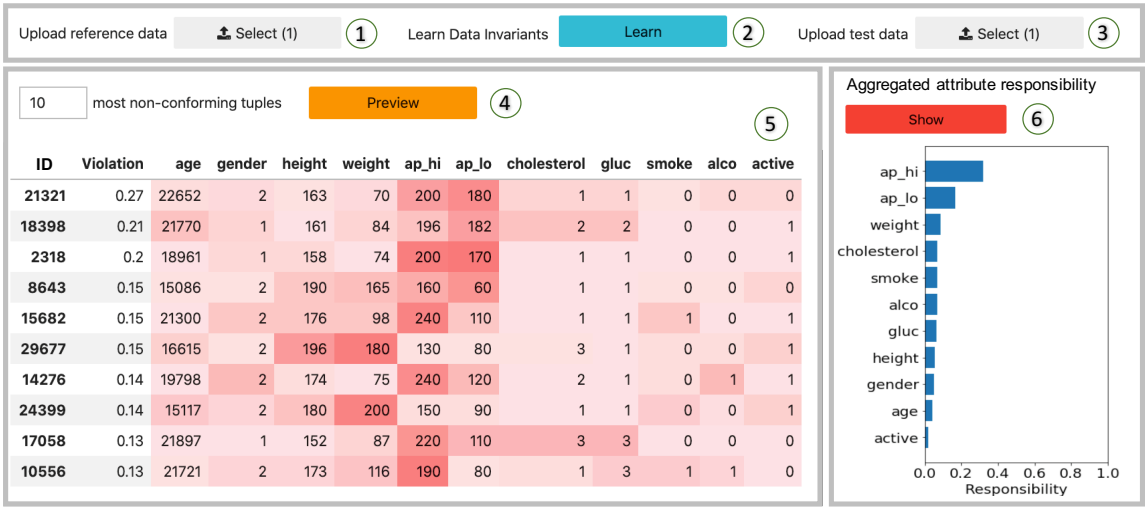
We start by *greedily picking* A_4 to alter as it contributes *the maximum* (15) to $F(\vec{x})$. We obtain $\vec{x}' = \langle 6, -5, 2, \underline{0}, 2\rangle$ and $F(\vec{x}') = 12 + (-20) + 14 + 0 + 8 = 14$. With this contingency, $A_1 = 6$ now counterfactually causes the violation as for $\vec{x}'' = \langle \underline{0}, -5, 2, \underline{0}, 2\rangle$, $F(\vec{x}'') = 0 + (-20) + 14 + 0 + 8 = 2$, which satisfies the conformance constraint. So, we found $M = 1$ and hence assign responsibility $\frac{1}{2}$ to A_1. If this was not a valid contingency, we would continue to intervene on the next most contributing attribute (A_3 in this case) to find a valid contingency.

Aggregating responsibility. Following the above procedure, we compute $R_{i,j,k}$ which denotes the responsibility of attribute A_i for causing tuple $\vec{x}^{(j)}$ to violate conformance constraint I_k. To ensure that responsibilities are proportionate to the degree of violations, we multiply $R_{i,j,k}$ by $violation(\vec{x}^{(j)}, I_k)$. Finally, we aggregate responsibilities over all conformance constraints and tuples and normalize the responsibilities across all attributes.

Figure 5.14 shows a screenshot of EXTUNE's graphical user interface, over a real-world cardiovascular disease dataset [52]. For the first tuple, the non-conformance comes mostly from the abnormally high blood pressures. For the second tuple, besides abnormally high blood pressures, he also has above normal glucose and cholesterol levels. For the sixth tuple, although the blood pressures look normal, she has an abnormally high weight of 180 kg (397 lbs) which is one of the prime causes for her non-conformance. Systolic blood pressure is most responsible for non-conformance, followed by diastolic blood pressure. This is very meaningful since abnormal blood pressure is a primary indicator for cardiovascular disease. This is followed by weight, cholesterol level, and smoking, three other well-known risk factors.

5.7.2 Experiments

We now present the effectiveness of EXTUNE through several case studies.



ID	Violation	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
21321	0.27	22652	2	163	70	200	180	1	1	0	0	0
18398	0.21	21770	1	161	84	196	182	2	2	0	0	1
2318	0.2	18961	1	158	74	200	170	1	1	0	0	1
8643	0.15	15086	2	190	165	160	60	1	1	0	0	0
15682	0.15	21300	2	176	98	240	110	1	1	1	0	1
29677	0.15	16615	2	196	180	130	80	3	1	0	0	1
14276	0.14	19798	2	174	75	240	120	2	1	0	1	1
24399	0.14	15117	2	180	200	150	90	1	1	0	0	1
17058	0.13	21897	1	152	87	220	110	3	3	0	0	0
10556	0.13	21721	2	173	116	190	80	1	3	1	1	0

Figure 5.14: The EXTUNE interface: ① upload reference data, ② learn conformance constraints, ③ upload test data, ④ select the number of most non-conforming tuples to preview, ⑤ tuple-wise attribute-responsibility heat map, ⑥ aggregated attribute responsibility.

Datasets. We use four datasets for this evaluation: (1) *Cardiovascular Disease* [52] is a real-world dataset that contains information about cardiovascular patients with attributes such as height, weight, cholesterol level, glucose level, systolic and diastolic blood pressures, etc. (2) *Mobile Prices* [236] is a real-world dataset that contains information about mobile phones with attributes such as ram, battery power, talk time, etc. (3) *House Prices* [148] is a real-world dataset that contains information about houses for sale with attributes such as basement area, number of bathrooms, year built, etc. (4) *LED* (Light Emitting Diode) [40] is a synthetic benchmark. The dataset has a digit attribute, ranging from 0 to 9, 7 binary attributes—each representing one of the 7 LEDs relevant to the digit attribute—and 17 irrelevant binary attributes. This dataset includes gradual concept drift every 25,000 rows.

Case Studies. EXTUNE produces bar-charts of responsibility values as depicted in Figure 5.15. Figures 5.15(a), 5.15(b), and 5.15(c) show the explanation results for Cardiovascular Disease, Mobile Price, and House Price datasets, respectively. For the cardiovascular disease dataset, the training and serving sets consist of data for patients without and with

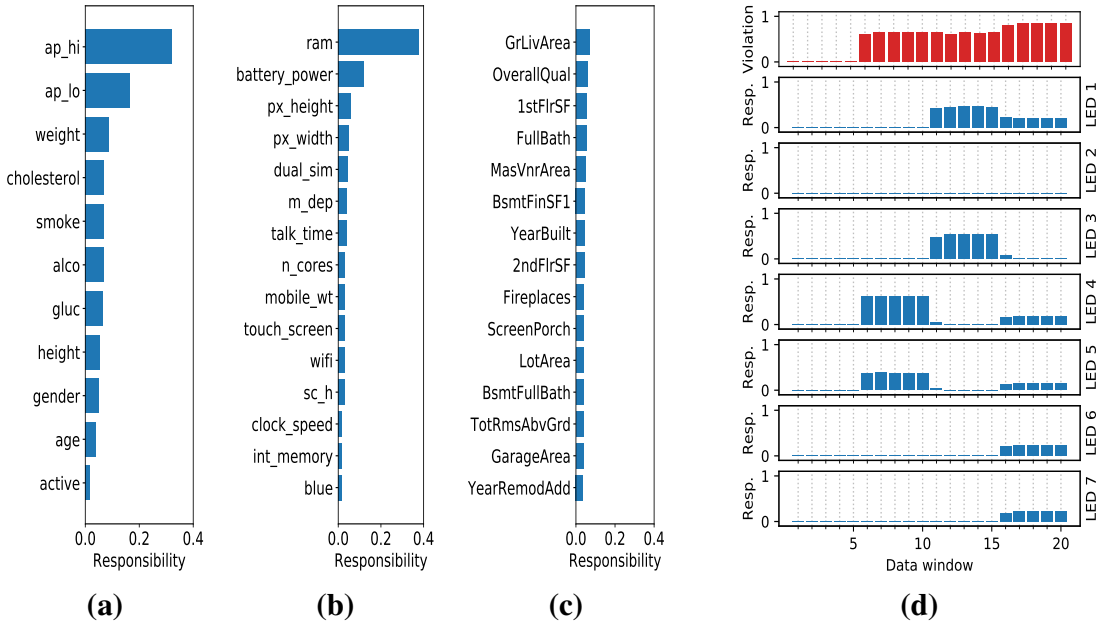


Figure 5.15: Responsibility assignment on attributes for drift on (a) Cardiovascular disease: trained on patients with no disease and served on patients with disease, (b) Mobile Prices: trained on cheap mobiles and served on expensive mobiles and (c) House Prices: trained on house with price $<=$ 100K and served on house with price $>=$ 300K. (d) Detection of drift on LED dataset. The dataset drifts every 5 windows (25,000 tuples). At each drift, a certain set of LEDs malfunction and take responsibility of the drift.

cardiovascular disease, respectively. For the House Price and Mobile Price datasets, the training and serving sets consist of houses and mobiles with prices below and above a certain threshold, respectively. As one can guess, we get many useful insights from the non-conformance responsibility bar-charts such as: "abnormal (high or low) blood pressure is a key cause for non-conformance of patients with cardiovascular disease w.r.t. normal people", "RAM is a distinguishing factor between expensive and cheap mobiles", "the reason for houses being expensive depends holistically on several attributes".

Figure 5.15(d) shows a similar result on the LED dataset. Instead of one serving set, we had 20 serving sets (the first set is also used as a training set to learn conformance constraints). We call each serving set a window where each window contains 5,000 tuples. This dataset introduces gradual concept drift every 25,000 rows (5 windows) by making a subset of LEDs malfunctioning. As one can clearly see, during the initial 5 windows, no

drift is observed. In the next 5 windows, LED 4 and LED 5 starts malfunctioning; in the next 5 windows, LED 1 and LED 3 starts malfunctioning, and so on.

5.8 Summary and Future Work

We introduced conformance constraints, and the notion of unsafe tuples for trusted machine learning. We presented an efficient and highly scalable approach for synthesizing conformance constraints; and demonstrated their effectiveness to tag unsafe tuples and quantify data drift. The experiments validate our theory and our principle of using low variance projections to generate effective conformance constraints.

We have studied only two use-cases from a large pool of potential applications using linear conformance constraints. An interesting extension is to explore more powerful non-linear conformance constraints that can be extracted from autoencoders. Another direction is to explore approaches to learn conformance constraints in a decision-tree-like structure where categorical attributes will guide the splitting conditions and leaves of the decision tree will contain simpler conformance constraints.

We have showcased how the tool CoCo can effectively learn useful conformance constraints according to the user's preferences and provide a natural way for data understanding and interactive data cleaning. We have also showcased how the tool ExTuNe can effectively detect non-conforming tuples and explain the causes of the observed non-conformance, and, thus, help users make decisions about (1) when to trust machine learning models and when not, and (2) how to enrich the training data towards building more robust models.

PART III: EXPLANATION FRAMEWORKS FOR DEBUGGING DATA SYSTEMS

CHAPTER 6

CAUSALITY-GUIDED ADAPTIVE INTERVENTIONAL DEBUGGING (AID)

Modern data management systems and database-backed applications run on commodity hardware and heavily rely on asynchronous and concurrent processing [70, 140, 241, 296]. As a result, they commonly experience runtime nondeterminism such as transient faults and variability in timing and thread scheduling. Unfortunately, software bugs related to handling nondeterminism are also common to these systems. Previous studies reported such bugs in MySQL [46, 219], PostgreSQL [218], NoSQL systems [201, 342], and database-backed applications [23], and showed that the bugs can cause crashes, unresponsiveness, and data corruptions. It is, therefore, crucial to identify and fix these bugs as early as possible.

Unfortunately, localizing root causes of intermittent failures is extremely challenging [211, 220, 350]. For example, concurrency bugs such as deadlocks, order and atomicity violation, race conditions, etc. may appear only under very specific thread interleavings. Even when an application executes with the same input in the same environment, these bugs may appear only rarely (e.g., in *flaky* unit tests [220]). When a concurrency bug is confirmed to exist, the debugging process is further complicated by the fact that the bug cannot be consistently reproduced. Heavy-weight techniques based on record-replay [16] and fine-grained tracing with lineage [12, 247] can provide insights on root causes after a bug manifests; but their runtime overheads often interfere with thread timing and scheduling, making it even harder for the intermittent bugs to manifest in the first place [195].

Statistical Debugging (SD) [167, 171, 205, 212] is a data-driven technique that partly addresses the above challenge. SD uses lightweight logging to capture an application's run-

time (mis)behaviors, called *predicates*. An example predicate indicates whether a method returns null in a particular execution or not. Given an application that intermittently fails, SD logs predicates from many successful and failed executions. SD then uses statistical analyses of the logs to identify *discriminative predicates* that are highly correlated with the failure.

SD has two key limitations. First, SD can produce many discriminative predicates that are correlated to, but not a true cause of, a failure. Second, SD does not provide enough insights that can explain how a predicate may eventually lead to the failure. Lack of such insights and the presence of many non-causal predicates make it hard for a developer to identify the true root cause of a failure. SD expects that a developer has sufficient domain knowledge about if/how a predicate can eventually cause a failure, even when the predicate is examined in isolation without additional context. This is often hard in practice, as is reported by real-world surveys [254].

Example 6.1. *To motivate our work, we consider a recently reported issue in Npgsql [245], an open-source ADO.NET data provider for PostgreSQL. On its GitHub repository, a user reported that a database application intermittently crashes when it tries to create a new PostgreSQL connection (GitHub issue #2485 [246]). The underlying root cause is a data race on an array index variable. The data race, which happens only when racing threads interleave in a specific way, causes one of the threads to access beyond the size of the array. This causes an exception that crashes the application.*

We used SD to localize the root cause of this nondeterministic bug (more details are in Section 6.6). SD identified 14 predicates, only three of which were causally related to the error. Other predicates were just symptoms of the root cause or happened to co-occur with the root cause.

In Section 6.6, we describe five other case studies that show the same general problem: SD produces too many predicates, only a small subset of which are causally related to the failure. Thus, SD is not specific enough, and it leaves the developer with the task

of identifying the root causes from a large number of candidates. This task is particularly challenging, since SD does not provide explanations of how a potential predicate can eventually lead to the failure.

We address these limitations with a new data-driven technique called *Adaptive Interventional Debugging* (AID). Given predicate logs from successful and failed executions of an application, AID can pinpoint *why* the application failed, by identifying one (or a small number of) predicate that indicates the real root cause (instead of producing a large number of potentially unrelated predicates). Moreover, AID can explain *how* the root cause leads to the failure, by automatically generating a causal chain of predicates linking the root cause, subsequent effects, and the failure. By doing so, AID enables a developer to quickly localize (and fix) the bug, even without deep knowledge about the application.

AID achieves the above by combining SD with causal analysis [227, 228, 229], fault injection [12, 134, 176], and group testing [151] in a novel way. Like SD, it starts by identifying discriminative predicates from successful and failed executions. In addition, AID uses temporal properties of the predicates to build an *approximate causal DAG* (Directed Acyclic Graph), which contains a superset of all true causal relationships among predicates. AID then progressively refines the DAG. In each round of refinement, AID uses ideas from adaptive group testing to carefully select a subset of predicates. Then, AID re-executes the application during which it *intervenes* (i.e., modifies application's behavior by e.g., injecting faults) application to fail or not, AID confirms or discards causal relationships in the approximate causal DAG, assuming counterfactual causality (C is a counterfactual cause of F iff F would not occur unless C occurs) and a single root cause. A sequence of interventions enables AID to identify the root cause and generate a *causal explanation path*, a sequence of causally related predicates that connect the root cause to the failure.

A key benefit of AID is its efficiency—it can identify root-cause and explanation predicates with significantly fewer rounds of interventions than adaptive group testing. In group testing, predicates are considered independent and hence each round selects *a random sub-*

set of predicates to intervene on and makes causality decisions about only those intervened predicates. In contrast, AID uses potential causality among predicates (in the approximate causal DAG). This enables AID to (1) make decisions not only about the intervened predicates, but also about other predicates; and (2) carefully select predicates whose intervention would maximize the effect of (1). Through theoretical and empirical analyses we show that this can significantly reduce the number of required interventions. This is an important benefit in practice since each round of intervention involves executing the application with fault injection and hence is time-consuming.

We evaluated AID on 3 open-source applications: Npgsql, Apache Kafka, Microsoft Azure Cosmos DB, and on 3 proprietary applications in Microsoft. We used known issues that cause these applications to intermittently fail even for same inputs. In each case, AID was able to identify the root cause of failure and generate an explanation that is consistent with the explanation provided by respective developers. Moreover, AID achieved this with significantly fewer interventions than traditional adaptive group testing. We also performed sensitivity analysis of AID with a set of synthetic workloads. The results show that AID requires fewer interventions than traditional adaptive group testing, and has significantly better worst-case performance than other variants.

In summary, we make the following contributions:

- We propose Adaptive Interventional Debugging (AID), a data-driven technique that localizes the root cause of an intermittent failure through a novel combination of statistical debugging, causal analysis, fault injection, and group testing (Section 6.1). AID provides significant benefits over the state-of-the-art Statistical Debugging (SD) techniques by (1) pinpointing the root cause of an application's failure and (2) generating an explanation of how the root cause triggers the failure (Sections 6.2–6.4). In contrast, SD techniques generate a large number of potential causes and without explaining how a potential cause may trigger the failure.

- We use information theoretic analysis to show that AID, by utilizing causal relationship among predicates, can converge to the true root cause and explanation significantly faster than traditional adaptive group testing (Section 6.5).

- We evaluate AID with six real-world applications that intermittently fail under specific inputs (Section 6.6). AID was able to identify the root causes and explain how the root causes triggered the failure, much faster than adaptive group testing and more precisely than SD. We also evaluate AID with many synthetically generated applications with known root causes and confirm that the benefits hold for them as well.

6.1 Background and Preliminaries

AID combines several existing techniques in a novel way. We now briefly review the techniques.

Statistical debugging. Statistical debugging (SD) aims to automatically pinpoint likely causes for an application's failure by statistically analyzing its execution logs from many successful and failed executions. It works by instrumenting an application to capture runtime *predicates* about the application's behavior. Examples of predicates include "the program takes the `false` branch at line 31", "the method `foo()` returns `null`", etc. Executing the instrumented application generates a sequence of predicate values, which we refer to as *predicate logs*. Without loss of generality, we assume that all predicates are Boolean.

Intuitively, the true root cause of the failure will cause certain predicates to be true only in the failed logs (or, only in the successful logs). Given logs from many successful executions and many failed executions of an application, SD aims to identify those *discriminative* predicates. Discriminative predicates encode program behaviors of failed executions that deviate from the ideal behaviors of the successful executions. Without loss of generality, we assume that discriminative predicates are `true` during failed executions. The predi-

cates can further be ranked based on their *precision* and *recall*, two well-known metrics that capture their discriminatory power.

$$precision(P) = \frac{\text{\#failed executions where } P \text{ is } \texttt{true}}{\text{\#executions where } P \text{ is } \texttt{true}}$$
$$recall(P) = \frac{\text{\#failed executions where } P \text{ is } \texttt{true}}{\text{\#failed executions}}$$

Causality. Informally, causality characterizes the relationship between an event and an outcome: the event is a cause if the outcome is a consequence of the event. There are several definitions of causality [132, 256]. In this work, we focus on *counterfactual* causes. According to *counterfactual causality*, C causes E iff E would not occur unless C occurs. Reasoning about causality frequently relies on a mechanism for *interventions* [142, 255, 301, 330], where one or more variables are forced to particular values, while the mechanisms controlling other variables remain unperturbed. Such interventions uncover counterfactual dependencies between variables.

Trivially, executing a program is a cause of its failure: if the program was not executed at the first place, the failure would not have occurred. However, our analysis targets *fully* discriminative predicates (with 100% precision and 100% recall), thereby eliminating such trivial predicates that are program invariants.

Fault injection. In software testing, *fault injection* [12, 134, 176, 224] is a technique to force an application, by instrumenting it or by manipulating the runtime environment, to execute a different code path than usual. We use the technique to *intervene* on (i.e., repair) discriminative predicates. Consider a method `ExecQuery()` that returns a result object in all successful executions and `null` in all failed executions. Then, the predicate "`ExecQuery()` returns `null`" is discriminative. The predicate can be intervened by forcing `ExecQuery()` to return the correct result object. Similarly, the predicate "there is a data race on X" can be intervened by delaying one access to X or by putting a lock around the code segments that access X to avoid simultaneous accesses to X.

Group testing. Given a set of discriminative predicates, a naïve approach to identify which predicates cause the failure is to intervene on one predicate at a time and observe if the intervention causes an execution to succeed. However, the number of required interventions is linear in number of predicates. *Group testing* reduces the number of interventions.

Group testing refers to the procedure that identifies certain items (e.g., defective) among a set of items while minimizing the number of *group tests* required. Formally, given a set $\mathcal{P}$ of N elements where D of them are defective, group testing performs k group tests, each on group $\mathcal{P}_i \subseteq \mathcal{P}$. Result of test on group $\mathcal{P}_i$ is positive if $\exists P \in \mathcal{P}_i$ s.t. P is defective, and negative otherwise. The objective is to minimize k, i.e., the number of group tests required. In our context, a group test is simultaneous intervention on a group of predicates, and the goal is to identify the predicates that cause the failure.

Two variations of group testing are studied in the literature: *adaptive* and *non-adaptive*. Our approach is based on adaptive group testing where the i-th group to test is decided *after* we observe the results of all $1 \leq j < i$ previous group tests. A trivial upper bound for adaptive group testing [151] is $\mathcal{O}(D \log N)$. A simple binary search algorithm can find each of the D defective items in at most $\log N$ group tests and hence a total of $D \log N$ group tests are sufficient to identify all defective items. Note that if $D \geq \frac{N}{\log N}$, then a linear strategy is preferable over any group testing scheme. Hence, we assume that $D < \frac{N}{\log N}$.

6.2 AID Overview

Adaptive Interventional Debugging (AID) targets applications (e.g., *flaky* tests [220]) that, even with the same inputs, intermittently fail due to various runtime nondeterminism such as thread scheduling and timing. Given predicate logs of successful and failed executions of an application, the goals of AID are to (1) identify *what* predicate actually causes the failure, and (2) generate an explanation of *how* the root cause leads to the failure (via a sequence of intermediate predicates). This is in contrast with traditional statistical

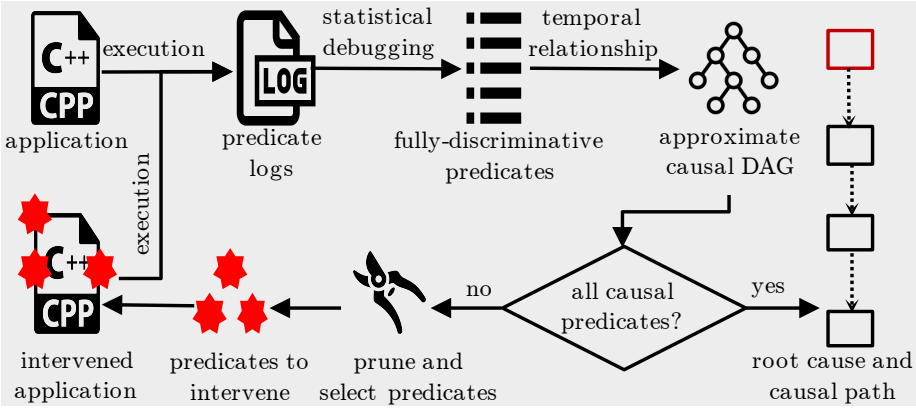


Figure 6.1: Adaptive Interventional Debugging workflow.

debugging, which generates a set of potential root-cause predicates (often a large number), without any explanation of how each potential root cause may lead to the failure.

Figure 6.1 shows an overview of AID. First, the framework employs standard SD techniques on predicate logs to identify a set of *fully* discriminative predicates, i.e., predicates that *always* appear in the failed executions and *never* appear in the successful executions. Then, AID uses the temporal relationships of predicates to infer *approximate causality*: if P_1 temporally precedes P_2 in *all* logs where they both appear, then P_1 *may* cause P_2. AID represents this approximate causality in a DAG called *Approximate Causal DAG* (AC-DAG), where predicates are nodes and edges indicate these possible causal relationships. We describe the AC-DAG in Section 6.3.

Based on its construction, the AC-DAG is guaranteed to contain all the true root-cause predicates and causal relationships among predicates. However, it may also contain additional predicates and edges that are not truly causal. The key insight of AID is that we can refine the AC-DAG and prune the non-causal nodes and edges through a sequence of interventions. To intervene on a predicate, AID changes the application's behavior through fault injection so that the predicate's value matches its value in successful executions. If the failure does not occur under the intervention, then, based on counterfactual causality, the predicate is guaranteed to be a root cause of the failure. Over several iterations,

AID intervenes on a set of carefully chosen predicates, refines the set of discriminative predicates, and prunes the AC-DAG, until it discovers the true root cause and the path that leads to the failure. We describe the intervention mechanism of AID in Section 6.4.

We now describe how AID adapts existing approaches in SD and fault injection for two of its core ideas: predicates and interventions.

6.2.1 AID Predicates

Predicate design. Similar to traditional SD techniques, AID is effective only if the initial set of predicates (in the predicate logs) contains a *root-cause predicate* that causes the failure. Predicate design is orthogonal to AID. We use predicates used by existing SD techniques, especially the ones used for finding root causes of concurrency bugs [167], a key reason behind intermittent failures [220]. Figure 6.2 shows examples of predicates in AID (column 1).

Predicate extraction. AID automatically instruments a target application to generate its *execution trace*. The trace contains each executed method's start and end time, its thread id, ids of objects it accesses, return values, whether it throws exception or not, and so on. This trace is then analyzed offline to evaluate a set of predicates at each execution point. This results in a sequence of predicates, called *predicate log*. The instrumented application is executed multiple times with the same input, to generate a set of predicate logs, each labeled as a successful or failed execution. Figure 6.2 shows the runtime conditions used to extract predicates (column 2).

Modeling nondeterminism. In practice, some predicates may cause a failure nondeterministically: two predicates A and B in *conjunction* cause a failure. AID does not consider such predicates since they are not fully discriminative (recall $< 100\%$). However, AID can still model these cases with *compound* predicates, adapted from state-of-the-art SD techniques [167], which model conjunctions. These compound predicates ("A and B") would

deterministically cause the failure and hence be fully discriminative. Note that AID focuses on counterfactual causality and thus does not support *disjunctive* root causes (as they are not counterfactual). In Section 6.4, we discuss AID's assumptions and their impact in practice.

6.2.2 AID Interventions

Intervention mechanism. AID uses an existing fault injection tool (similar to LFI [224]) to intervene on fully discriminative predicates; interventions change a predicate to match its value in a successful execution. In a way, AID's interventions try to locally "repair" a failed execution. Figure 6.2 shows examples of AID's interventions (column 3). Most of the interventions rely on changing timing and thread scheduling that can occur naturally by the underlying execution environment and runtime. More specifically, AID can slow down the execution of a method (by injecting delays), force or prevent concurrent execution of methods in different threads (by using synchronization primitives such as locks), change the execution order of concurrent threads (by injecting delays), etc. Such interventions can repair many concurrency bugs.

Validity of intervention. AID supports two additional intervention types, return-value alteration and exception-handling, which, in theory, can have undesirable runtime side-effects. Consider two predicates: (1) method `QueryAvgSalary` fails returning `null` and (2) method `UpdateSalary` fails returning `error`. AID can intervene to match their return values in successful executions, e.g., `50` and `OK`, respectively. The intervention on the first predicate does not modify any program state and, as the successful execution shows, the return value `50` can be safely used by the application. However, altering the return value of `UpdateSalary`, but not updating the salary, may not be sufficient intervention: other parts of the application that rely on the updated salary may fail. Inferring such side-effects is hard, if not impossible.

AID is restricted to safe interventions. It relies on developers to indicate which methods do not change (internal or external) application states and limits return-value interventions to only those methods (e.g., to `QueryAvgSalary`, but not to `UpdateSalary`). The same holds for exception-handling interventions. AID removes from predicate logs any predicates that cannot be safely intervened without undesirable side-effects. This ensures that the rest of the AID pipeline can safely intervene on any subset of predicates. Excluding some interventions may limit AID's precision, as it may eliminate a root-cause predicate. In such cases, AID may find another intervenable predicate that is causally related to the root cause, and is still useful for debugging. In our experiments (Section 6.6) we did not observe this issue, since the root-cause predicates were safe to intervene.

6.2.3 Program Instrumentation

AID separates program instrumentation and predicate extraction unlike prior SD techniques [167, 205, 212]. One advantage of our separation of instrumentation and predicate extraction is that it enables us to design predicates after collection of the application's execution traces. In contrast, prior works in SD instrument applications to directly extract the predicates. For example, to assess if two methods return the same value, prior work would instrument the program using a hard coded conditional statement "`pred = (foo() == bar())`". In contrast, our instrumentation simply collects the return values of the two methods and stores them in the execution trace. AID later evaluates the predicates based on the execution traces. This gives us the flexibility to design predicates post-execution, often based on knowledge of some domain-expert. For example, in this case, we can design multiple predicates such as whether two values are equal, unequal, or satisfy any custom relation.

Instrumentation granularity. Instrumentation granularity is orthogonal to AID. Like prior SD work, we could have instrumented at a finer granularity such as at each conditional branch; but instrumenting method calls were sufficient for our purpose. Since our

(1) Predicate	(2) Extraction condition	(3) Intervention mechanism
There is a data race involving methods M_1 and M_2	M_1 and M_2 temporally overlap accessing some object X while one of them is a `write`	Put locks around the code segments within M_1 and M_2 that access X
Method M fails	M throws an exception	Put M in a `try-catch` block
Method M runs too fast	M's duration is less than the minimum duration for M among all successful executions	Insert delay before M's `return` statement
Method M runs too slow	M's duration is greater than the maximum duration for M among all successful executions	Prematurely return from M the correct value that M returns in all successful executions
Method M returns incorrect value	M's return value $\neq x$, where x is the correct value returned by M in all successful executions	Alter M's `return` statement to force it to return the correct value x

Figure 6.2: Few example predicates, conditions used to extract them, and the corresponding interventions using fault injection.

instrumentation is of much sparser granularity than existing SD work [167, 205, 212] that employ sampling based finer granularity instrumentation, we do not use any sampling.

6.3 Approximating Causality

AID relies on traditional SD to derive a set of fully discriminative predicates. Using the logs of successful and failed executions, AID extracts temporal relationships among these predicates, and uses temporal precedence to approximate causality. It is clear that in the absence of feedback loops, a cause temporally precedes an effect [257]. To handle loops, AID considers multiple executions of the same program statement (e.g., within a loop, recursion, or multiple method calls) as separate instances, identified by their relative order of appearances during program execution, and maps them to separate predicates. This ensures that temporal precedence among predicates correctly *over-approximates* causality.

Approximate causal DAG. AID represents the approximation of causality in a DAG: each node represents a predicate, and an edge $P_1 \rightarrow P_2$ indicates that P_1 temporally precedes P_2 in *all* logs where both predicates appear. Figure 6.4(a) shows an example of the approximate causal DAG (AC-DAG). We use circles to explicitly depict *junctions* in

the AC-DAG; junctions are not themselves predicates, but denote splits or merges in the precedence ordering of predicates. Therefore, each predicate has in- and out-degrees of at most 1, while junctions have in- or out-degrees greater than 1. Note that, for clarity of visuals, in our depictions of the AC-DAG, we omit edges implied by transitive closure. For example, there exists an edge $P_3 \rightarrow P_5$, implied by $P_3 \rightarrow P_4$ and $P_4 \rightarrow P_5$, but it is not depicted. AID enforces an assumption of counterfactual causality by excluding from the AC-DAG any predicates that were not observed in *all* failed executions: if some executions failed without manifesting P, then P cannot be a cause of the failure.

Completeness of AC-DAG. The AC-DAG is complete with respect to the available, and safely intervenable, predicates: it contains all fully discriminative predicates that are safe to intervene, and if P_1 causes P_2, it includes the edge $P_1 \rightarrow P_2$. However, it may not be complete with respect to all possible true root causes, as a root cause may not always be represented by the available predicates (e.g., if the true root cause is a data race and no predicate is used to capture it). In such cases, AID will identify the (intervenable) predicate that is closest to the root cause and is causally related to the failure.

Since temporal precedence among predicates is a necessary condition for causality, the AC-DAG is guaranteed to contain the true causal relationships. However, temporal precedence is not sufficient for causality, and thus some edges in the AC-DAG may not be truly causal.

Temporal precedence. Capturing temporal precedence is not always straightforward. For simplicity of implementation, AID relies on computer clocks, which works reasonably well in practice. Relying on computer clocks is not always precise as the time gap between two events may be too small for the granularity of the clock; moreover, events may occur on different cores or machines whose clocks are not perfectly synchronized. These issues can be addressed with the use of logical clocks such as Lamport's Clock [196].

Another challenge is that some predicates are associated with time windows, rather than time points. The correct policy to resolve temporal precedence of two temporally overlapping predicates often depends on their semantics. However, the predicate types give important clues regarding the correct policy. In AID, predicate design involves specifying a set of rules that dictates the temporal precedence of two predicates. In constructing the AC-DAG, AID uses those rules.

For example, consider a scenario where `foo()` calls `bar()` and waits for `bar()` to end—so, `foo()` starts *before* but ends *after* `bar()`.

- (Case 1): Consider two predicates P_1: "`foo()` is running slow" and P_2: "`bar()` is running slow". Here, P_2 can cause P_1 but not the other way around. In this case, AID uses the policy that *end-time implies temporal precedence*.

- (Case 2): Now consider P_1: "`foo()` starts later than expected" and P_2 : "`bar()` starts later than expected". Here, P_1 can cause P_2 but not the other way around. Therefore, in this case, *start-time implies temporal precedence*.

AID works with any policy of deciding precedence, as long as it does not create cycles in the AC-DAG. Since temporal precedence is a necessary condition for causality, any conservative heuristic for deriving temporal precedence would work. A conservative heuristic may introduce more false positives (edges that are not truly causal), but those will be pruned by interventions (Section 6.4).

6.4 Causal Intervention

In this section, we describe AID's core component, which refines the AC-DAG through a series of *causal interventions*. An intervention on a predicate forces the predicate to a particular state; the execution of the application under the intervention asserts or contradicts the causal connection of the predicate with the failure, and AID prunes the AC-DAG accordingly. Interventions can be costly, as they require the application to be re-

Notation	Description
$\mathcal{G}$	Approximate causal DAG (AC-DAG)
$\mathbb{P}$	Causal path
F	Failure indicating predicate
P	A predicate
$\mathcal{P}$	Set of predicates
$P(r)$	Predicate P is observed in execution r
$\neg P(r)$	Predicate P is not observed in execution r
$P_1 \rightsquigarrow P_2$	There is a path from P_1 to P_2 in $\mathcal{G}$

Figure 6.3: Summary of notations used in Section 6.4.

executed. AID minimizes this cost by (1) smartly selecting the proper predicates to intervene, (2) grouping interventions that can be applied in a single application execution, and (3) aggressively pruning predicates even without direct intervention, but based on outcomes of other interventions. Figure 6.3 summarizes the notations used in this section.

We start by formalizing the problem of *causal path discovery* and state our assumptions (Section 6.4.1). Then we provide an illustrative example to show how AID works (Section 6.4.2). We proceed to describe *interventional pruning* that AID applies to aggressively prune predicates during group intervention rounds (Section 6.4.3). Then we present AID's causality-guided group intervention algorithm (Section 6.4.4) which administers group interventions to derive the causal path.

6.4.1 Problem Definition and Assumptions

Given an application that intermittently fails, our goal is to provide an informative explanation for the failure. To that end, given a set of fully discriminative predicates $\mathcal{P}$, we want to find an ordered subset of $\mathcal{P}$ that defines the causal path from the root-cause predicate to the predicate indicating the failure. Informally, AID finds a chain of predicates that starts from the root-cause predicate, ends at the failure predicate, and contains the maximal number of explanation predicates such that each is caused by the previous one in the

chain. We address the problem in a similar setting as SD, and make the following two assumptions:

Assumption 1 (Single Root-cause Predicate). The root cause of a failure is the predicate whose absence (i.e., a value of `false`) certainly avoids the failure, and there is no other predicate that causes the root cause. We assume that in all the failed executions, there is exactly one root-cause predicate.

This assumption is prevalent in the SD literature [167, 205, 212], and is supported by several studies on real-world concurrency bug characteristics [219, 316, 329], which show that a vast majority of root causes can be captured with reasonably simple single predicates and hence this assumption is very common in the SD literature [167, 205, 212]. Some notable findings include: (1) "97% of the non-deadlock concurrency bugs are covered by two simple patterns: *atomicity violation* and *order violation*" [219], (2) "66% of the non-deadlock concurrency bugs involve only one variable" [219] (3) "The manifestation of 96% of the concurrency bugs involves no more than two threads." [219], (4) "most fault localization approaches assume that each buggy source file has exactly one line of faulty code" [329], (5) "The majority of flaky test bugs occur when the test does not wait properly for asynchronous calls during the exercise phase of testing." [316], etc. In practice, even with specific inputs, a program may fail in multiple ways. However, failures by the same root cause generate a unique *failure signature* and hence can be grouped together using metadata (e.g., stack trace of the failure, location of the failure in the program binary, etc.) collected by failure trackers [120]. AID can then treat each group separately, targeting a single root cause for a specific failure. Moreover, the single-root-cause assumption is reasonable in many simpler settings such as unit tests that exercise small parts of an application.

Note that this assumption does not imply that the root cause consists of a single event; a predicate can be arbitrarily complex to capture multiple events. For example, the predicate "there is a data race on X" is `true` when two threads access the same shared memory X

at the same time, the accesses are not lock-protected, and one of the accesses is a write operation. Whether a single predicate is sufficient to capture the root cause depends on predicate design, which is orthogonal to AID. AID adapts the state-of-the art predicate design, tailored to capture root causes of concurrency bugs [167], which is sophisticated enough to capture all common root causes using single predicates. If no single predicate captures the true root cause, AID still finds the predicate closest to the true root cause in the true causal path.

Assumption 2 (Deterministic Effect). A root-cause predicate, if triggered, causes a fixed sequence of intermediate predicates (i.e., effects) before eventually causing the failure. We call this sequence *causal path*, and we assume that there is a unique one for each root-cause-failure pair.

Prior work has considered, and shown evidence of, a unique causal path between a root cause and the failure in sequential applications [170, 304]. The unique causal path assumption is likely to hold in concurrent applications as well for two key reasons. First, the predicates in AID's causal path may remain unchanged, despite nondeterminism in the underlying instruction sequence. For example, the predicate "there is a data race between methods X and Y" is not affected by which method starts first, as long as they temporally overlap. Second, AID only considers fully discriminative predicates. If such predicates exist to capture the root cause and its effects, by the definition of being fully discriminative, there will be a unique causal path (of predicates) from the root cause to the failure. In all six of our real-world case studies (Section 6.6), such predicates existed and there were unique causal paths from the root causes to the failures.

Note that it is possible to observe some degree of disjointness within the true causal paths. For example, consider a case where the root cause C triggers the failure F in two ways: in some failed executions, the causal path is $C \rightarrow A_1 \rightarrow B \rightarrow F$ and, for others, $C \rightarrow A_2 \rightarrow B \rightarrow F$. This implies that neither A_1 nor A_2 is fully discriminative. Since AID only considers fully discriminative predicates, both of them are excluded from the AC-

DAG. In this case, AID reports $C \rightarrow B \rightarrow F$ as the causal path; this is the shared part of the two causal paths, which includes all counterfactual predicates and omits any disjunctive predicates. One could potentially relax this assumption by encoding the interaction of such predicates through a fully discriminative predicate (e.g., $A = A_1 \vee A_2$ encodes disjunction and is fully discriminative).

Based on these assumptions, we define the causal path discovery problem formally as follows.

Definition 6.1 (Causal Path Discovery)**.** *Given an approximate causal DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a predicate $F \in \mathcal{V}$ indicating a specific failure, the **causal path discovery** problem seeks a path $\mathbb{P} = \langle C_0, C_1, \ldots, C_n \rangle$ such that the following conditions hold:*

- *C_0 is the root cause of the failure and $C_n = F$.*
- *$\forall\ 0 \leq i \leq n$, $C_i \in \mathcal{V}$ and $\forall\ 0 \leq i < n$, $(C_i, C_{i+1}) \in \mathcal{E}$.*
- *$\forall\ 0 \leq i < j \leq n$, C_i is a counterfactual cause of C_j.*
- *$|\mathbb{P}|$ is maximized.*

6.4.2 Illustrative Example

AID performs causal path discovery through an intervention algorithm (Section 6.4.4). Here, we illustrate the main steps and intuitions through an example.

Figure 6.4(a) shows an AC-DAG derived by AID (Section 6.3). The AC-DAG contains all edges implied by transitive closure, but we do not depict them to have clearer visuals. The true causal path for the failure F is $P1 \rightarrow P2 \rightarrow P11 \rightarrow F$, depicted with dashed red outline. The AC-DAG is a superset of the actual causal graph, which is shown in Figure 6.4(b).

AID follows an intervention-centric approach for discovering the causal path. Intervening on a predicate *forces* it to behave the way it does in the successful executions, which

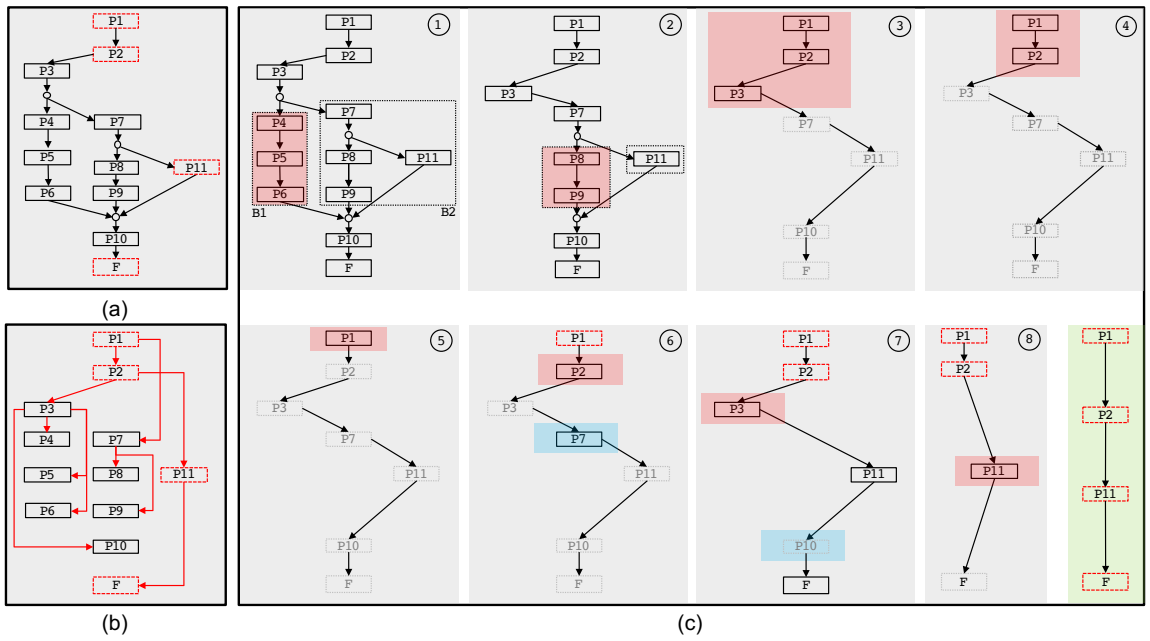


Figure 6.4: (a) AC-DAG as constructed by AID. The DAG includes all edges implied by transitive closure, but we omit them for clarity of the visuals. We indicate the predicates in the causal path with the dashed red outline. (b) The actual causal DAG is a subgraph of the AC-DAG. (c) Step by step illustration to discover the causal path (shown at bottom right). Steps ① and ② perform branch pruning, steps ③–⑧ perform group intervention with pruning on the predicate chain, steps ⑥ and ⑦ apply interventional pruning.

is by definition, the opposite of the failed executions. (Recall that, without loss of generality, we assume that all predicates are boolean.) Following the adaptive group testing paradigm, AID performs group intervention, which is simultaneous intervention on a set of predicates, to reduce the total number of interventions. Figure 6.4(c) shows the steps of the intervention algorithm, numbered ①–⑧.

AID first aims to reduce the AC-DAG by pruning entire chains that are not associated with the failure, through a process called *branch pruning* (Section 6.4.4). Starting from the root of the AC-DAG, AID discovers the first junction, after predicate $P3$. For each child of a junction, AID creates a compound predicate, called an *independent branch*, or simply *branch*, that is a disjunction over the child and all its descendants that are not descendants of the other children. So, for the junction after $P3$, we get branches $B1 =$

$P4 \vee P5 \vee P6$ and $B2 = P7 \vee P8 \vee P9 \vee P11$. AID intervenes on one of the branches chosen at random—in this case $B1$—at step ①; this requires an intervention on all of its disjunctive predicates ($P4$, $P5$, and $P6$) in order to make the branch predicate `False`. Despite the intervention, the program continues to fail, and AID prunes the entire branch of $B1$, resolving the junction after $P3$. For a junction of B branches, AID would need $\log B$ interventions to resolve it using a divide-and-conquer approach. At step ②, AID similarly prunes a branch at the junction after $P7$. At this point, AID is done with branch pruning since it is left with just a chain of predicates (step ③).

What is left for AID is to prune any non-causal predicate from the remaining chain. AID achieves that through a divide-and-conquer strategy that intervenes on groups of predicates at a time (Algorithm 3). It intervenes on the top half of the chain—$\{P1, P2, P3\}$—which stops the failure and confirms that the root cause is in this group (step ③). With two more steps that narrow down the interventions (steps ④ and ⑤), AID discovers that $P1$ is the root cause. Note that we cannot simply assume that the root of the AC-DAG is a cause, because the edges are not all necessarily causal.

After the discovery of the root cause, AID needs to derive the causal path. Continuing the divide-and-conquer steps, it intervenes on $P2$ (step ⑥). This stops the failure, confirming that $P2$ is in the causal path. In addition, since $P7$ is not causally dependent on $P2$, the intervention on $P2$ does not stop $P7$ from occurring. This observation allows AID to prune $P7$ without intervening on it directly. At step ⑦, AID intervenes on $P3$. The effect of this intervention is that the failure is still observed, but $P10$ no longer occurs, indicating that $P10$ is causally connected to $P3$, but not to the failure; this allows AID to prune both $P3$ and $P10$. Finally, at step ⑧, AID intervenes on $P11$ and confirms that it is causal, completing the causal path derivation. AID discovered the causal path in 8 interventions, while naïvely we would have needed 11—one for each predicate.

6.4.3 Predicate Pruning

In the initial construction of the AC-DAG, AID excludes predicates based on a simple rule: a predicate P is excluded if there exists a program execution r, where P occurs and the failure does not ($P(r) \wedge \neg F(r)$), or P does not occur and the failure does ($\neg P(r) \wedge F(r)$). Intervening executions follow the same basic intuition for pruning the intervened predicate C: By definition C does not occur in an execution r_C that intervenes on predicate C ($\neg C(r_C)$); thus, if the failure still occurs on r_C ($F(r_C)$), then C is pruned from the AC-DAG.

As we saw in the illustrative example, intervention on a predicate C may also lead to the pruning of additional predicates. However, the same basic pruning logic needs to be applied more carefully in this case. In particular, we can never prune predicates that precede C in the AC-DAG, as their potential causal effect on the failure may be muted by the intervention on C. Thus, we can only apply the pruning rule to any predicate X that is not an ancestor of C in the AC-DAG ($X \not\rightsquigarrow C$). We formalize the predicate pruning strategy over $\mathcal{G}(\mathcal{V}, \mathcal{E})$ in the following definition.

Definition 6.2 (Interventional Pruning)**.** *Let $R_{\mathcal{C}}$ be a set of program executions*[1] *intervening on a group of predicates $\mathcal{C} \subseteq \mathcal{V}$. Every $C \in \mathcal{C}$ is pruned from $\mathcal{G}$ iff $\exists r \in R_{\mathcal{C}}$ such that $F(r)$. Any other predicate $P \notin \mathcal{C}$ is pruned from $\mathcal{G}$ iff $\nexists C \in \mathcal{C}$ such that $P \rightsquigarrow C$ and $\exists r \in R_{\mathcal{C}}$ such that $(P(r) \wedge \neg F(r)) \vee (\neg P(r) \wedge F(r))$.*

6.4.4 Causality-guided Intervention

AID's core intervention method is described in Algorithm 3: Group Intervention With Pruning (GIWP). GIWP applies adaptive group testing to derive causal and spurious (non-causal) nodes in the AC-DAG. The algorithm applies a divide-and-conquer approach that groups predicates based on their topological order (a linear ordering of its nodes such that

[1] Because of nondeterminism issues in concurrent applications, we execute a program multiple times with the same intervention. However, it is sufficient to identify a single counter-example execution to invoke the pruning rule.

Algorithm 3: GIWP $(\mathcal{P}, \mathcal{G}, F)$

Input : A set of candidate predicates, $\mathcal{P}$,
AC-DAG, $\mathcal{G}$
Failure indicating predicate, F

Output : The set of counterfactual causes of F, $\mathcal{C}$
The set of spurious predicates, $\mathcal{X}$

1 $\mathcal{C} = \emptyset$ /* causal predicate set */
2 $\mathcal{X} = \emptyset$ /* spurious predicate set */
3 **while** $\mathcal{P} \neq \emptyset$ **do**
4 $\mathcal{P}_1$ = first half of $\mathcal{P}$ in topological order
5 $R_{\mathcal{P}_1}$ = Intervene ($\mathcal{P}_1$)
6 **if** $\nexists r \in R_{\mathcal{P}_1}$ *s.t.* $F(r)$ **then** /* failure stopped */
7 **if** $\mathcal{P}_1$ *contains a single predicate* **then**
8 $\mathcal{C} = \mathcal{C} \cup \mathcal{P}_1$ /* a cause is confirmed */
9 **else** /* need to confirm causes */
10 $\mathcal{C}', \mathcal{X}' =$ **GIWP**$(\mathcal{P}_1, \mathcal{G}, F)$
11 $\mathcal{C} = \mathcal{C} \cup \mathcal{C}'$ /* confirmed causes */
12 $\mathcal{X} = \mathcal{X} \cup \mathcal{X}'$ /* spurious predicates */
/* interventional pruning */
13 **if** $\exists r \in R_{\mathcal{P}_1}$ *s.t.* $F(r)$ **then** /* failure didn't stop */
14 $\mathcal{X} = \mathcal{X} \cup \mathcal{P}_1$ /* pruning */
15 **foreach** $P \in \mathcal{P} - \mathcal{P}_1$ *s.t.* $\forall P' \in \mathcal{P}_1 \;\; P \not\rightsquigarrow P'$ **do**
16 **if** $\exists r \in R_{\mathcal{P}_1}$ *s.t.* $(P(r) \wedge \neg F(r)) \vee (\neg P(r) \wedge F(r))$ **then**
17 $\mathcal{X} = \mathcal{X} \cup \{P\}$ /* pruning */
18 $\mathcal{P} = \mathcal{P} - (\mathcal{C} \cup \mathcal{X})$ /* remove confirmed and spurious predicates from candidate predicate pool */
19 **return** $\mathcal{C}$, $\mathcal{X}$

for every directed edge $P_1 \rightarrow P2$, P_1 comes before P_2 in the ordering). In every iteration, GIWP selects the predicates in the lowest half of the topological order, resolving ties randomly, and intervenes by setting all of them to `False` (lines 4–5). The intervention returns a set of predicate logs.

If the failure is not observed in any of the intervening executions (line 6), based on counterfactual causality, GIWP concludes that the intervened group contains at least one predicate that causes the failure. If the group contains a single predicate, it is marked as causal (line 8). Otherwise, GIWP recurses to trace the causal predicates within the group (line 10).

Algorithm 4: Branch-Prune ($\mathcal{G}, F$)

Input : AC-DAG, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Failure indicating predicate, F
Output : Reduces $\mathcal{G}$ to an approximate causal chain

1 $\mathcal{C} = \emptyset$ /* potential causal predicate set */
2 $\mathcal{X} = \emptyset$ /* spurious predicate set */
3 **while** $\mathcal{V} - \mathcal{C} \neq \emptyset$ **do**
4 $\mathcal{P}$ = predicates at the lowest topological level in $\mathcal{V} - \mathcal{C}$
5 **if** $\mathcal{P}$ *contains a single predicate* **then**
6 $\mathcal{C} = \mathcal{C} \cup \mathcal{P}$ /* add to potential causal set */
7 **else** /* this is a junction */
8 $\mathcal{B} = \emptyset$
9 **foreach** $P \in \mathcal{P}$ **do**
10 $\mathcal{B}_P = \bigvee\{Q : P \rightsquigarrow Q \wedge \forall P' \in \mathcal{P} - \{P\}\ P' \not\rightsquigarrow Q\}$
11 $\mathcal{B}_P = P \vee \mathcal{B}_P$
12 $\mathcal{B} = \mathcal{B} \cup \{\mathcal{B}_P\}$ /* set of branches */
13 $\mathcal{C}', \mathcal{X}' =$ **GIWP** $(\mathcal{B}, \mathcal{G}, F)$
14 $\mathcal{C} = \mathcal{C} \cup \mathcal{C}'$ /* add to potential causal set */
15 $\mathcal{X} = \mathcal{X} \cup \mathcal{X}'$ /* add to spurious set */
/* refining $\mathcal{G}$ */
16 $\mathcal{U} = \{U : \mathcal{C} \neq \emptyset \wedge \forall C \in \mathcal{C}\ C \not\rightsquigarrow U\}$ /* unreachable */
17 $\mathcal{V} = \mathcal{V} - \mathcal{X}$ /* remove spurious predicates */
18 $\mathcal{V} = \mathcal{V} - \mathcal{U}$ /* remove unreachable predicates */

During each intervention round, GIWP applies Definition 6.2 to prune predicates that are determined to be non-causal (lines 13–17). First, if the algorithm discovers an intervening execution that still exhibits the failure, then it labels all intervened predicates as spurious and marks them for removal (line 14). Second, GIWP examines each other predicate that does not precede any intervened predicate and observes if any of the intervened executions demonstrate a counterfactual violation between the predicate and the failure. If a violation is found, that predicate is pruned (line 17).

At completion of each intervention round, GIWP refines the predicate pool by eliminating all confirmed causes and spurious predicates (line 18) and enters the next intervention round . It continues the interventions until all predicates are either marked as causal or spurious and the remaining predicate pool is empty. Finally, GIWP returns two disjoint predicate sets—the causal predicates and the spurious predicates (line 19).

Algorithm 5: Causal-Path-Discovery ($\mathcal{G}, F, Flag_B$)

Input : AC-DAG, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Failure indicating predicate, F
$Flag_B$, whether to apply branch pruning or not
Output : A causal path

1 **if** $Flag_B$ **then**
2 **Branch-Prune** ($\mathcal{G}, F$)
3 $\mathcal{C}, \mathcal{X} =$ **GIWP** ($\mathcal{V}-\{F\}, \mathcal{G}, F$)
4 **return** $\mathcal{C}$

6.4.4.1 Branch Pruning

GIWP is sufficient for most practical applications and can work directly on the AC-DAG. However, when the AC-DAG satisfies certain conditions (analyzed in Section 6.5.3.1), we can reduce the number of required interventions through a process called *branch pruning*. The intuition is that since there is a single causal path that explains the failure, junctions (where multiple paths exist) can be used to quickly identify independent branches to be pruned or confirmed as causal as a group. The branches can be used to more effectively identify groups for intervention, reducing the overall number of required interventions.

Branch pruning iteratively prunes branches at junctions (steps ① and ② in the illustrative example) to reduce the AC-DAG to a chain of predicates. The process is detailed in Algorithm 4. The algorithm traverses the DAG based on its topological order, and does not intervene while it encounters a single node at a time, which means it is still in a chain (line 5). When it encounters multiple nodes at the same topological level, it means it encountered a junction (line 7). A junction means that the true causal path can only continue in one direction, and AID can perform group intervention to discover it. The algorithm invokes GIWP to perform this intervention over a set of special predicates constructed from the branches at the encountered junction (lines 10–12). A branch at predicate P is defined as a disjunctive predicate over P and all descendants of P that are not descendants of any other predicate at the same topological level as P. An example branch from our illustrative

example is $B1 = P4 \vee P5 \vee P6$. To intervene on a branch, one has to intervene on all of its disjunctive predicates. The algorithm defines $\mathcal{B}$ as the union of all branches, which corresponds to a completely disconnected graph (no edges between the nodes), thus all branch predicates are at the same topological level. GIWP is then invoked (line 13) to identify the causal branch. The algorithm removes any predicate that is not causally connected to the failure (line 17) or is no longer reachable from the correct causal chain (line 18), and updates the AC-DAG accordingly. At the completion of branch pruning, AID reduces the AC-DAG to simple chain of predicates.

Finally, Algorithm 5 presents the overall method that AID uses to perform causal path discovery, which optionally invokes branch pruning before the divide-and-conquer group intervention through GIWP.

6.5 Theoretical Analysis

In this section, we theoretically analyze the performance of AID in terms of the number of interventions required to identify all *causal predicates*, which are the predicates causally related to the failure.[2] Similar to the analysis of group testing algorithms, we study the information-theoretic lower bound, which specifies the minimum number of bits of information that an algorithm must extract to identify all causal predicates for any instance of a problem. We also study the lower and the upper bounds that quantify the minimum and the maximum number of group interventions required to identify all causal predicates, respectively, for AID versus a Traditional Adaptive Group Testing (TAGT) algorithm.

Any group testing algorithm takes N items (predicates), D of which are faulty (causal), and aims to identify all faulty items using as few group interventions as possible. Since there are $\binom{N}{D}$ possible outcomes, the information-theoretic lower bound for this problem is $\log \binom{N}{D}$. The upper bound on the number of interventions using TAGT is $\mathcal{O}(D \log N)$,

[2]Causal predicates correspond to *faulty predicates* in group testing. This distinction in terminology is because group testing does not meaningfully reason about causality.

since $\log N$ group interventions are sufficient to reveal each causal predicate. Here, we assume $D < \frac{N}{\log N}$; otherwise, a linear approach that intervenes on one predicate at a time is preferable.

We now show that the Causal Path Discovery (CPD) problem (Definition 6.1) can reduce the lower bound on the number of required interventions compared to Group Testing (GT). We also show that the upper bound on the number of interventions is lower for AID than TAGT, because of the two assumptions of CPD (Section 6.4.1). In TAGT, predicates are assumed to be independent of each other, and hence, after each intervention, decisions (about whether predicates are causal) can be made only about the intervened predicates. In contrast, AID uses the precedence relationships among predicates in the AC-DAG to (1) aggressively prune, by making decisions not only about the intervened predicates but also about other predicates, and to (2) select predicates based on the topological order, which enables effective pruning during each intervention.

Example 6.2. *Consider the AC-DAG of Figure 6.5(a), consisting of $N = 6$ predicates and the failure predicate F. If* AID *intervenes on all predicates in one branch (e.g., $\{A_1, B_1, C_1\}$) and finds causal connection to the failure, it can avoid intervening on predicates in the other branch according to the deterministic effect assumption.* AID *can also use the structure of the AC-DAG to intervene on A_1 (or A_2) before other predicates since the intervention can prune a large set of predicates. Since GT algorithms do not assume relationships among predicates, they can only intervene on predicates in random order and can make decisions about only the intervened predicates.*

6.5.1 Search Space

The temporal precedence and potential causality encoded in the AC-DAG restrict the possible causal paths and significantly reduce the search space of CPD compared to GT.

Example 6.3. *In the example of Figure 6.5(a), GT considers all subsets of the 6 predicates as possible solutions, and thus its search space includes $2^6 = 64$ candidates. CPD lever-*

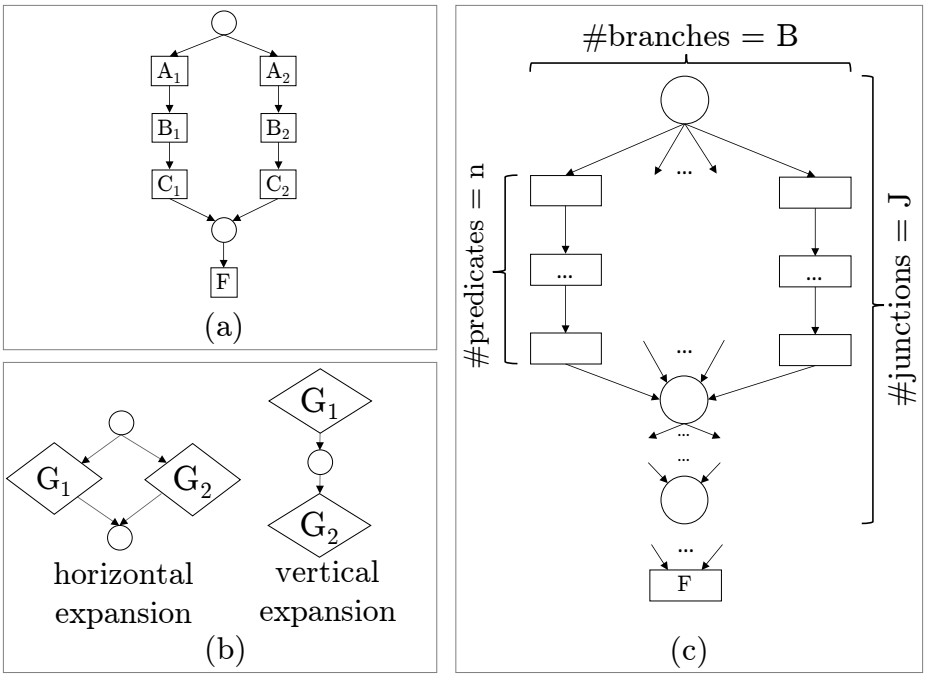


Figure 6.5: (a) An AC-DAG with failure predicate F. (b) Horizontal and vertical expansion. (c) A symmetric AC-DAG with J junctions where each junction has B branches and each branch has n predicates.

ages the AC-DAG and the deterministic effect assumption (Section 6.4.1) to identify invalid candidates and reduce the search space considerably. For example, the candidate solution $\{A_1, B_2, C_1\}$ *is not possible under CPD, because it involves predicates in separate paths on the AC-DAG. In fact, based on the AC-DAG, CPD does not need to explore any solutions with more than 3 predicates. The complete search space of CPD includes all subsets of predicates along each branch of length 3, thus a total of* $2 \cdot (2^3 - 1) + 1 = 15$ *possible solutions.*

We proceed to characterize the search space of CPD compared to GT more generally. We use $|G|$ to denote the number of predicates in an AC-DAG represented by G, and W_G^{GT} and W_G^{CPD} to denote the size of the search space for GT and CPD, respectively. We start from the simplest case of DAG, a chain, and then using the notions of horizontal and vertical expansion, we can derive the search space for any DAG.

If G is a simple chain of predicates, then GT and CPD have the same search space: $2^{|G|}$. CPD reduces the search space drastically when junctions split the predicates into separate branches, like in Example 6.3. We call this case a *horizontal expansion*: a DAG $\mathcal{G}_H$ is a horizontal expansion of two subgraphs G_1 and G_2 if it connects them in parallel through two junctions, at the roots (lowest topological level) and leaves (highest topological level). In contrast, $\mathcal{G}_V$ is a *vertical expansion*, if it connects them sequentially via a junction. Horizontal and vertical expansion are depicted in Figure 6.5(b). In horizontal expansion, the search space of CPD is additive over the combined DAGs, while in vertical expansion it is multiplicative.

Lemma 6.1 (DAG expansion)**.** *Let $W_{G_1}^{CPD}$ and $W_{G_2}^{CPD}$ be the numbers of valid solutions for CPD over DAGs G_1 and G_2, respectively. Let $\mathcal{G}_H$ and $\mathcal{G}_V$ represent their horizontal and vertical expansion, respectively. Then:*

$$W_{\mathcal{G}_H}^{CPD} = 1 + (W_{G_1}^{CPD} - 1) + (W_{G_2}^{CPD} - 1)$$
$$W_{\mathcal{G}_V}^{CPD} = W_{G_1}^{CPD} W_{G_2}^{CPD}$$

In contrast, in both cases, the search space of GT is $2^{|G_1|+|G_2|}$.

Intuitively, in horizontal expansion, the valid solutions for $\mathcal{G}_H$ are those of G_1 and those from G_2, but no combinations between the two are possible. Note that both $W_{G_1}^{CPD}$ and $W_{G_2}^{CPD}$ have the empty set as a common solution, so in the computation of $W_{\mathcal{G}_H}^{CPD}$, one solution is subtracted from each search space ($W_{G_i}^{CPD} - 1$) and then added to the overall result.

Symmetric AC-DAG. Lemma 6.1 allows us to derive the size of the search space for CPD over any AC-DAG. To further highlight the difference between GT and CPD, we analyze their search space over a special type of AC-DAG, a symmetric AC-DAG, depicted in Figure 6.5(c). A symmetric AC-DAG has J junctions, and B branches at each junction, where each branch is a simple chain of n predicates. Therefore, the total number

of predicates in the symmetric AC-DAG is $N = JBn$, and the search space of GT is $W^{GT} = 2^{JBn}$. For CPD, based on horizontal expansion, the subgraph in-between two subsequent junctions has a total of $1+\sum_i^B(2^n-1) = 1+B(2^n-1)$ candidate solutions. Then, based on vertical expansion, the overall search space of CPD is:

$$W^{CPD} = (B(2^n-1)+1)^J$$

6.5.2 Lower Bound of Number of Interventions

We now show that, due to the predicate pruning mechanisms, and the strategy of picking predicates according to topological order, the lower bound[3] on the required number of interventions in CPD is significantly reduced. For the sake of simplicity, we drop the deterministic effect assumption in this analysis. In GT, after each group test, we get at least 1 bit of information. Since after retrieving all information, the remaining information should be ≤ 0, therefore, the number of required interventions in GT is bounded below by $\log\binom{N}{D}$. In contrast, for CPD, we have the following theorem.

Theorem 6.1. *The number of required group interventions in CPD is bounded below by* $\frac{N}{N+DS_1}\log\binom{N}{D}$*, where at least* S_1 *predicates are discarded (either pruned using the pruning rule or marked as causal) during each group intervention (proof is in Appendix C.1).*

Since $\frac{DS_1}{N} > 0$, we obtain a reduced lower bound for the number of required interventions in CPD than GT. In general, as S_1 increases, the lower bound in CPD decreases. Note that we are not claiming that AID achieves this lower bound for CPD; but this sets the possibility that improved algorithms can be designed in the setting of CPD than GT.

[3]Lower bound is a theoretical bound which states that, it might be possible to design an algorithm that can solve the problem which requires number of steps equal to the lower bound. Note that, this does not imply that there exists one such algorithm.

	Search space	#Interventions	
		Lower bound	Upper bound (AID/TAGT)
CPD	$(B(2^n-1)+1)^J$	$\frac{JBn}{JBn+DS_1}\log\binom{JBn}{D}$	$J\log B+D\log(Jn)-\frac{D(D-1)S_2}{2Jn}$
GT	2^{JBn}	$\log\binom{JBn}{D}$	$D\log B+D\log(Jn)-\frac{D(D-1)}{2JBn}$

Figure 6.6: Theoretical comparison between CPD and GT for the symmetric AC-DAG of Figure 6.5(c).

Symmetric AC-DAG. Figure 6.6 shows the lower bound on the number of required interventions in CPD and GT for the symmetric AC-DAG of Figure 6.5(c), assuming that each intervention discards at least S_1 predicates in CPD.

6.5.3 Upper Bound of Number of Interventions

We now analyze the upper bound on the number of interventions for AID under (1) branch pruning, which exploits the deterministic effect assumption, and (2) predicate pruning.

6.5.3.1 Branch Pruning

Whenever AID encounters a junction, it has the option to apply branch pruning. In CPD, at most one branch can be causal at each junction; hence, we can find the causal branch using $\log B$ interventions at each junction, where B is the number of branches at that junction. Also, B is upper-bounded by the number of threads T in the program. This holds since we assume that the program inputs are fixed and there is no different conditional branching due to input variation in different failed executions within the same thread. If there are J junctions and at most T branches at each junction, the number of interventions required to reduce the AC-DAG to a chain is at most $J\log T$. Now let us assume that the maximum number of predicates in any path in the AC-DAG is N_M. Therefore, the chain found after branch pruning can contain at most N_M predicates. If D of them are causal predicates, we need at most $D\log N_M$ interventions to find them. Therefore, the total

number of required interventions for AID is $\leq J \log T + D \log N_M$. In contrast, the number of required interventions for TAGT, which does not prune branches, is $\leq D \log(TN_M) = D \log T + D \log N_M$. Therefore, whenever $J < D$, the upper bound on the number of interventions for AID is smaller than the upper bound for TAGT.

6.5.3.2 Predicate Pruning

For an AC-DAG with N predicates, D of which are causal, we now focus on the upper bound on the number of interventions in AID using only predicate pruning. In the worst case, when no pruning is possible, the number of required interventions would be the same as that of TAGT without pruning, i.e., $\mathcal{O}(D \log N)$.

Theorem 6.2. *If at least S_2 predicates are discarded (pruned or marked as causal) from the candidate predicate pool during each causal predicate discovery, then the number of required interventions for* AID *is $\leq D \log N - \frac{D(D-1)S_2}{2N}$ (proof is in Appendix C.2).*

Hence, the reduction depends on S_2. When $S_2 = 1$, we are referring to TAGT, in absence of pruning, because once TAGT finds a causal predicate, it removes that predicate from the candidate predicate pool.

Symmetric AC-DAG. Figure 6.6 shows the upper bound on the number of required interventions using AID and TAGT for the symmetric AC-DAG of Figure 6.5(c), assuming that at least S_2 predicates are discarded during each causal predicate discovery by AID.

6.6 Experimental Evaluation

We now empirically evaluate AID. We first use AID on six real-world applications to demonstrate its effectiveness in identifying root cause and generating explanation on how the root cause causes the failure. Then we use a synthetic benchmark to compare AID and its variants against traditional adaptive group testing approach to do a sensitivity analysis of AID on various parameters of the benchmark.

6.6.1 Case Studies of Real-world Applications

We now use three real-world open-source applications and three proprietary applications to demonstrate AID's effectiveness in identifying root causes of transient failures. Figure 6.7 summarizes the results and highlights the key benefits of AID:

- AID is able to identify the true root cause and generate an explanation that is consistent with the explanation provided by the developers in corresponding GitHub issues.
- AID requires significantly fewer interventions than traditional adaptive group testing (TAGT), which does not utilize causality among predicates (columns 5 and 6).
- In contrast, SD generates a large number of discriminative predicates (column 3), only a small number of which is actually causally related to the failures (column 4).

6.6.1.1 Data Race in Npgsql

As a case study, we first consider a recently discovered concurrency bug in Npgsql [245], an open-source ADO.NET Data Provider for PostgreSQL. The bug (GitHub issue #2485) causes an Npgsql-baked application to intermittently crash when it tries to create a new PostgreSQL connection. We use AID to check if it can identify the root cause and generate an explanation of how the root cause triggers the failure.

We used one of the existing unit tests in Npgsql that causes the issue, and generated logs from 50 successful executions and 50 failed executions of the test. By applying SD, we found a total of 14 discriminative predicates. However, SD did not pinpoint the root cause or generate any explanation.

We then applied AID on the discriminative predicates. In the branch pruning step, it used 3 rounds of interventions to prune 8 of the 14 predicates. In the next step, it required 2 more rounds of interventions. Overall, AID required a total of 5 intervention rounds; in contrast, TAGT would require 11 interventions in the worst case.

After all the interventions, AID identified a data race as the root cause of the failure and produced the following explanation: (1) two threads race on an index variable:

(1) Application	(2) GitHub Issue #	(3) #Discrim. preds (SD)	(4) #Preds in causal path	#Interventions	
				(5) AID	(6) TAGT
Npgsql [245]	2485 [246]	14	3	5	11
Kafka [172]	279 [173]	72	5	17	33
Azure Cosmos DB [63]	713 [64]	64	7	15	42
`Network`	N/A	24	1	2	5
`BuildAndTest`	N/A	25	3	10	15
`HealthTelemetry`	N/A	93	10	40	70

Figure 6.7: Results from case studies of real-world applications. SD produces way too many spurious predicates beyond the correct causal predicates (columns 3 & 4). SD actually produces even more predicates, but here we only report the number of fully discriminative predicates. AID and traditional adaptive group testing (TAGT) both pin-point the correct causal predicates using interventions, but AID does so with significantly fewer interventions (columns 5 & 6).

one increments it while the other reads it (2) The second thread accesses an array at the incremented index location, which is outside the array size. (3) This access throws `IndexOutOfRange` exception (4) Application fails to handle the exception and crashes. This explanation matches the root cause provided by the developer who reported the bug to Npgsql GitHub repository.

6.6.1.2 Use-after-free in Kafka

Next, we use AID on an application built on Kafka [172], a distributed message queue. On Kafka's GitHub repository, a user reported an issue [173] that causes a Kafka application to intermittently crash or hang. The user also provided a sample code to reproduce the issue; we use a similar code for this case study.

As before, we collected predicate logs from 50 successful and 50 failed executions. Using SD, we identified 72 discriminative predicates. The AC-DAG identified 30 predicates with no causal path to the failure indicating predicate, and hence were discarded. AID then used the intervention algorithm on the remaining 42 predicates. After a sequence of 7 interventions, AID could identify the root-cause predicate. It took an additional 10 rounds

(total 17) of interventions to discover a causal path of 5 predicates that connects the root cause and the failure. The causal path gives the following explanation: (1) The main thread that creates a Kafka `consumer` C starts a child thread (2) the child thread runs too slow before calling a method on C (3) main thread disposes C (4) child thread calls a commit method on C (5) since C has already been disposed by the main thread, the previous step causes an exception, causing the failure. The explanation matches well with the description provided in GitHub.

Overall, AID required 17 interventions to discover the root cause and explanation. In contrast, SD generates 72 predicates, without pinpointing the true root cause or explanation. TAGT could identify all predicates in the explanation, but it takes 33 interventions in the worst case.

6.6.1.3 Timing Bug in Azure Cosmos DB Application

Next, we use AID on an application built on Azure Cosmos DB [63], Microsoft's globally distributed database service for operational and analytics workloads. The application has an intermittent timing bug similar to the one mentioned in a Cosmos DB's pull request on GitHub [64]. In summary, the application populates a cache with several entries that would expire after 1 second, performs a few tasks, and then accesses one of the cached entries. During successful executions, the tasks run fast and end before the cached entries expire. However, a transient fault triggers expensive fault handling code that makes a task run longer than the cache expiry time. This makes the application fail as it cannot find the entry in the cache (i.e., it has already expired).

Using SD, we identified 64 discriminative predicates from successful and failed executions of the application. Applying AID on them required 15 interventions and it generated an explanation consisting of 7 predicates that are consistent with the aforementioned informal explanation. In contrast, SD would generate 64 predicates and TAGT would take 42 interventions in the worst case.

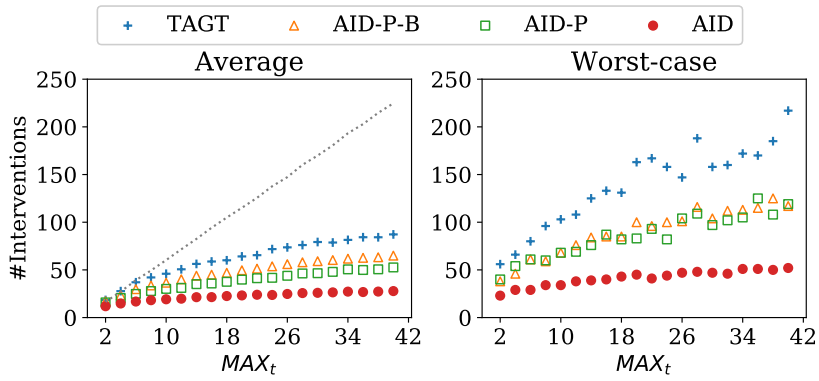


Figure 6.8: Number of interventions required in the average and worst case by traditional adaptive group testing (TAGT) and different variations of AID with varying MAX_t. For average case analysis, total number of predicates is shown using a grey dotted line. Total number of predicates is not shown for the worst-case analysis, because the worst cases vary across approaches.

6.6.1.4 Bugs in Proprietary Software

We applied AID for finding root causes of intermittent failures of several proprietary applications inside Microsoft. We here report our experience with three of the applications that we name as follows (Figure 6.7): (1) `Network`: the control plane of a data center network, (2) `BuildAndTest`: a large-scale software build and test platform, and (3) `HealthTelemetry`: a module used by various services to report their runtime health. Parts of these applications (and associated tests) had been intermittently failing for several months and their developers could not identify the exact root causes. This highlights that the root causes of these failures were nontrivial. AID identified the root causes and generated explanations for how the root causes lead to failures: for `Network`, the root cause was a random number collision, for `BuildAndTest`, it was an order violation of two events, and for `HealthTelemetry`, it was a race condition. Developers of the applications confirmed that the root causes identified by AID are indeed the correct ones and that the explanations given by AID correctly showed how the root causes lead to the (intermittent) failures.

Figure 6.7 also shows the performance of AID with these applications. As before, SD produces many discriminative predicates, only a subset of which are causally related to the

failures. Moreover, for all applications, AID requires significantly fewer interventions than what TAGT would require in the worse case.

6.6.2 Sensitivity Analysis

We further evaluate AID on a benchmark of synthetically generated applications, designed to fail intermittently and with known root causes. We generate multi-threaded applications ranging the maximum number of threads MAX_t from 2 to 40. For each parameter setting, we generate 500 applications. In these applications, the total number of predicates N ranges from 4 to 284, and we randomly choose the number of causal predicates in the range $[1, \frac{N}{\log N}]$.

For this experiment, we compare four approaches: TAGT, AID, AID without predicate pruning (AID-P), and AID without predicate or branch pruning (AID-P-B). All four approaches derive the correct causal paths but differ in the number of required interventions. Figure 6.8 shows the average (left) and the maximum (right) number of interventions required by each approach. The grey dotted line in the average case shows the average number of predicates over the 500 instances for that setting. This experiment provides two key observations:

Interventions in topological order converge faster. Causally related predicates are likely to be topologically close to each other in the AC-DAG. AID discards all predicates in an intervened group only when none are causal. This is unlikely to occur when predicates are grouped randomly. For this reason, AID-P-B, which uses topological ordering, requires fewer interventions than TAGT.

Pruning reduces the required number of interventions. We observe that both predicate and branch pruning reduce the number of interventions. Pruning is a key differentiating factor of AID from TAGT. In the worst-case setting in particular, the

margin between AID and TAGT is significant: TAGT requires up to 217 interventions in one case, while the highest number of interventions for AID is 52.

6.7 Summary and Future Work

In this work, we defined the problem of causal path discovery for explaining failure of concurrent programs. Our key contribution is the novel Adaptive Interventional Debugging (AID) framework, which combines existing statistical debugging, causal analysis, fault injection, and group testing techniques in a novel way to discover root cause of program failure and generate the causal path that explains how the root cause triggers the failure. Such explanation provides better interpretability for understanding and analyzing the root causes of program failures. We showed both theoretically and empirically that AID is both efficient and effective to solve the causal path discovery problem. A possible extension is to incorporate additional information regarding the program behavior to better approximate the causal relationship among predicates, and address the cases of multiple root causes and multiple causal paths. Another interesting direction is to to address the challenge of explaining multiple types of failures as well.

CHAPTER 7

EXPOSING DISCONNECT BETWEEN DATA AND SYSTEMS (DATAEXPOSER)[1]

Traditional software debugging aims to identify errors and bugs in the mechanism—such as source code, configuration files, and runtime conditions—that may cause a system to malfunction [96, 131, 205]. However, in modern systems, data has become a central component that itself can cause a system to fail. Data-driven systems comprise complex pipelines that rely on data to solve a target task. Prior work addressed the problem of debugging machine-learning models [51] and finding root causes of failures in computational pipelines [216], where certain values of the pipeline parameters—such as a specific model and/or a specific dataset—cause the pipeline failure. However, just knowing that a pipeline fails for a certain dataset is not enough; naturally, we ask: what *properties* of a dataset caused the failure?

Two common reasons for malfunctions in data-driven systems are: (1) incorrect data, and (2) *disconnect* between the assumptions about the data and the design of the system that operates on the data. Such disconnects may happen when the system is not robust, i.e., it makes strict assumptions about metadata (e.g., data format, domains, ranges, and distributions), and when new data drifts from the data over which the system was tested on before deployment [270] (e.g., when a system expects a data stream to have a weekly frequency, but the data provider suddenly switches to daily data).

Therefore, in light of a failure, one should investigate potential issues in the data. Some specific examples of commonly observed system malfunctions caused by data include:

[1]The work was done in collaboration with another Ph.D. student Sainyam Galhotra.

(1) decline of a machine-learned model's accuracy (due to out-of-distribution data), (2) unfairness in model predictions (due to imbalanced training data), (3) excessive processing time (due to a system's failure to scale to large data), and (4) system crash (due to invalid input combination in the data tuples beyond what the system was designed to handle). These examples indicate a common problem: *disconnect* or *mismatch* between the data and the system design. Once the mismatch is identified, then possible fixes could be either to repair the data to suit the system design, or to adjust the system design (e.g., modify source code) to accommodate data with different properties.

A naïve approach to deal with potential issues in the data is to identify outliers: report tuples as potentially problematic based on how atypical they are with respect to the rest of the tuples in the dataset. However, without verifying whether the outliers actually cause unexpected outcomes, we can never be certain about the actual root causes. As pointed out in prior work [30]: *"With respect to a computation, whether an error is an outlier in the program's input distribution is not necessarily relevant. Rather, potential errors can be spotted by their effect on a program's output distribution."* To motivate our work, we start with an example taken from a real-world incident, where Amazon's delivery service was found to be racist [159].

Example 7.1 (Biased Classifier)**.** *An e-commerce company wants to build an automated system that suggests who should get discounts. To this end, they collect information from the customers' purchases over one year and build a dataset over the attributes* `name`*,* `gender`*,* `age`*,* `race`*,* `zip_code`*,* `phone`*,* `products_purchased`*, etc. Anita, a data scientist, is then asked to develop a machine learning (ML) pipeline over this dataset to predict whether a customer will spend over a certain amount, and, subsequently, should be offered discounts. Within this pipeline, Anita decides to use a logistic regression classifier for prediction and implements it using an off-the-shelf ML library. To avoid discrimination over any group and to ensure that the classifier trained on this dataset is fair, Anita decides to drop the sensitive attributes—*`race`* and* `gender`*—during the pre-processing step of*

the ML pipeline, before feeding it to the classifier. However, despite this effort, the trained classifier turns out to be highly biased against African American people and women. After a close investigation, Anita discovers that: (1) In the training data, `race` *is highly correlated with* `zip_code`*, and (2) The training dataset is imbalanced: a larger fraction of the people who purchase expensive products are male. Now she wonders: if these two properties did not hold in the dataset, would the learned classifier be fair? Have either (or both) of these properties caused the observed unfairness?*

Unfortunately, existing tools (e.g., CheckCell [30]) that blame individual cells (values) for unexpected outcomes cannot help here, as no single cell in the training data is responsible for the observed discrimination, rather, global statistical properties (e.g., correlation) that involve multiple attributes over the entire data are the actual culprits. Furthermore, Anita only identified two potential or *correlated* data issues that may or may not be the actual cause of the unfairness. To distinguish mere correlation from true causation and to verify if there is indeed a *causal* connection between the data properties and the observed unfairness, we need to dig deeper.

Example 7.1 is one among many incidents in real-world applications where issues in the data caused systems to malfunction [38, 122]. A recent study of 112 high-severity incidents in Microsoft Azure services showed that 21% of the bugs were due to inconsistent assumptions about data format by different software components or versions [213]. The study further found that 83% of the data-format bugs were due to inconsistencies between data producers and data consumers, while 17% were due to mismatch between interpretations of the same data by different data consumers. Similar incidents happened due to misspelling and incorrect date-time format [275], and issues pertaining to data fusion where schema assumptions break for a new data source [75, 325]. We provide another illustrative example where a system times out when the distribution of the data, over which the system operates, exhibits significant skew.

Example 7.2 (Process Timeout)**.** *A toll collection software* EZGo *checks if vehicles passing through a gate have electronic toll pass installed. If it fails to detect a toll pass, it uses an external software* OCR *to extract the registration number from the vehicle's license plate.* EZGo *operates in a batch mode and processes every* 1000 *vehicles together by reserving* AWS *for one hour, assuming that one hour is sufficient for processing each batch. However, for some batches,* EZGo *fails. After a close investigation, it turns out that the external software* OCR *uses an algorithm that is extremely slow for images of black license plates captured in low illumination. As a result, when a batch contains a large number of such cases (significantly skewed distribution),* EZGo *fails.*

The aforementioned examples bring forth two key challenges. First, we need to correctly identify potential causes of unexpected outcomes and generate *hypotheses* that are expressive enough to capture the candidate root causes. For example, "outliers cause unexpected outcomes" is just one of the many possible hypotheses, which offers very limited expressivity. Second, we need to *verify* the hypotheses to confirm or refute them, which enables us to pinpoint the actual root causes, eliminating false positives.

Data profile as root cause. Towards solving the first challenge, our observation is that data-driven systems often function properly for certain datasets, but malfunction for others. Such malfunction is often rooted in certain *properties* of the data, which we call *data profiles*, that distinguish passing and failing datasets. Examples include size of a dataset, domains and ranges of attribute values, correlations between attribute pairs, conditional independence [337], functional dependencies and their variants [53, 90, 154, 187, 251], and other more complex data profiles [61, 197, 252, 299].

Oracle-guided root cause identification. Our second observation is that if we have access to an *oracle* that can indicate whether the system functions desirably or not, we can verify our hypotheses. Access to an oracle allows us to precisely isolate the correct root causes of the undesirable malfunction from a set of candidate causes. Here, an oracle is a

mechanism that can characterize whether the system functions properly over the input data. The definition of proper functioning is application-specific; for example, achieving a certain accuracy may indicate proper functioning for an ML pipeline. Such oracles are often available in many practical settings, and have been considered in prior work [96, 216].

Solution sketch. We propose DATAEXPOSER, a framework that identifies and exposes data profiles that cause a data-driven system to malfunction. Our framework involves two main components: (1) an intervention-based mechanism that alters the profiles of a dataset, and (2) a mechanism that speeds up analysis by carefully selecting appropriate interventions. Given a scenario where a system malfunctions (fails) over a dataset but functions properly (passes) over another, DATAEXPOSER focuses on the *discriminative* profiles, i.e., data profiles that significantly differ between the two datasets. DATAEXPOSER's intervention mechanism modifies the "failing" dataset to alter one of the discriminative profiles; it then observes whether this intervention causes the system to perform desirably, or the malfunction persists. DATAEXPOSER speeds up this analysis by favoring interventions on profiles that are deemed more likely causes of the malfunction. To estimate this likelihood, we leverage three properties of a profile: (1) *coverage*: the more tuples an intervention affects, the more likely it is to fix the system behavior, (2) *discriminating power*: the bigger the difference between the failing and the passing datasets over a profile, the more likely that the profile is a cause of the malfunction, and (3) *attribute association*: if a profile involves an attribute that is also involved with a large number of other discriminative profiles, then that profile has high likelihood to be a root cause. This is because altering such a profile is likely to passively repair other discriminative profiles as a side-effect (through the associated attribute). We also provide a group-testing-based technique that allows *group intervention*, which helps expedite the root-cause analysis further.

Scope. In this work, we assume knowledge of the classes of (domain-specific) data profiles that encompass the potential root causes. E.g., in Example 7.1, we assume the knowl-

edge that correlation between attribute pairs and disparity between the conditional probability distributions (the probability of belonging to a certain gender, given price of items bought) are potential causes of malfunction. This assumption is realistic because: (1) For a number of tasks there exists a well-known set of relevant profiles: e.g., class imbalance and correlation between sensitive and nonsensitive attributes are common causes of unfairness in classification [36]; and violation of conformance constraints [101], missing values, and out-of-distribution tuples are well-known causes of ML model's performance degradation. (2) Domain experts are typically aware of the likely class of data profiles for the specific task at hand and can easily provide this additional knowledge as a *conservative* approximation, i.e., they can include extra profiles just to err on the side of caution. Notably, this assumption is also extremely common in software debugging techniques [96, 205, 350], which rely on the assumption that the "predicates" (traps to extract certain runtime conditions) are expressive enough to encode the root causes, and software testing [231], validation [193], and verification [139] approaches, which rely on the assumption that the test cases, specifications, and invariants reasonably cover the codebase and correctness constraints.

To support a data profile, DATAEXPOSER further needs the corresponding mechanisms for discovery and intervention. In this work, we assume knowledge of the profile discovery and intervention techniques, as they are orthogonal to our work. Nevertheless, we discuss some common classes of data profiles supported in DATAEXPOSER and the corresponding discovery and intervention techniques. For data profile discovery, we rely on prior work on pattern discovery [262], statistical-constraint discovery [337], data-distribution learning [137], knowledge-graph-based concept identification [113], etc. While our evaluation covers specific classes of data profiles (for which there exist efficient discovery techniques), our approach is generic and works for any class of data profiles, as long as the corresponding discovery and intervention techniques are available.

Limitations of prior work. To find potential issues in data, Dagger [274, 275] provides data debugging primitives for human-in-the-loop interactions with data-driven computational pipelines. Other explanation-centric efforts [24, 60, 118, 325] report salient properties of historical data based only on observations. In contrast with observational techniques, the presence of an oracle allows for interventional techniques [216] that can query the oracle with additional, system-generated test cases to identify root causes of system malfunction more accurately. One such approach is CheckCell [30], which presents a ranked list of cells of data rows that unusually affect output of a given target function. CheckCell uses a fine-grained approach: it removes one cell of the data at a time, and observes changes in the output distribution. While it is suitable for small datasets, where it is reasonable to expect a human-in-the-loop paradigm to fix cells one by one, it is not suitable for large datasets, where no individual cell is significantly responsible, rather, a holistic property of the entire dataset (profile) causes the problem.

Interpretable machine learning is related to our problem, where the goal is to explain behavior of machine-learned models. However, prior work on interpretable machine learning [276, 277] typically provides *local* (tuple-level) explanations, as opposed to *global* (dataset-level) explanations. While some approaches provide feature importance as a global explanation for model behavior [54], they do not model feature interactions as possible explanations.

Software testing and debugging techniques [16, 18, 56, 107, 121, 128, 144, 169, 205, 350] are either application-specific, require user-defined test suites, or rely only on observational data. The key contrast between software debugging and our setting is that the former focuses on white-box programs: interventions, runtime conditions, program invariants, control-flow graphs, etc., all revolve around program source code and execution traces. Unlike programs, where lines have logical and semantic connections, tuples in data do not have similar associations. Data profiles significantly differ in their semantics, and discovery and intervention techniques from program profiles, and, thus, techniques

for program profiling do not trivially apply here. We treat data as a first-class citizen in computational pipelines, while considering the program as a black box.

Contributions. We make the following contributions:

- We formalize the novel problem of identifying root causes (and fixes) of the disconnect between data and data-driven systems in terms of data profiles (and interventions). (Section 7.1)
- We design a set of data profiles that are common root causes of data-driven system malfunctions, and discuss their discovery and intervention techniques based on available technology. (Section 7.2)
- We design and develop a novel interventional approach to pinpoint causally verified root causes. The approach leverages a few properties of the data profiles to efficiently explore the space of candidate root causes with a small number of interventions. Additionally, we develop an efficient group-testing-based algorithm that further reduces the number of required interventions. (Section 7.3)
- We evaluate DATAEXPOSER on three real-world applications, where data profiles are responsible for causing system malfunction, and demonstrate that DATAEXPOSER successfully explains the root causes with a very small number of interventions (< 5). Furthermore, DATAEXPOSER requires 10–$1000\times$ fewer interventions when compared against two state-of-the-art techniques for root-cause analysis: BugDoc [216] and Anchors [277]. Through an experiment over synthetic pipelines, we further show that the number of required interventions by DATAEXPOSER increases sub-linearly with the number of discriminative profiles, thanks to our group-testing-based approach. (Section 7.4)

7.1 Preliminaries and Problem Definition

In this section, we first formalize the notions of system malfunction and data profile, its violation, and transformation function used for intervention. We then proceed to define explanation (cause and corresponding fix) of system malfunction and formulate the problem of data-profile-centric explanation of system malfunction.

Basic notations. We use $\mathcal{R}(A_1, A_2, \ldots, A_m)$ to denote a relation schema over m attributes, where A_i denotes the i^{th} attribute. We use $\texttt{Dom}_i$ to denote the domain of attribute A_i. Then the set $\mathbf{Dom}^m = \texttt{Dom}_1 \times \cdots \times \texttt{Dom}_m$ specifies the domain of all possible tuples. A dataset $D \subseteq \mathbf{Dom}^m$ is a specific instance of the schema $\mathcal{R}$. We use $t \in \mathbf{Dom}^m$ to denote a tuple in the schema $\mathcal{R}$. We use $t.A_i \in \texttt{Dom}_i$ to denote the value of the attribute A_i of the tuple t and use $D.A_j$ to denote the multiset of values all tuples in D take for attribute A_j.

7.1.1 Quantifying System Malfunction

To measure how much the system malfunctions over a dataset, we use the *malfunction score*.

Definition 7.1 (Malfunction score)**.** *Let $D \subseteq \mathbf{Dom}^m$ be a dataset, and S be a system operating on D. The malfunction score $m_S(D) \in [0, 1]$ is a real value that quantifies how much S* malfunctions *when operating on D.*

The malfunction score $m_S(D) = 0$ indicates that S functions properly over D and a higher value indicates a higher degree of malfunction, with 1 indicating extreme malfunction. A threshold parameter τ defines the acceptable degree of malfunction and translates the continuous notion of malfunction to a Boolean value. If $m_S(D) \leq \tau$, then D is considered to pass with respect to S; otherwise, there exists a mismatch between D and S, whose cause (and fix) we aim to expose.

Example 7.3. *For a binary classifier, its misclassification rate (additive inverse of accuracy) over a dataset can be used as a malfunction score. Given a dataset D, if a classifier*

S makes correct predictions for tuples in $D' \subseteq D$, and incorrect predictions for the remaining tuples, then S achieves accuracy $\frac{|D'|}{|D|}$, and, thus, $m_S(D) = 1 - \frac{|D'|}{|D|}$.

Example 7.4. *In fair classification, we can use disparate impact [152], which is defined by the ratio between the number of tuples with favorable outcomes within the unprivileged and the privileged groups, to measure malfunction.*

7.1.2 Profile-Violation-Transformation (PVT)

Once we detect existence of a mismatch, the next step is to investigate its cause. We characterize the issues in a dataset that are responsible for the mismatch between the dataset and the system using *data profiles*. Structure or schema of data profiles is given by profile *templates*, which contains holes for parameters. Parameterizing a profile template gives us a *concretization* of the corresponding profile (P). Given a dataset D, we use existing data-profiling techniques to find out parameter values to obtain concretized data profiles, such that D satisfies the concretized profiles. To evaluate how much a dataset D satisfies or violates a data profile, we need a corresponding violation function (V). Violation functions provide *semantics* of the data profiles. Finally, to alter a dataset D, with respect to a data profile and the corresponding violation function, we need a transformation function (T). Transformation functions provide the intervention mechanism to alter data profiles of a dataset and suggest fix to remove the cause of malfunction. DATAEXPOSER requires the following three components over the schema ⟨**P**rofile, **V**iolation function, **T**ransformation function⟩, PVT in short:

1. P: a (concretized) profile along with its parameters, which follows the schema ⟨profile type, parameters⟩.
2. $V(D, P)$: a violation function that computes how much the dataset D violates the profile P and returns a violation score.

3. $T(D, P, V)$: a transformation function that transforms the dataset D to another dataset D' such that D' no longer violates the profile P with respect to the violation function V. (When clear from the context, we omit the parameters P and V when using the notation for transformation functions.)

For a PVT triplet X, we define X_P as its profile, X_V as the violation function and X_T as the transformation function. We provide examples and additional discussions on data profiles, violation functions, and transformation functions in Section 7.2.

7.1.2.1 Data Profile

Intuitively, data profiles encode dataset characteristics. They can refer to a single attribute (e.g., mean of an attribute) or multiple attributes (e.g., correlation between a pair of attributes, functional dependencies, etc.).

Definition 7.2 (Data Profile)**.** *Given a dataset D, a data profile P denotes properties or constraints that tuples in D (collectively) satisfy.*

7.1.2.2 Profile Violation Function

To quantify the degree of violation a dataset incurs with respect to a data profile, we use a *profile violation function* that returns a numerical violation score.

Definition 7.3 (Profile violation function)**.** *Given a dataset D and a data profile P, a* profile violation function $V(D, P) \mapsto [0, 1]$ *returns a real value that quantifies how much D violates P.*

$V(D, P) = 0$ implies that D fully complies with P (does not violate it at all). In contrast, $V(D, P) > 0$ implies that D violates P. The higher the value of $V(D, P)$, the higher the profile violation.

		Profile	Data type	Discovery over D	Interpretation	Violation by D	Transformation func.
strict	1	$\langle$DOMAIN, A_j, $\mathbb{S}\rangle$	Categorical	$\mathbb{S} = \bigcup_{t \in D} \{t.A_j\}$	Values are drawn from a specific domain.	$\frac{\sum_{t \in D} [\![t.A_j \notin \mathbb{S}]\!]}{\lvert D \rvert}$	Map values outside $\mathbb{S}$ to values in $\mathbb{S}$ using domain knowledge.
	2	$\langle$DOMAIN, A_j, $\mathbb{S}\rangle$	Numerical	$\mathbb{S}$ = $[\texttt{lb}, \texttt{ub}]$, where $\texttt{lb} = \min_{t \in D} t.A_j$, $\texttt{ub} = \max_{t \in D} t.A_j$	Values lie within a bound.	$\frac{\sum_{t \in D} [\![t.A_j \notin \mathbb{S}]\!]}{\lvert D \rvert}$	(1) Use monotonic linear transformation and transform all values. (2) Use winsorization techniques to replace the violating values only.
	3	$\langle$DOMAIN, A_j, $\mathbb{S}\rangle$	Text	$\mathbb{S} = [t \in \texttt{Dom}_j \mid t \models \mathbb{P}]$, where $\mathbb{P}$ is a regex over $D.A_j$ learned via pattern discovery [262]	Values satisfy a regular expression or length of values lie within a bound.	$\frac{\sum_{t \in D} [\![t.A_j \notin \mathbb{S}]\!]}{\lvert D \rvert}$	Minimally alter data to satisfy regular expression. For example, insert (remove) characters to increase (reduce) text length.
thresholded by data coverage	4	$\langle$OUTLIER, A_j, O, $\theta\rangle$	All	$\theta = \frac{\sum_{t \in D} [\![O(D.A_j, t.A_j)]\!]}{\lvert D \rvert}$, where O is learned from $D.A_j$'s distribution [137]	Fraction of outliers within an attribute does not exceed a threshold.	$\max\left(0, \frac{\sum_{t \in D} [\![O(D.A_j, t.A_j)]\!] - \theta \cdot \lvert D \rvert}{\lvert D \rvert \cdot (1-\theta)}\right)$	(1) Replace outliers with the expected value (mean, median, mode) of the attribute. (2) Map all values above (below) the maximum (minimum) limit with highest (lowest) valid value.
	5	$\langle$MISSING, A_j, $\theta\rangle$	All	$\theta = \frac{\sum_{t \in D} [\![t.A_j = \texttt{NULL}]\!]}{\lvert D \rvert}$	Fraction of missing values within an attribute does not exceed a threshold.	$\max\left(0, \frac{\sum_{t \in D} [\![t.A_j = \texttt{NULL}]\!] - \theta \cdot \lvert D \rvert}{\lvert D \rvert \cdot (1-\theta)}\right)$	Use missing value imputation techniques.
	6	$\langle$SELECTIVITY, $\mathbb{P}$, $\theta\rangle$	All	$\theta = \frac{\lvert \sigma_{\mathbb{P}}(D) \rvert}{\lvert D \rvert}$	Fraction of tuples satisfying a given constraint (selection predicate) does not exceed a threshold.	$\max\left(0, \frac{\lvert \sigma_{\mathbb{P}}(D) \rvert - \theta \cdot \lvert D \rvert}{\lvert D \rvert \cdot (1-\theta)}\right)$	Undersample tuples that satisfy the predicate $\mathbb{P}$.
thresholded by parameter	7	$\langle$INDEP, A_j, A_k, $\alpha\rangle$	Categorical	α denotes Chi-squared statistic between $D.A_j$ and $D.A_k$	χ^2 statistic between a pair of attributes is below a threshold with a p-value ≤ 0.05.	$1 - e^{-\max(0, \chi^2(D.A_j, D.A_k) - \alpha)}$	Modify attribute values to remove/reduce dependence.
	8	$\langle$INDEP, A_j, A_k, $\alpha\rangle$	Numerical	α denotes Pearson correlation coefficient between $D.A_j$ and $D.A_k$	PCC between a pair attributes is below a threshold with a p-value ≤ 0.05.	$\max\left(0, \frac{\lvert \texttt{PCC}(D.A_j, D.A_k) \rvert - \lvert \alpha \rvert}{1 - \lvert \alpha \rvert}\right)$	Add noise to remove/reduce dependence between attributes.
	9	$\langle$INDEP, A_j, A_k, $\alpha\rangle$	Categorical, numerical	Learn causal graph and causal coefficients (α) using TETRAD [288]	A causal relationship between a pair of attributes is unlikely (with causal coefficient less than α).	$\max\left(0, \frac{\lvert \texttt{coeff}(A_j, A_k) \rvert - \alpha}{1 - \alpha}\right)$	Change data distribution to modify the causal relationship.

Figure 7.1: A list of PVT triplets that we consider in this work, their syntax, and semantics.

7.1.2.3 Transformation Function

In our work, we assume knowledge of a passing dataset for which the system functions properly, and a failing dataset for which the system malfunctions. Our goal is to identify which profiles of the failing dataset caused the malfunction. We seek answer to the question: how to "fix" the issues within the failing dataset such that the system no longer malfunctions on it (i.e., mismatch is resolved)? To this end, we apply *interventional causal reasoning*: we intervene on the failing dataset by altering its attributes such that the profile of the altered dataset matches the corresponding correct profile of the passing dataset. To perform intervention, we need *transformation functions* with the property that it should push the failing dataset "closer" to the passing dataset in terms of the profile that we are interested to alter. More formally, after the transformation, the profile violation score should decrease.

Definition 7.4 (Transformation function)**.** *Given a dataset D, a data profile P, and a violation function V, a transformation function $T(D, P, V) \mapsto 2^{\mathbf{Dom}^m}$ alters D to produce D' such that $V(D', P) = 0$.*

A dataset can be transformed by applying a series of transformation functions, for which we use the composition operator ($\circ$).

Definition 7.5 (Composition of transformations)**.** *Given a dataset D, and two PVT triplets X and Y, $(X_T \circ Y_T)(D) = X_T(Y_T(D))$. Further, if $D'' = (X_T \circ Y_T)(D)$, then $X_V(D'', X_P) = Y_V(D'', Y_P) = 0$.*

7.1.3 Problem Definition

We expose a set of PVT triplets for explaining the system malfunction. The explanation contains both the *cause* and the corresponding *fix*: profile within a PVT triplet indicates the cause of system malfunction with respect to the corresponding transformation function, which suggests the fix.

Definition 7.6 (Explanation of system malfunction)**.** *Given*

1. *a system S with a mechanism to compute $m_S(D)$ $\forall D \subseteq \mathbf{Dom}^m$,*
2. *an allowable malfunction threshold τ,*
3. *a passing dataset D_{pass} for which $m_S(D_{pass}) \leq \tau$,*
4. *a failing dataset D_{fail} for which $m_S(D_{fail}) > \tau$, and*
5. *a set of candidate PVT triplets $\mathcal{X}$ such that $\forall X \in \mathcal{X} \quad X_V(D_{pass}, X_P) = 0 \wedge X_V(D_{fail}, X_P) > 0$,*

the explanation of the malfunction of S for D_{fail}, but not for D_{pass}, is a set of PVT triplets $\mathcal{X}^* \subseteq \mathcal{X}$ such that $m_S((\circ_{X \in \mathcal{X}^*} X_T)(D_{fail})) \leq \tau$.

Informally, $\mathcal{X}^*$ explains the cause: why S malfunctions for D_{fail}, but not for D_{pass}. More specifically, *failing to satisfy* the profiles of the PVT triplets in $\mathcal{X}^*$ is the cause of malfunction. Furthermore, the transformation functions of the PVT triplets in $\mathcal{X}^*$ suggest the fix: how can we repair D_{fail} to eliminate system malfunction. However, there could be many possible such $\mathcal{X}^*$ and we seek a *minimal* set $\mathcal{X}^*$ such that transformation for every $X \in \mathcal{X}^*$ is necessary to bring down the malfunction score below the threshold τ.

Definition 7.7 (Minimal explanation of system malfunction)**.** *Given a system S that malfunctions for D_{fail} and an allowable malfunction threshold τ, an explanation $\mathcal{X}^*$ of S's malfunction for D_{fail} is minimal if $\forall \mathcal{X}' \subset \mathcal{X}^*$ $m_S((\circ_{X \in \mathcal{X}'} X_T)(D_{fail})) > \tau$.*

Note that there could be multiple such minimal explanations and we seek any one of them, as any minimal explanation exposes the causes of mismatch and suggests minimal fixes.

Problem 7.1 (Discovering explanation of mismatch between data and system)**.** *Given a system S that malfunctions for D_{fail} but functions properly for D_{pass}, the problem of discovering the explanation of mismatch between D_{fail} and S is to find a minimal explanation*

that captures (1) the cause why S malfunctions for D_{fail} but not for D_{pass} and (2) how to repair D_{fail} to remove the malfunction.

7.2 Data Profiles, Violation Functions, & Transformation Functions

We now provide an overview of the data profiles we consider, how we discover them, how we compute the violation scores for a dataset w.r.t. a data profile, and how we apply transformation functions to alter profiles of a dataset. While a multitude of data-profiling primitives exist in the literature, we consider a carefully chosen subset of them that are particularly suitable for modeling issues in data that commonly cause malfunction or failure of a system. We focus on profiles that, by design, can better "discriminate" a pair of datasets as opposed to "generative" profiles (e.g., data distribution) that can profile the data better, but nonetheless are less useful for the task of discriminating between two datasets. However, the DATAEXPOSER framework is generic, and other profiles can be plugged into it.

As discussed in Section 7.1, a PVT triplet encapsulates a profile, and corresponding violation and transformation functions. Figure 7.1 provides a list of profiles along with the data types they support, how to learn their parameters from a given dataset, how to interpret them intuitively, and the corresponding violation and transformation functions. In this work, we assume that a profile can be associated with multiple transformation functions (e.g., rows 2 and 4), but each transformation function can be associated with at most one profile. This assumption helps us to blame a unique profile as cause of the system malfunction when at least one of the transformation functions associated with that profile is verified to be a fix.

PVT triplets can be classified in different ways. Based on the strictness of the violation function, they can be classified as follows:

- *Strict*: *All* tuples are expected to satisfy the profile (rows 1–3).
- *Thresholded by data coverage:* Certain fraction (θ) of data tuples are allowed to violate the profile (rows 4–6).

- *Thresholded by a parameter:* Some degree of violation is allowed with respect to a specific parameter (α) (rows 7–9).

Further, PVT triplets can be classified in two categories based on the nature of the transformation functions:

- *Local* transformation functions can transform a tuple in isolation without the knowledge of how other tuples are being transformed (e.g., rows 1–3). Some local transformation functions only transform the violating tuples (e.g., row 2, transformation (2)), while others transform all values (e.g., row 2, transformation (1)). For instance, in case of unit mismatch (kilograms vs. lbs), it is desirable to transform all values and not just the violating ones.

- *Global* transformation functions are holistic, as they need the knowledge of how other tuples are being transformed while transforming a tuple (e.g., rows 6 and 9).

Example 7.5. DOMAIN *requires two parameters: (1) an attribute $A_j \in \mathcal{R}(D)$, and (2) a set $\mathbb{S}$ specifying its domain. A dataset D satisfies $\langle \text{DOMAIN}, A_j, \mathbb{S} \rangle$ if $\forall t \in D \;\; t.A_j \in \mathbb{S}$. The profile $\langle \text{DOMAIN}, A_j, \mathbb{S} \rangle$ is minimal w.r.t. D if $\nexists \mathbb{S}' \subset \mathbb{S}$ s.t. D satisfies the profile $\langle \text{DOMAIN}, A_j, \mathbb{S}' \rangle$. The technique for discovering a domain $\mathbb{S}$ varies depending on the data type of the attribute. Rows 1–3 of Figure 7.1 show three different domain-discovery techniques for different data types.*

$People_{fail}$ (Figure 7.2) satisfies $\langle \text{DOMAIN}, \texttt{gender}, \{F, M\} \rangle$, as all tuples draw values from $\{F, M\}$ for the attribute `gender`. Our case studies of `Sentiment Prediction` and `Cardiovascular Disease Prediction` show the application of the profile DOMAIN *(Section 7.4).*

Example 7.6. OUTLIER *requires three parameters: (1) an attribute $A_j \in \mathcal{R}(D)$, (2) an outlier detection function $O(A, a) \mapsto \{\texttt{True}, \texttt{False}\}$ that returns* `True` *if a is an outlier w.r.t. the values within A, and* `False` *otherwise, and (3) a threshold $\theta \in [0, 1]$. A dataset D*

satisfies $\langle$OUTLIER$, A_j, O, \theta\rangle$ *if the fraction of outliers within the attribute* A_j*—according to* O*—does not exceed* θ*. Otherwise, we compute how much the fraction of outliers exceeds the allowable fraction of outliers (*θ*) and then normalize it by dividing by* $(1-\theta)$*. The profile* $\langle$OUTLIER$, A_j, O, \theta\rangle$ *is minimal if* $\nexists\theta' < \theta$ *s.t.* D *satisfies* $\langle$OUTLIER$, A_j, O, \theta'\rangle$*.*

An outlier detection function $O_{1.5}$ *identifies values that are more than* 1.5 *standard deviation away from the mean as outliers. In* $People_{fail}$*,* `age` *has a mean* 34.5 *and a standard deviation* 11.78*. According to* $O_{1.5}$*, only* t_3*—which is* 0.1 *fraction of the tuples—is an outlier in terms of* `age` *as* t_3*'s age* $60 > (34.5 + 1.5 \times 11.78) = 52.17$*. Therefore,* $People_{fail}$ *satisfies* $\langle$OUTLIER$,$ `age`$, O_{1.5}, 0.1\rangle$*.*

Example 7.7. INDEP *requires three parameters: two attributes* $A_j, A_k \in \mathcal{R}(D)$*, and a real value* α*. A dataset* D *satisfies the profile* $\langle$INDEP$, A_j, A_k, \alpha\rangle$ *if the dependency between* $D.A_j$ *and* $D.A_k$ *does not exceed* α*. Different techniques exist to quantify the dependency and rows 6–9 of Figure 7.1 show three different ways to model dependency, where the first two are correlational and the last one is causal.*

$\langle$INDEP$,$ `race`$,$ `high_expenditure`$, 0.67\rangle$ *is satisfied by* $People_{fail}$ *using the PVT triplet of row 7, as* χ^2*-statistic between* `race` *and* `high_expenditure` *over* $People_{fail}$ *is* 0.67*. We show the application of the profile* INDEP *in our case study involving the task of* `Income Prediction` *in Section 7.4.*

While the profiles in Figure 7.1 are defined over the entire data, analogous to conditional functional dependency [90], an extension to consider is *conditional* profiles, where only a subset of the data is required to satisfy the profiles.

7.3 Intervention Algorithms

We now describe our intervention algorithms to explain the mismatch between a dataset and a system malfunctioning on that dataset. Our algorithms consider a failing and a passing dataset as input and report a *collection* of PVT triplets (or simply PVTs) as the

id	name	gender	age	race	zip code	phone	high expenditure
t_1	Shanice Johnson	F	45	A	01004	2088556597	no
t_2	DeShawn Bad	M	40	A	01004	2085374523	no
t_3	Malik Ayer	M	60	A	01005	2766465009	no
t_4	Dustin Jenner	M	22	W	01009	7874891021	yes
t_5	Julietta Brown	F	41	W	01009		yes
t_6	Molly Beasley	F	32	W		7872899033	no
t_7	Jake Bloom	M	25	W	01101	4047747803	yes
t_8	Luke Stonewald	M	35	W	01101	4042127741	yes
t_9	Scott Nossenson	M	25	W	01101		yes
t_{10}	Gabe Erwin	M	20	W		4048421581	yes

Figure 7.2: A sample dataset $People_{fail}$ with 10 entities. A logistic regression classifier trained over this dataset discriminates against African Americans (`race` = 'A') and women (`gender` = 'F') (Example 7.1).

id	name	gender	age	race	zip code	phone	high expenditure
t_1	Darin Brust	M	25	W	01004	2088556597	no
t_2	Rosalie Bad	F	22	W	01005		no
t_3	Kristine Hilyard	F	50	W	01004	2766465009	yes
t_4	Chloe Ayer	F	22	A		7874891021	yes
t_5	Julietta Mchugh	F	51	W	01009	9042899033	yes
t_6	Doria Ely	F	32	A	01101		yes
t_7	Kristan Whidden	F	25	W	01101	4047747803	no
t_8	Rene Strelow	M	35	W	01101	6162127741	yes
t_9	Arial Brent	M	45	W	01102	4089065769	yes

Figure 7.3: A sample dataset $People_{pass}$ with 9 entities. A logistic regression classifier trained over this dataset does not discriminate against any specific race or gender, and, thus, is fair (Example 7.1).

explanation (cause and fix) of the observed mismatch. To this end, we first identify a set of *discriminative* PVTs—whose profiles take different values in the failing and passing datasets—as potential explanation units, and then intervene on the failing dataset to alter the profiles and observe change in system malfunction.

We develop two approaches that differ in terms of the number of PVTs considered simultaneously during an intervention. DATAEXPOSER$_{\text{GRD}}$ is a greedy approach that considers only one PVT at a time. However, in worst case, the number of interventions required by DATAEXPOSER$_{\text{GRD}}$ is linear in number of discriminative PVTs. Therefore, we propose a second algorithm DATAEXPOSER$_{\text{GT}}$, built on the group-testing paradigm, that considers

multiple PVTs to reduce the number of interventions, where the number of required interventions is *logarithmically* dependent on the number of discriminative PVTs. We start with an example scenario to demonstrate how DATAEXPOSER$_{\text{GRD}}$ works and then proceed to describe our algorithms.

7.3.1 Example Scenario

Consider the task of predicting the attribute `high_expenditure` to determine if a customer should get a discount (Example 7.1). The system uses bias of the trained classifier against the unprivileged groups (measured using disparate impact [152]) as its malfunction score. We seek the causes of mismatch between this prediction pipeline and $People_{fail}$ (Figure 7.2), for which the pipeline fails with a malfunction score of 0.75. We assume the knowledge of $People_{pass}$ (Figure 7.3), for which the malfunction score is 0.15. The goal is to identify a minimal set of PVTs whose transformation functions bring down the malfunction score of $People_{fail}$ below 0.20.

(Step 1) The first goal is to identify the profiles whose parameters differ between $People_{fail}$ and $People_{pass}$. To do so, DATAEXPOSER$_{\text{GRD}}$ identifies the exhaustive set of PVTs over $People_{pass}$ and $People_{fail}$ and discards the identical ones (PVTs with identical profile-parameter values). We call the PVTs of the passing dataset whose profile-parameter values differ over the failing dataset *discriminative* PVTs. Figure 7.5 lists a few profiles of the discriminative PVTs w.r.t. $People_{pass}$ and $People_{fail}$.

(Step 2) Next, DATAEXPOSER$_{\text{GRD}}$ ranks the set of discriminative PVTs based on their likelihood of offering an explanation of the malfunction. Our intuition here is that if an attribute A is related to the malfunction, then many PVTs containing A in their profiles would differ between $People_{fail}$ and $People_{pass}$. Additionally, altering A with respect to one PVT is likely to automatically "fix" other PVTs associated with A.[2] Based on this in-

[2]Altering values of A w.r.t. a PVT may also increase violation w.r.t. some other PVTs. However, for ease of exposition, we omit such issues in this example.

tuition, DATAEXPOSER$_{\text{GRD}}$ constructs a *bipartite graph*, called PVT-attribute graph, with discriminative PVTs on one side and data attributes on the other side (Figure 7.4). In this graph, a PVT X is connected to an attribute A if X_P is defined over A. In the bipartite graph, the degree of an attribute A captures the number of discriminative PVTs associated with A. During intervention, DATAEXPOSER$_{\text{GRD}}$ prioritizes PVTs associated with a high-degree attributes. For instance, since `high_expenditure` has the highest degree in Figure 7.4, PVTs associated with it are considered for intervention before others.

(Step 3) DATAEXPOSER$_{\text{GRD}}$ further ranks the subset of the discriminative PVTs that are connected to the highest-degree attributes in the PVT-attribute graph based on their *benefit score*. Benefit score of a PVT X encodes the likelihood of reducing system malfunction when the failing dataset is altered using X_T. The benefit score of X is estimated from (1) the violation score that the failing dataset incurs w.r.t. X_V, and (2) the number of tuples in the failing dataset that are altered by X_T. For example, to break the dependence between `high_expenduture` and `race`, the transformation corresponding to INDEP modifies five tuples in $People_{fail}$ by perturbing (adding noise to) `high_expenditure`. In contrast, the transformation for MISSING needs to change only one value (t_6 or t_{10}). Since more tuples are affected by the former, it has higher likelihood of reducing the malfunction score. The intuition behind this is that if a transformation alters more tuples in the failing dataset, the more likely it is to reduce the malfunction score. This holds particularly in applications where the system optimizes aggregated statistics such as accuracy, recall, F-score, etc.

(Step 4) DATAEXPOSER$_{\text{GRD}}$ starts intervening on $People_{fail}$ using the transformation of the PVT corresponding to the profile $\langle$INDEP, `race`, `high_expenditure`, $0.04\rangle$ as its transformation offers the most likely fix. Then, it evaluates the malfunction of the system over the altered version of $People_{fail}$. Breaking the dependence between `high_expenditure` and `race` helps reduce bias in the trained classifier, and, thus,

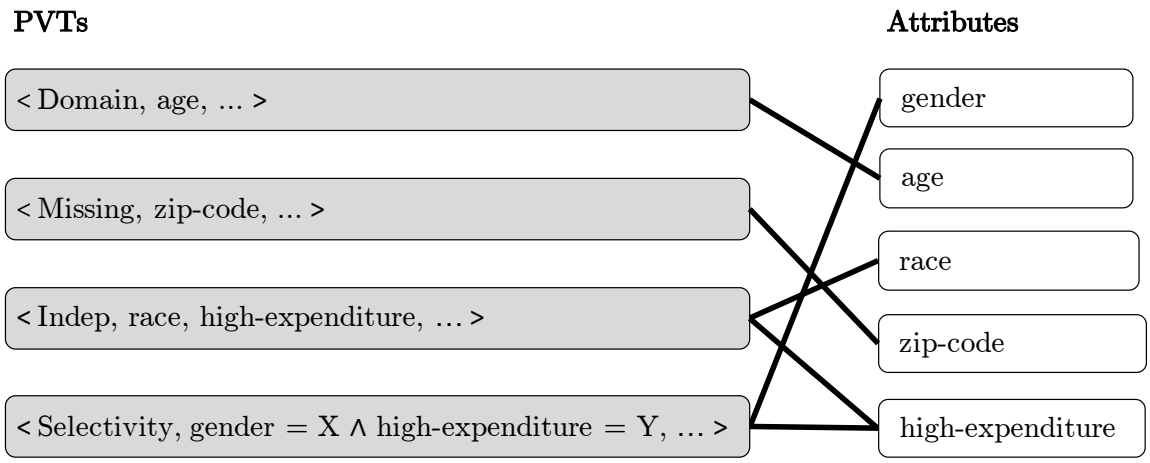


Figure 7.4: PVT-attribute graph. The attribute `high_expenditure` is associated with two discriminative PVTs. For ease of exposition, we only show profile within a PVT to denote the entire PVT.

$People_{pass}$	$People_{fail}$
$\langle$DOMAIN, `age`, $[22, 51]\rangle$	$\langle$DOMAIN, `age`, $[20, 60]\rangle$
$\langle$MISSING, `zip_code`, $0.11\rangle$	$\langle$MISSING, `zip_code`, $0.2\rangle$
$\langle$INDEP, `race`, `high_expenditure`, $0.04\rangle$	$\langle$INDEP, `race`, `high_expenditure`, $0.67\rangle$
$\langle$SELECTIVITY, `gender` $=$ F $\wedge$`high_expenditure` $=$ yes, $0.44\rangle$	$\langle$SELECTIVITY, `gender` $=$ F $\wedge$`high_expenditure` $=$ yes, $0.1\rangle$

Figure 7.5: A list of PVTs that discriminate $People_{pass}$ (Figure 7.3) and $People_{fail}$ (Figure 7.2) based on the scenario of Example 7.1 . We omit the violation and transformation functions for ease of exposition.

we observe a malfunction score of 0.35 w.r.t. the altered dataset. This exposes the first explanation of malfunction.

(Step 5) DATAEXPOSER$_{\text{GRD}}$ then removes the processed PVT (INDEP) from the PVT-attribute graph, updates the graph according to the altered dataset, and re-iterates steps 2–4. Now the PVT corresponding to the profile SELECTIVITY is considered for intervention as it has the highest benefit score. To do so, DATAEXPOSER$_{\text{GRD}}$ oversamples tuples corresponding to female customers with `high_expenditure` $= yes$. This time, DATAEXPOSER$_{\text{GRD}}$ intervenes on the transformed dataset obtained from the previous step. After this transformation, bias of the learned classifier further reduces and the malfunction score falls below

the required threshold. Therefore, with these two interventions, DATAEXPOSER$_{\text{GRD}}$ is able to expose two issues that caused undesirable behavior of the prediction model trained on $People_{fail}$.

(Step 6) DATAEXPOSER$_{\text{GRD}}$ identifies an initial explanation over two PVTs: INDEP and SELECTIVITY. However, to verify whether it is a minimal, DATAEXPOSER$_{\text{GRD}}$ tries to drop from it one PVT at a time to obtain a proper subset of the initial explanation that is also an explanation. This procedure guarantees that the explanation only consists of PVTs that are *necessary*, and, thus, is minimal. In this case, both INDEP and SELECTIVITY are necessary, and, thus, are part of the minimal explanation. DATAEXPOSER$_{\text{GRD}}$ finally reports the following as a minimal explanation of the malfunction, where failure to satisfy the profiles is the cause and the transformations indicate fix (violation and transformation functions are omitted).

$$\{\langle \text{INDEP}, \texttt{race}, \texttt{high\_expenditure}, 0.04\rangle,$$
$$\langle \text{SELECTIVITY}, \texttt{gender} = \text{F} \wedge \texttt{high\_expenditure} = \text{yes}, 0.44\rangle\}$$

7.3.2 Assumptions and Observations

We now proceed to describe our intervention algorithms more formally. We first state our assumptions and then proceed to present our observations that lead to the development of our algorithms.

Assumptions. DATAEXPOSER makes the following assumptions:

(A1) The ground-truth explanation of malfunction is captured by at least one of the discriminative PVTs. This assumption is prevalent in software-debugging literature where program predicates are assumed to be expressive enough to capture the root causes [96, 205].

(A2) If the fix corresponds to a composition of transformations, then the malfunction score achieved after applying the composition of transformations is less than the malfunc-

tion score achieved after applying any of the constituents, and all these scores are less than the malfunction score of the original dataset. For example, consider two discriminative PVTs X and Y and a failing dataset D_{fail}. Our assumption is that if $\{X, Y\}$ corresponds to a minimal explanation, then $m_S((Y_T \circ X_T)(D_{fail})) < m_S(X_T(D_{fail})) < m_S(D_{fail})$ and $m_S((Y_T \circ X_T)(D_{fail})) < m_S(Y_T(D_{fail})) < m_S(D_{fail})$. Intuitively, this assumption states that X and Y have consistent (independent) effect on reducing the malfunction score, regardless of whether they are intervened together or individually. If this assumption does not hold, DATAEXPOSER can still work with additional knowledge about multiple failing and passing datasets [112].

Observations. We make the following observations:

(O1) If the ground-truth explanation of malfunction corresponds to an attribute, then multiple PVTs that involve the same attribute are likely to differ across the passing and failing datasets. This observation motivates us to prioritize interventions based on PVTs that are associated with high-degree attributes in the PVT-attribute graph. Additionally, intervening on the data based on one such PVT is likely to result in an automatic "fix" of other PVTs connecting via the high-degree attribute. For example, adding noise to `high_expenditure` in Example 7.1 breaks its dependence with not only `race` but also with other attributes.

(O2) The PVT for which the failing dataset incurs higher violation score is more likely to be a potential explanation of malfunction.

(O3) A transformation function that affects a large number of data tuples is likely to result in a higher reduction in the malfunction score, after the transformation is applied.

PVT-attribute graph. DATAEXPOSER leverages observation O1 by constructing a bipartite graph (G_{PA}), *called PVT-attribute graph,* with all attributes $A \in \mathcal{R}(D)$ as nodes on one side and all discriminative PVTs $X \in \mathcal{X}$ on the other side. An attribute A is connected to a PVT X if and only if X_P has A as one of its parameters. E.g., Figure 7.4 shows the PVT-attribute graph w.r.t. $People_{fail}$ and $People_{pass}$ (Example 7.1). In this graph, the PVT

Algorithm 6: DATAEXPOSER$_{\text{GRD}}$ (greedy)

Input: Failing dataset D_{fail}, passing dataset D_{pass}, malfunction score threshold τ
Output: A minimal explanation set of PVTs $\mathcal{X}^*$

$\mathcal{X}_f \leftarrow$ DISCOVER-PVT(D_{fail})
$\mathcal{X}_p \leftarrow$ DISCOVER-PVT(D_{pass})
$\mathcal{X}_\cap \leftarrow \mathcal{X}_f \cap \mathcal{X}_p$ /* Common PVTs */
$\mathcal{X} \leftarrow \mathcal{X}_p \setminus \mathcal{X}_\cap$ /* Discriminative PVTs */
$G_{PA}(V_G, E_G) \leftarrow$ CONSTRUCT-PVT-ATTR-GRAPH($\mathcal{X}, D_{fail}$)
$B \leftarrow$ CALCULATE-BENEFIT-SCORE($\mathcal{X}, G_{PA}, D_{fail}$)
$\mathcal{X}^* \leftarrow \emptyset$ /* Initialize minimal explanation set to be empty */
$D \leftarrow D_{fail}$ /* Initialize dataset to the failing dataset */
while $m_S(D) > \tau$ **do**
  $\mathcal{X}_{\texttt{hda}} = \{X \in \mathcal{X} | (X, A) \in E_G \wedge A = \text{argmax}_{A \in \mathcal{R}(D)} deg_G(A)\}$ /* PVTs adjacent to high-degree attributes in G_{PA} */
  $X = \text{argmax}_{X \in \mathcal{X}_{\texttt{hda}}} B(X)$ /* Highest-benefit PVT */
  $\Delta \leftarrow m_S(D) - m_S(X_T(D))$ /* Malfunction reduction */
  $G_{PA} \leftarrow G_{PA}$.REMOVE(X) /* Update G_{PA} */
  if $\Delta > 0$ **then** /* Reduces malfunction */
    $D \leftarrow X_T(D)$ /* Apply transformation */
    G_{PA}.UPDATE(D) /* Update the PVT-attribute graph */
    B.UPDATE(D) /* Update benefit scores */
    $\mathcal{X}^* \leftarrow \mathcal{X}^* \cup \{X\}$ /* Add P to explanation set */
    $\mathcal{X} \leftarrow \mathcal{X} \setminus \{X\}$ /* Remove P from the candidates */
$\mathcal{X}^*$ = MAKE-MINIMAL($\mathcal{X}^*$) /* Obtain minimality of $\mathcal{X}^*$ */
return $\mathcal{X}^*$ /* $\mathcal{X}^*$ is a minimal explanation */

corresponding to $\langle$INDEP, `race`, `high_expenditure`$\rangle$ is connected to two attributes, `race` and `high_expenditure`. Intuitively, this graph captures the dependence relationship between PVTs and attributes, where an intervention with respect to a PVT X modifies an attribute A connected to it. If this intervention reduces the malfunction score then it could possibly fix other PVTs that are connected to A.

Benefit score computation. DATAEXPOSER uses the aforementioned observations to compute a benefit score for each PVT to model their likelihood of reducing system malfunction if the corresponding transformation is used to modify the failing dataset D_{fail}. Intuitively, it assigns a high score to a PVT with a high violation score (O2) and if the corresponding transformation function modifies a large number of tuples in the dataset (O3). Formally,

the benefit score of a PVT X is defined as the product of violation score of D_{fail} w.r.t. X_V and the "coverage" of X_T. The coverage of X_T is defined as the fraction of tuples that it modifies. Note that the procedure for benefit computation acts as a proxy of the likelihood of a PVT to offer an explanation, without actually applying any intervention.

7.3.3 Greedy Approach

Algorithm 6 presents the pseudocode of our greedy technique DATAEXPOSER$_{\text{GRD}}$, which takes a passing dataset D_{pass} and a failing dataset D_{fail} as input and returns the set of PVTs that corresponds to a minimal explanation of system malfunction.

Lines 1–2 Identify two sets of PVTs $\mathcal{X}_f$ and $\mathcal{X}_p$ satisfied by D_{fail} and D_{pass}, respectively.

Lines 3–4 Discard the PVTs $\mathcal{X}_f \cap \mathcal{X}_p$ from $\mathcal{X}_p$ and consider the remaining discriminative ones $\mathcal{X} \equiv \mathcal{X}_p \setminus \mathcal{X}_f$ as candidates for potential explanation of system malfunction.

Line 5 Compute the PVT-attribute graph G_{PA}, where the candidate PVTs $\mathcal{X}$ correspond to nodes on one side and the data attributes correspond to nodes on the other side.

Line 6 Compute the benefit score of each discriminative PVT $X \in \mathcal{X}$ w.r.t. D_{fail}. This procedure relies on the violation score using the violation function of the PVT and the coverage of the corresponding transformation function over D_{fail}.

Line 7–8 Initialize the solution set $\mathcal{X}^*$ to $\emptyset$ and the dataset to perform intervention on D to the failing dataset $\mathcal{D}_{fail}$. In subsequent steps, $\mathcal{X}^*$ will converge to a minimal explanation set and D will be transformed to a dataset for which the system passes.

Line 9 Iterate over the candidate PVTs $\mathcal{X}$ until the dataset D (which is being transformed iteratively) incurs an acceptable violation score (less than the allowable threshold τ).

Line 10 Identify the subset of PVTs $\mathcal{X}_{\texttt{hda}} \subseteq \mathcal{X}$ such that all $X \in \mathcal{X}_{\texttt{hda}}$ are adjacent to at least one of the highest degree attributes in the current PVT-attribute graph (Observation O1).

Line 11 Choose the PVT $X \in \mathcal{X}_{\texttt{hda}}$ that has the maximum benefit.

Line 12 Compute the reduction in malfunction score if the dataset D is transformed according to the transformation X_T.

Line 13 Remove X from G_{PA} as it has been explored.

Lines 14–19 If the malfunction score reduces over $X_T(D)$, then X is added to the solution set $\mathcal{X}^*$, and D is updated to $X_T(D)$, which is then used to update the PVT-attribute graph and benefit of each PVT. The update procedure recomputes the benefit scores of all PVTs that are connected to the attributes adjacent to X in G_{PA}.

Line 20 Post-process the set $\mathcal{X}^*$ to identify a minimal subset that ensure that malfunction score remains less than the threshold τ. This procedure iteratively removes one PVT at a time (say X) from $\mathcal{X}^*$ and recomputes the malfunction score over the failing dataset D_{fail} transformed according to the transformation functions of the PVTs in the set $\mathcal{X}' = \mathcal{X}^* \setminus \{X\}$. If the transformed dataset incurs a violation score less than τ then $\mathcal{X}^*$ is replaced with $\mathcal{X}'$.

7.3.4 Group-testing-based Approach

We now present our second algorithm DATAEXPOSER$_{\text{GT}}$, which performs group interventions to identify the minimal explanation that exposes the mismatch between a dataset and a system. The group intervention methodology is applicable under the following assumption along with assumptions A_1 and A_2 (Section 7.3.2).

(A3) The malfunction score incurred after applying a composition of transformations is less than the malfunction score incurred by the the original dataset if and only if at least one of the constituent transformations reduces the malfunction score. For two PVTs X and Y, $m_S((X_T \circ Y_T)(D_{fail})) < m_S(D_{fail})$, iff $m_S(X_T(D_{fail})) < m_S(D_{fail})$ or $m_S(Y_T(D_{fail})) < m_S(D_{fail})$. Note that this assumption is crucial to consider group interventions and is prevalent in the group-testing literature [80].

DATAEXPOSER$_{\text{GT}}$ follows the classical adaptive group testing (GT) paradigm [80] for interventions. To this end, it iteratively partitions the set of discriminative PVTs into two "almost" equal subsets (when the number of discriminative PVTs is odd, then the size of the two partitions will differ by one). During each iteration, all PVTs in a partition are considered for intervention *together* (group intervention) to evaluate the change in malfunction score. If a partition does not help reduce the malfunction score, all PVTs within that partition are discarded. While traditional GT techniques [80] would use a random partitioning of the PVTs, DATAEXPOSER$_{\text{GT}}$ can leverage the dependencies among PVTs (inferred from the PVT-attribute graph) to achieve more effective partitioning. Intuitively, it is beneficial to assign all PVTs whose transformations operate on the same attribute to the same partition, which is likely to enable aggressive pruning of spurious PVTs that do not reduce malfunction.

DATAEXPOSER$_{\text{GRD}}$ captures the dependencies among PVTs by constructing a PVT-dependency graph G_{PD}. Two PVTs U and V are connected by an edge in G_{PD} if they are connected via some attribute in G_{PA}. G_{PD} is equivalent to G_{PA}^2 (transitive closure of G_{PA}), restricted to PVT nodes (excluding the attribute nodes). This ensures that PVTs that are associated via some attribute in G_{PA} are connected in G_{PD}. DATAEXPOSER$_{\text{GRD}}$ partitions G_{PD} such that the number of connections (edges) between PVTs that fall in different partitions are minimized. More formally, we aim to construct two "almost" equal-sized partitions of $\mathcal{X}$ such that the number of edges between PVTs from different partitions *minimum bisection* of a graph [116]. The minimum bisection problem is NP-hard [116] and approximate algorithms exist [102, 103]. In this work, we use the *local search algorithm* [102].

We proceed to demonstrate the benefit of using DATAEXPOSER$_{\text{GT}}$ as opposed to traditional GT with the following example.

Example 7.8. *Consider a set of* 8 *PVTs* $\mathcal{X} = \{X_1, \ldots, X_8\}$ *where the ground-truth (minimal) explanation is either* $\{X_1, X_6\}$ *or* $\{X_4, X_8\}$ *(disjunction). An example of steps for a traditional adaptive GT approach is shown in Figure 7.6(c). In this case, it requires*

Algorithm 7: DATAEXPOSER$_{\text{GT}}$ (group-testing-based)

Input: Failing dataset D_{fail}, passing dataset D_{pass}, malfunction score threshold τ
Output: A minimal explanation set of PVTs $\mathcal{X}^*$

1 $\mathcal{X}_f \leftarrow$ DISCOVER-PVT(D_{fail})
2 $\mathcal{X}_p \leftarrow$ DISCOVER-PVT(D_{pass})
3 $\mathcal{X}_\cap \leftarrow \mathcal{X}_f \cap \mathcal{X}_p$ /* Common PVTs */
4 $\mathcal{X} \leftarrow \mathcal{X}_p \setminus \mathcal{X}_\cap$ /* Discriminative PVTs */
5 $G_{PA}(V_G, E_G) \leftarrow$ CONSTRUCT-PVT-ATTR-GRAPH($\mathcal{X}, D_{fail}$)
6 $D, \mathcal{X}^* \leftarrow$ GROUP-TEST($\mathcal{X}, D_{fail}, G^2_{PA}, \tau$) /* Obtain an exp. */
7 $\mathcal{X}^*$ = MAKE-MINIMAL($\mathcal{X}^*$) /* Obtain minimality of $\mathcal{X}^*$ */
8 **return** $\mathcal{X}^*$ /* $\mathcal{X}^*$ is a minimal cause */

a total of 14 *interventions. Note that adaptive GT is a randomized algorithm and this example demonstrates one such execution. However, we observed similar results for other instances. In contrast to adaptive GT,* DATAEXPOSER$_{GT}$ *constructs a min-bisection of the graph during each iteration: it does not partition* $\{X_2, X_3\}$ *and* $\{X_5, X_7\}$ *as none of these PVTs help reduce the malfunction. Therefore, it requires only* 10 *interventions.*

Algorithm 7 presents the DATAEXPOSER$_{\text{GT}}$ algorithm. It starts with a set of discriminative PVTs $\mathcal{X}$ and the PVT-attribute graph G_{PA}. All candidate PVTs are then considered by GROUP-TEST subroutine to identify the explanation $\mathcal{X}^*$.

GROUP-TEST. Algorithm 8 presents the procedure that takes the set of discriminative PVTs $\mathcal{X}$, a failing dataset D, PVT-dependency graph G_{PD}, and the malfunction score threshold τ as input. It returns a transformed (fixed) dataset and an explanation.

Line 1 Initialize the solution set $\mathcal{X}^*$ to $\emptyset$.

Lines 2–3 Return the candidate PVT set $\mathcal{X}$ if its cardinality is 1.

Line 4 Partition $\mathcal{X}$ into $\mathcal{X}_1$ and $\mathcal{X}_2$ using min-bisection of the PVT-dependency graph G_{PD}.

Line 5 Compute the malfunction score of the input dataset.

Algorithm 8: GROUP-TEST

Input: Candidate PVT $\mathcal{X}$, dataset D, PVT-dependency graph G_{PD}, malfunction score threshold τ

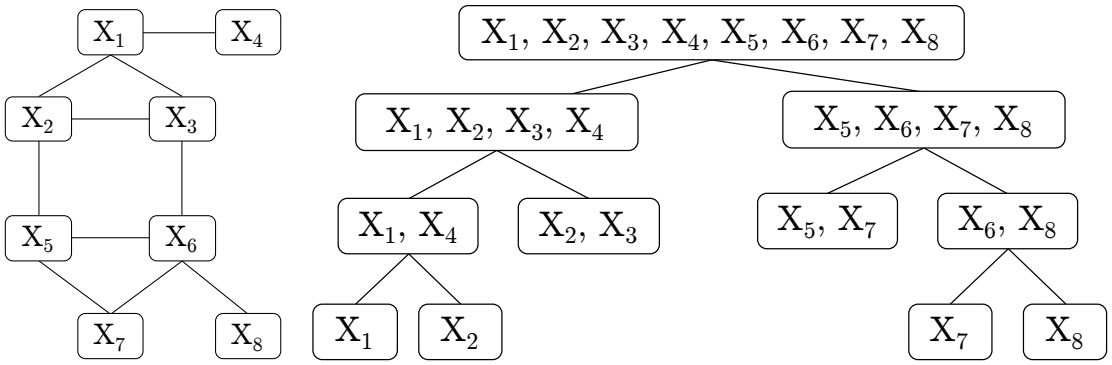
Output: A transformed dataset D' and an explanation set of PVTs $\mathcal{X}^*$

1 $\mathcal{X}^* \leftarrow \emptyset$ /* Initialize explanation set to be empty */
2 **if** $|\mathcal{X}| = 1$ **then** /* Only a single PVT is candidate */
3 **return** $T_{\mathcal{X}}(D), \mathcal{X}$
4 $\mathcal{X}^1, \mathcal{X}^2 \leftarrow$ GET-MIN-BISECTION$(G_{PD}, \mathcal{X})$ /* Partition $\mathcal{X}$ */
5 $\mathcal{M} \leftarrow m_S(D)$ /* Initial malfunction score */
6 $\Delta_1 \leftarrow \mathcal{M} - m_S(\mathcal{X}_T^1(D))$ /* Malfunction reduction by $\mathcal{X}_T^1$ */
7 **if** $\mathcal{M} - \Delta_1 > \tau$ **then** /* $\mathcal{X}^1$ alone is insufficient */
8 $\Delta_2 \leftarrow \mathcal{M} - m_S(\mathcal{X}_T^2(D))$ /* Malfunction reduction by $\mathcal{X}_T^2$ */
9 **if** $(\mathcal{M} - \Delta_1 \leq \tau)$ *OR* $(\Delta_1 > 0$ *AND* $\mathcal{M} - \Delta_2 > \tau)$ **then**
/* $\mathcal{X}_1$ is sufficient OR $\mathcal{X}_1$ helps AND $\mathcal{X}_2$ is insufficient */
10 $D, \mathcal{X}' \leftarrow$ GROUP-TEST$(\mathcal{X}_1, D, G_{PD})$
11 $\mathcal{X}^* = \mathcal{X}^* \cup \mathcal{X}'$ /* Augment explanation set */
12 **if** $\mathcal{M} - \Delta_1 \leq \tau$ **then** /* Malfunction is acceptable */
13 **return** $D, \mathcal{X}^*$ /* No need to check $\mathcal{X}_2$ */
14 **if** $\Delta_2 > 0$ **then** /* $T_{\mathcal{X}_2}$ reduces malfunction */
15 $D, \mathcal{X}' \leftarrow$ GROUP-TEST$(\mathcal{X}_2, D, G_{PD})$
16 $\mathcal{X}^* = \mathcal{X}^* \cup \mathcal{X}'$ /* Augment explanation set */
17 **return** $D, \mathcal{X}^*$

Line 6 Compute the reduction in malfunction score Δ_1 if D is intervened w.r.t. all PVTs $\mathcal{X}_1$.

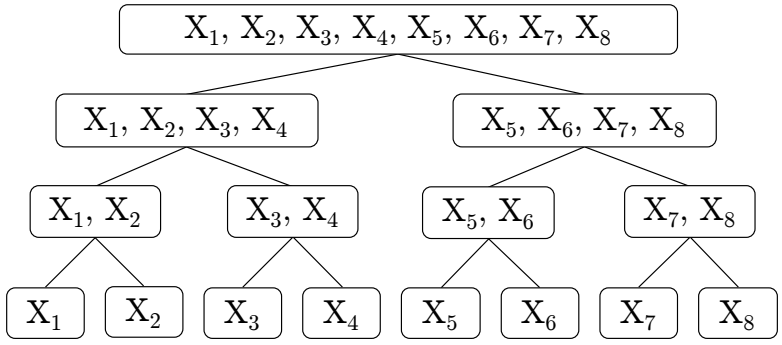
Lines 7–8 If the malfunction exceeds τ even after intervening on D w.r.t. all PVTs in $\mathcal{X}_1$ then try out $\mathcal{X}_2$: compute the reduction in malfunction score Δ_2 if D is intervened w.r.t. all PVTs in $\mathcal{X}_2$.

Lines 9–13 Recursively call GROUP-TEST on the partition $\mathcal{X}_1$ if one of the following conditions hold: (1) Intervening on D w.r.t. all PVTs in $\mathcal{X}_1$ reduces the malfunction to be lower than τ: the explanation over $\mathcal{X}_1$ is returned as the final explanation. (2) Intervening on D w.r.t. all PVTs in $\mathcal{X}_1$ reduces the malfunction, but still remains above τ, but intervening on D w.r.t. all PVTs in $\mathcal{X}_2$ brings the malfunction below τ:



(a) Dependency graph G_{PD}

(b) Execution of DATAEXPOSER$_{\text{GT}}$



(c) Execution of traditional group testing algorithm

Figure 7.6: Comparison between DATAEXPOSER$_{\text{GT}}$ and adaptive group testing on a toy example.

the explanation returned by the recursive call on $\mathcal{X}_1$ is added to the set $\mathcal{X}^*$ and $\mathcal{X}_2$ is processed next.

Lines 14–16 Recursively call GROUP-TEST on $\mathcal{X}_2$ if intervening on all PVTs in $\mathcal{X}_2$ reduces malfunction. The set of PVTs returned by this recursive call of the algorithm are added to the solution set $\mathcal{X}^*$.

Discussion on DATAEXPOSER$_{\text{GRD}}$ vs. DATAEXPOSER$_{\text{GT}}$. DATAEXPOSER$_{\text{GRD}}$ intervenes by considering a single discriminative PVT at a time. Hence, in the worst case, it requires $O(|\mathcal{X}|)$ interventions where $\mathcal{X}$ denotes the set of discriminative PVTs. Note that DATAEXPOSER$_{\text{GRD}}$ requires much fewer interventions in practice and would require

$O(|\mathcal{X}|)$ only when any of the mentioned observations (O1–O3) do not hold. In contrast, DATAEXPOSER$_{\text{GT}}$ performs group intervention by recursively partitioning the set of discriminative PVTs. Thus, the maximum number of interventions required by DATAEXPOSER$_{\text{GT}}$ is $O(t \log |\mathcal{X}|)$ where t denotes the number of PVTs that help reduce malfunction if the corresponding profile is altered. Note that, in expectation, DATAEXPOSER$_{\text{GT}}$ requires fewer interventions than DATAEXPOSER$_{\text{GRD}}$ whenever $t = o(|\mathcal{X}|/\log|\mathcal{X}|)$. DATAEXPOSER$_{\text{GT}}$ is particularly helpful when multiple PVTs *disjunctively* explain the malfunction. However, DATAEXPOSER$_{\text{GT}}$ requires an additional assumption assumption A3 (Section 7.3.4). We discuss the empirical impact of this assumption in Section 7.4.1 (Cardiovascular disease prediction). Overall, we conclude that DATAEXPOSER$_{\text{GT}}$ is beneficial for applications whenever $t = O(|\mathcal{X}|/\log|\mathcal{X}|)$ and observations O1-O3 hold.

As discussed in Section 7.3.4, DATAEXPOSER$_{\text{GRD}}$ always identifies the ground-truth cause of malfunction, but, the number of interventions required may increase if observations O1-O3 do not hold. If O1 does not hold, then the initial ordering of attributes, which is computed based on their degree in PVT-Attribute graph G_{PA}, may not be accurate. In this case, DATAEXPOSER$_{\text{GRD}}$ may require $O(\mathcal{X})$ interventions to explain the cause of system malfunction. Similarly, whenever O2 or O3 fail, the benefit-based ordering of the PVTs is likely to be incorrect. In applications where O1 holds but O2 and O3 do not hold, G_{PA} is expected to be accurate and the number of required interventions is $O(r)$, where r is the degree of the attribute that has the highest degree in G_{PA}. In contrast, DATAEXPOSER$_{\text{GT}}$ requires $O(t \log |\mathcal{X}|)$ interventions even when these observations are violated. Therefore, DATAEXPOSER$_{\text{GRD}}$ is beneficial whenever all of these observations hold, or $t = \Omega(|\mathcal{X}|/\log|\mathcal{X}|)$.

7.4 Experimental Evaluation

Our experiments involving DATAEXPOSER aim to answer the following questions: (Q1) Can DATAEXPOSER *correctly* identify the cause and corresponding fix of mismatch

	Number of Interventions				
Application	DATAEXPOSER$_{GRD}$	DATAEXPOSER$_{GT}$	BugDoc	Anchor	GrpTest
Sentiment	2	3	10	303	3
Income	1	8	20	800	10
Cardiovascular	1	NA	100	5900	NA

Figure 7.7: Comparison of number of interventions of DATAEXPOSER with other baselines. NA denotes that the technique could not identify the cause of malfunction because assumption A3 did not hold.

	Execution Time (seconds)				
Application	DATAEXPOSER$_{GRD}$	DATAEXPOSER$_{GT}$	BugDoc	Anchor	GrpTest
Sentiment	25.1	23.4	64.6	4594.9	21.2
Income	11.8	12.5	20.0	195.5	10.4
Cardiovascular	7.6	NA	62.1	8602.9	NA

Figure 7.8: Comparison of running time of DATAEXPOSER with other baselines. NA denotes that the technique could not identify the cause of malfunction because assumption A3 did not hold.

between a system and a dataset for which the system fails? (Q2) How *efficient* is DATAEXPOSER compared to other alternative techniques? (Q3) Is DATAEXPOSER *scalable* with respect to the number of discriminating PVTs?

Baselines. Since there is no prior work on modifying a dataset according to a PVT, we adapted state-of-the-art debugging and explanation techniques to incorporate profile transformations and explain the cause of system failure. We consider three baselines: (1) `BugDoc` [216] is a recent debugging technique that explores different parameter configurations of the system to understand its behavior. We adapt `BugDoc` to consider each PVT as a parameter of the system and interventions as the modified configurations of the pipeline. (2) `Anchor` [277] is a local explanation technique for classifiers that explains individual predictions based on a surrogate model. We train `Anchor` with PVTs as features, and the prediction variable is Pass/Fail where Pass (Fail) denotes the case where an input dataset incurs malfunction below (above) τ. In this technique, each intervention creates a

new data point to train the surrogate model. (3) `GrpTest` [80] is an adaptive group testing approach that performs group interventions to expose the mismatch between the input dataset and the system. It is similar to DATAEXPOSER$_{GT}$ with a difference that the recursive partitioning of PVTs is performed randomly without exploiting the PVT-dependency graph.

7.4.1 Real-world Case Studies

We design three case studies focusing on three different applications, where we use well-known ML models [4, 10, 269] as black-box systems. For all of the three case studies, we use real-world datasets. Figures 7.7 and 7.8 presents a summary of our evaluation results.

Sentiment Prediction. The system in this study predicts sentiment of input text (reviews/tweets) and computes misclassification rate as the malfunction score. It uses flair [10], a pre-trained classifier to predict sentiment of the input records and assumes a `target` attribute in the input data, indicating the ground truth sentiment: A value of 1 for the attribute `target` indicates positive sentiment and a value of -1 indicates negative sentiment. We test the system over `IMDb` dataset [156] ($50K$ tuples) and a `twitter` dataset [291] (around 1.6M tuples). The malfunction score of the system on the `IMDb` dataset is only 0.09 while on the `twitter` dataset it is 1.0. We considered `IMDb` as the passing dataset and `twitter` as the failing dataset and used both DATAEXPOSER$_{GRD}$ and DATAEXPOSER$_{GT}$ to explain the mismatch between the `twitter` dataset and the system. The ground-truth cause of the malfunction is that the `target` attribute in the `twitter` dataset uses "4" to denote positive and "0" to denote negative sentiment [291]. DATAEXPOSER$_{GRD}$ identifies a total of 3 discriminative PVTs between the two datasets. One such PVT includes the profile DOMAIN of the `target` attribute that has corresponding parameter $\mathbb{S} = \{-1, 1\}$ for `IMDb` and $\mathbb{S} = \{0, 4\}$ for the `twitter` dataset. DATAEXPOSER$_{GRD}$ performs two interventions and identifies that the malfunction score reduces to 0.36 by

mapping $0 \to -1$ and $4 \to 1$ by intervening w.r.t. the PVT corresponding DOMAIN, which is returned as an explanation of the malfunction.

DATAEXPOSER$_{\text{GT}}$ and `GrpTest` both require 3 group interventions to explain the cause of system malfunction. `BugDoc` and `Anchor` require 10 and 303 interventions, respectively. `Anchor` computes system malfunction on datasets transformed according to various local perturbations of the PVTs in the failing dataset.

Income Prediction. The system in this study trains a Random Forest classifier [269] to predict the income of individuals while ensuring fairness towards marginalized groups. The pipeline returns the normalized disparate impact [152] of the trained classifier w.r.t. the protected attribute (`sex`), as the malfunction score. Our input data includes census records [81] containing demographic attributes of individuals along with information about income, education, etc. We create two datasets through a random selection of records, and manually add noise to one of them to break the dependence between `target` and `sex`. The system has malfunction score of 0.195 for the passing dataset and 0.58 for the failing dataset due to the dependence between `target` and `sex`. DATAEXPOSER$_{\text{GRD}}$ identifies a total of 43 discriminative PVTs and constructs a PVT-attribute graph. In this graph, the `target` attribute has degree 15 while all other attributes have degree 2. The PVTs that include `target` are then intervened in non-increasing order of benefit. The transformation w.r.t. INDEP PVT on the `target` attribute breaks the dependence between `target` and all other attributes, thereby reducing the malfunction score to 0.32. Therefore, DATAEXPOSER$_{\text{GRD}}$ requires one intervention to explain the cause of the malfunction. Our group testing algorithm DATAEXPOSER$_{\text{GT}}$ and `GrpTest` require 8 and 10 interventions, respectively. Note that group testing is not very useful because the datasets contain few discriminative PVTs.

`BugDoc` and `Anchor` do not identify discriminative PVTs explicitly and consider all PVTs (136 for this dataset) as candidates for intervention. `Anchor` performs 800 local interventions to explain the malfunction. `BugDoc` identifies the ground truth malfunction

in 50% of the runs when allowed to run fewer than 10 interventions. It identifies the mismatch with intervention budget of 20 but the returned solution of PVTs is not minimal. For instance, `BugDoc` returns two PVTs: $\{\langle$INDEP, `target`, `education`$\rangle$ `and` $\langle$INDEP, `target`, `sex`$\rangle\}$ as the explanation of malfunction.

Cardiovascular Disease Prediction. This system trains an AdaBoost classifier [4] on patients' medical records [52] containing age, height (in centimeters) and weight along with other attributes. It predicts if the patient has a disease and does not optimize for false positives. Therefore, the system computes recall over the patients having cardiovascular disease, and the goal is to achieve more than 0.70 recall. The pipeline returns the additive inverse of recall as the malfunction score. We tested the pipeline with two datasets generated through a random selection of records: (1) the passing dataset satisfies the format assumptions of the pipeline; (2) for the failing dataset we manually converted `height` to inches. DATAEXPOSER$_{\text{GRD}}$ identifies 86 discriminative PVTs with `height`, `weight` and `age` having the highest degree of 15 in the PVT-attribute graph. Among the PVTs involving these attributes, the DOMAIN of `height` has the maximum benefit, which is the ground-truth PVT too. DATAEXPOSER$_{\text{GRD}}$ alters the failing dataset by applying a linear transformation and it reduces the malfunction from 0.71 to 0.30. This explanation matches the ground truth difference in the passing and the failing dataset. Among baselines, `BugDoc` and `Anchor` performed 100 and 5900 interventions, respectively. Group testing techniques are not applicable because assumption A3 (Section 7.3.4) does not hold. We observe that the malfunction score with a composition of transformation functions is higher than the one in the original dataset if the composition involves INDEP PVT. This behavior is observed because adding noise to intervene with respect to INDEP PVT worsens the classifier performance. If we remove PVTs that violate this assumption, then DATAEXPOSER$_{\text{GT}}$ and `GrpTest` require 6 and 9 interventions, respectively.

Efficiency. Figure 7.8 presents the execution time of considered techniques for real-world applications presented above. DATAEXPOSER$_{GRD}$, DATAEXPOSER$_{GT}$ and `GrpTest` are highly efficient and require less than 30 seconds to explain the ground-truth cause of malfunction. In contrast, `Anchor` is extremely inefficient as it requires more than 143 minutes for `cardiovascular`, while DATAEXPOSER$_{GRD}$ and `BugDoc` explain the malfunction within 63 seconds.

Key takeaways. Among all real-world case studies, the greedy approach DATAEXPOSER$_{GRD}$ requires the fewest interventions to explain the cause of malfunction. Group testing techniques, DATAEXPOSER$_{GT}$ and `GrpTest`, require fewer interventions than `BugDoc` and `Anchor` whenever assumption A3 (Section 7.3.4) holds. `Anchor` requires the highest number of interventions as it performs many local transformations to explain the cause of failure. `BugDoc` optimizes interventions by leveraging combinatorial design algorithms: it requires more interventions than DATAEXPOSER but fewer than `Anchor`.

7.4.2 Synthetic Pipelines

We evaluate the effectiveness and scalability of DATAEXPOSER$_{GRD}$ and DATAEXPOSER$_{GT}$ for a diverse set of synthetic scenarios.

DATAEXPOSER$_{GRD}$ vs. DATAEXPOSER$_{GT}$. In this experiment, we consider a pipeline where the ground-truth explanation of malfunction violates the observations discussed in Section 7.3.2. Specifically, the explanation requires modifying one particular value in the dataset and its likelihood (as estimated by DATAEXPOSER$_{GRD}$) is ranked 54 among the set of discriminative PVTs. Therefore, it requires 54 interventions to explain the cause of malfunction. On the other hand, DATAEXPOSER$_{GT}$ performs group interventions and requires only 9 interventions. This experiment demonstrates that DATAEXPOSER$_{GT}$ requires fewer interventions than DATAEXPOSER$_{GRD}$ when the failing dataset and the corresponding PVTs do not satisfy the observations DATAEXPOSER$_{GRD}$ relies on.

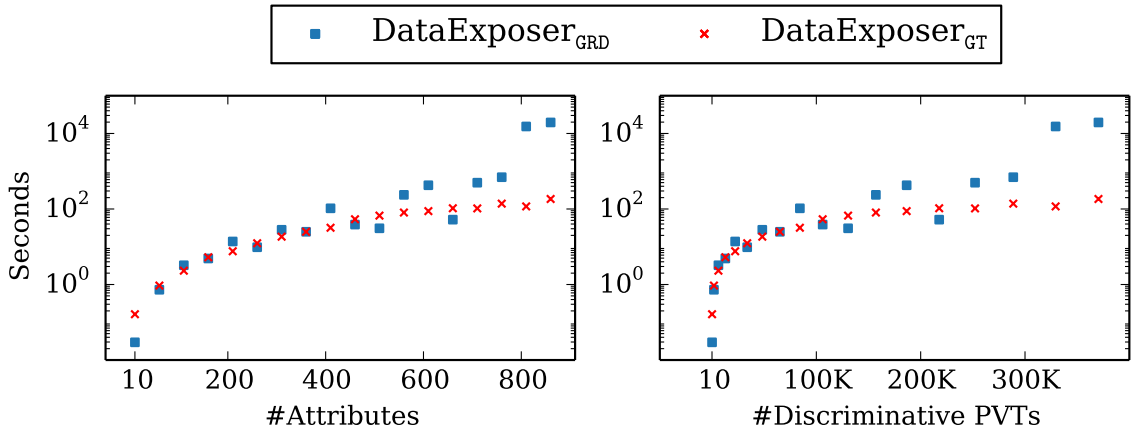


Figure 7.9: Execution time of DATAEXPOSER$_{GRD}$ vs. DATAEXPOSER$_{GT}$ with varying number of data attributes (left) and discriminative PVTs (right) over synthetic pipelines.

Scalability. To test the scalability of our techniques, we compare their running time with increasing number of attributes and discriminative PVTs. Figure 7.9 shows that the time required by DATAEXPOSER$_{GRD}$ and DATAEXPOSER$_{GT}$ to explain the malfunction grows sub-linearly in the number of attributes and discriminative PVTs. We observe similar trend of the number of required interventions on varying these parameters. This experiment demonstrates that DATAEXPOSER$_{GRD}$ requires fewer than $O(|\mathcal{X}|)$ interventions in practice (where $\mathcal{X}$ denotes the set of discriminative profiles) and validates the logarithmic dependence of DATAEXPOSER$_{GT}$ on $|\mathcal{X}|$.

7.4.3 Effect of Various Parameters

In this experiment, we test the effect of the number of dataset attributes and the number of discriminative PVTs on the efficacy of DATAEXPOSER algorithms, and contrast those with other state-of-the-art baselines. We also investigate the influence of the number of PVTs involved in the root causes and their interactions on the number of interventions each method requires.

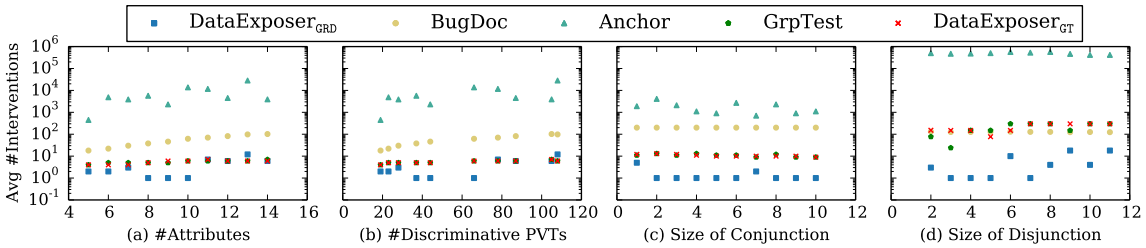


Figure 7.10: Average number of interventions required by two versions of DATAEXPOSER and three other techniques for varying number of attributes, discriminative PVTs, size of single conjunctive root causes, and size of disjunctive root causes.

7.4.3.1 Effect of Number of Attributes and PVTs

Figure 7.10(a) presents the effect of changing the number of attributes in the datasets on the number of required interventions. DATAEXPOSER$_{GRD}$ requires fewer than 5 interventions on average. In contrast, `BugDoc` and `Anchor` require orders of magnitude more interventions. The number of interventions required by `BugDoc` grows linearly with the number of attributes. At the same time, `Anchor` perturbs all PVTs to solve a multi-armed bandit problem: the more PVTs affect the pipeline errors, the more interventions are needed. Group-testing-based approaches also require more intervention than that of DATAEXPOSER$_{GRD}$, and grow logarithmically with the number of data attributes.

Figure 7.10(b) depicts the effect of the number of discriminative PVTs on the number of required interventions. DATAEXPOSER$_{GRD}$ shows superior performance as it requires fewer than 10 interventions even when the number of discriminative PVTs go beyond 100. Here, we observe trends similar the one in Figure 7.10(a) for other baselines as the number of PVTs are strongly and positively correlated to the number attributes.

7.4.3.2 Effect of Number of Root Causes and their Interactions

The pipelines presented in Figures 7.10(a) and 7.10(b) have a single PVT as the root cause of the malfunction. In Figure 7.10(c), we fix the number of attributes to 15 and the number of discriminative PVTs between the passing and the failing datasets to 136. We also define the root cause to be a conjunction over a set of PVTs of varying cardinalities.

We find that the cardinality of the root-cause set (length of the conjunctive cause) does not impact the number of interventions as much as the number of attributes and the number of discriminative PVTs do. However, having more than one cause for malfunction (i.e., a disjunctive cause) requires more interventions for `Anchor` and the Group-Test techniques, as shown in Figure 7.10(d). DATAEXPOSER$_{\text{GRD}}$ still needs orders of magnitude fewer interventions than these other approaches, although the probability of failing to find any feasible transformation, which decreases malfunctions scores, increases with the number of possible root causes within the disjunction.

7.5 Summary and Future Work

We introduced the problem of identifying causes and fixes of mismatch between data and systems that operate on data. To this end, we presented DATAEXPOSER, a framework that reports violation of data profiles as causally verified root causes of system malfunction and reports fixes in the form of transformation functions. We demonstrated the effectiveness and efficacy of DATAEXPOSER in explaining the reason of mismatch in several real-world and synthetic data-driven pipelines, significantly outperforming the state of the art. In future, we want to extend DATAEXPOSER to support more complex classes of data profiles. Additionally, we plan to investigate ways that can facilitate automatic repair of both data and the system guided by the identified data issues.

CHAPTER 8

CONCLUSIONS AND FUTURE DIRECTIONS

In summary, this thesis targets fundamental problems in data platforms and analytics: example-driven techniques complement traditional task specification mechanisms in various data platforms; conformance constraints have a broad range of applications, including trusted machine learning; explanation frameworks offer support in debugging various components of complex data systems. The results of this work are foundational for data systems to become more accessible and transparent to a much broader set of users in all data platforms. The work presented in this thesis has inspired other work in various areas such as data systems democratization [168], querying relational databases without SQL [22, 267], use of causality in debugging [189, 191] and so on.

There are many possible directions one can follow from the work presented here. Particularly, providing support in debugging complex data-driven system is still an underdeveloped area of research. The main challenge in black-box data-driven systems is lack of interpretability and explainability, which, in turn, makes debugging these systems difficult. There is also a growing concern related to trust and fairness in data-driven systems, as systems are being more and more data-dependent. Finally, enhancing usability of data systems, and systems in general, remains a challenging task for both the systems and HCI community. We hope the work presented in this thesis will help in the areas of enhancing usability, understandability, and explainability of a wide variety of systems, leading towards democratization of systems for common good.

Appendix

APPENDIX A

EXAMPLE-DRIVEN INTENT DISCOVERY

A.1 SQUID: Contrast with Prior Work

We provide a summary of prior work in the data management community to contrast with SQUID in the comparison matrix of Figure A.1. We proceed to explain the comparison metrics and highlight the key differences among different classes of query by example techniques and their variants. We organize the prior work into three categories: QBE (query by example), QRE (query reverse engineering), and DX (data exploration). Furthermore, we group QBE methods into two sub-categories, methods for relational databases, and methods for knowledge graphs. All QRE and DX methods that we discuss are developed on relational databases. We first describe our comparison metrics:

- *Query class* encodes the expressivity of a query. We use four primitive SQL operators (join, projection, selection, and aggregation) as comparison metrics. While data retrieval mechanisms (e.g., graph query, SPARQL) for knowledge graphs do not directly support all these operators, they support similar expressivity through alternative equivalent operators.

- *Semi-join* is a special type of join which is particularly useful for QBE systems. A system is considered to support semi-join if it allows inclusion of relations in the output query that have no attribute projected in the input schema (e.g., in Example 3.1, no attribute of `research` appears in the input tuples, but `Q2` includes `research`). While knowledge-graph-based systems do not directly support semi-join as defined in the relational database setting, they support same expressivity through alternative mechanism.

Legend			query class								
QBE: Query by Example QRE: Query Reverse Engineering DX: Data Exploration KG: Knowledge Graph ◐: with significant restrictions			join	projection	selection	aggregation	semi-join	implicit property	scalable	open-world	additional requirements
QBE	relational	**SQUID**	●	●	●	●	●	●	●	●	
		Bonifati et al. [44]	●	●	◐		●		●	●	user feedback
		QPlain [72]	●	●	●		●		●	●	provenance input
		Shen et al. [294]	●	●					●	●	
		FASTTOPK [260]	●	●					●	●	
	KG	Arenas et al. [14]	●	●	◐		●		●	●	
		SPARQLByE [73]	●	●	◐		●		●	●	negative examples
		GQBE [160]	●	●	◐		●		●	●	
		QBEES [232]	●	●	◐		●		●	●	
QRE	relational	PALEO-J [250]	●	●	●	●			●		top-k queries only
		SQLSynthesizer [346]	●	●	●	●	●				schema knowledge
		SCYTHE [322]	●	●	●	●	●				schema knowledge
		Zhang et al. [345]	●						●	●	
		REGAL [307]		●	●	●			●		
		REGAL+ [308]	●	●	●	●					
		FASTQRE [175]	●	●					●	●	
		QFE [203]	●	●	●						user feedback
		TALOS [312]	●	●	●	●	●		●		
DX	rel.	AIDE [74]			●				●	●	user feedback
		REQUEST [117]			●				●	●	user feedback

Figure A.1: SQUID captures complex intents and more expressive queries than prior work.

- *Implicit property* refers to the properties that are not directly stated in the data (e.g., number of comedies an actor appears in). In SQUID, we compute implicit properties by aggregating direct properties of affiliated entities.

- *Scalability* characterizes how the system scales when data increases. While deciding on scalability of a system, we mark a system scalable only if it either had a rigorous scalability experiment, or was shown to perform well on real-world big datasets. Thus, we do not consider approaches as scalable if the dataset is too small (e.g., contains 100 cells).

- *Open-world* assumption states that what is not known to be true is simply unknown. In QBE and related work, if a system assumes that tuples that are not in the examples are not necessarily outside of user interest, it follows the open-world assumption. In contrast, closed-world assumption states that when a tuple is not specified in the user example, it is definitely outside of user interest.

- Apart from the aforementioned metrics, we also report any *additional requirement* of each prior art. We briefly discuss different types of additional requirements here: *User feedback* involves answering any sort of system generated questions. It ranges from simply providing relevance feedback (yes/no) to a system-suggested tuple to answering complicated questions such as "if the input database is changed in a certain way, would the output table change in this way?" Another form of requirement involves providing *negative tuples* along with positive tuples. *Provenance input* requires the user to explain the reason why they provided each example. Some systems require the user to provide the example tuples sorted in a particular order (top-k), aiming towards reverse engineering top-k queries. *Schema-knowledge* is assumed when the user is supposed to provide provenance of examples or sample input database along with the example tuples.

A.2 Proof of Theorem 3.1

Proof. We prove Theorem 3.1 by contradiction. Suppose that φ is the optimal set of filters, i.e., Q^{φ} is the most likely query. Additionally, suppose that φ is the minimal set of filters for obtaining such optimality, i.e., $\nexists \varphi'$ such that $|\varphi'| < \varphi \wedge Pr(Q^{\varphi'} \mid E) = Pr(Q^{\varphi} \mid E)$. Now suppose that, Algorithm 1 returns a sub-optimal query $Q^{\varphi'}$, i.e., $Pr(Q^{\varphi'} \mid E) < Pr(Q^{\varphi} \mid E)$. Since $Q^{\varphi'}$ is suboptimal, $\varphi' \neq \varphi$; therefore at least one of the following two cases must hold:

Case 1: $\exists \phi$ such that $\phi \in \varphi \wedge \phi \notin \varphi'$. Since Algorithm 1 did not include ϕ, it must be the case that $\texttt{include}_{\phi} \leq \texttt{exclude}_{\phi}$. Therefore, we can exclude ϕ from φ to obtain $\varphi - \{\phi\}$ and according to Equation 3.5, $Pr(Q^{\varphi - \{\phi\}} \mid E) \geq Pr(Q^{\varphi} \mid E)$ which contradicts with our assumption about the optimality and minimality of φ.

Case 2: $\exists \phi$ such that $\phi \notin \varphi \wedge \phi \in \varphi'$. Since Algorithm 1 included ϕ, it must be the case that $\texttt{include}_{\phi} > \texttt{exclude}_{\phi}$. Therefore, we can add ϕ to φ to obtain $\varphi \cup \{\phi\}$ and

ID	Task	J	S	#Result
IQ1	Entire cast of Pulp Fiction	3	1	113
IQ2	Actors who appeared in all of The Lord of the Rings trilogy	8	7	20
IQ3	Canadian actresses born after 1970	3	4	1531
IQ4	Sci-Fi movies released in USA in 2016	5	3	1374
IQ5	Movies Tom Cruise and Nicole Kidman acted together	5	2	12
IQ6	Movies directed by Clint Eastwood	4	2	36
IQ7	All movie genres	1	0	35
IQ8	Movies by Al Pacino	4	2	71
IQ9*	Indian actors who acted in at least 15 Hollywood movies	6	4	23
IQ10*	Actors who acted in more than 10 Russian movies after 2010	6	4	84
IQ11	Hollywood Horror-Drama movies in 2005 – 2008	7	5	291
IQ12	Movies produced by Walt Disney Pictures	3	1	394
IQ13	Animation movies produced by Pixar	5	2	57
IQ14	Sci-Fi movies acted by Patrick Stewart	6	3	22
IQ15	Japanese Animation movies	5	2	2512
IQ16*	Walt Disney Pictures movies with more than 15 American cast members	5	3	207

* Includes `GROUP BY` and `HAVING` clauses

Figure A.2: Benchmark queries for the IMDb dataset. **J** and **S** denote the number of joins and selection predicates, respectively.

ID	Task	J	S	#Result
DQ1	Authors who collaborated with both U Washington and Microsoft Research Redmond	5	2	30
DQ2*	Authors with at least 10 SIGMOD and at least 10 VLDB publications	8	4	52
DQ3	SIGMOD publications in 2010 – 2012	3	3	468
DQ4	Publications Jiawei Han, Xifeng Yan, and Philip S. Yu published together	7	3	15
DQ5	Publications between USA and Canada	5	2	336

* Includes `GROUP BY`, `HAVING`, and `INTERSECT`

Figure A.3: Benchmark queries for the DBLP dataset. **J** and **S** denote the number of joins and selection predicates, respectively.

according to Equation 3.5, $Pr(Q^{\varphi \cup \{\phi\}} \mid E) > Pr(Q^{\varphi} \mid E)$ which again contradicts with our assumption about the optimality of φ.

Hence, $Q^{\varphi'}$ cannot be suboptimal and this implies that Algorithm 1 returns the most likely query. □

In a special case where $\texttt{include}_{\phi} = \texttt{exclude}_{\phi}$, Algorithm 1 drops the filter using Occam's razor principle to keep the query as simple as possible. However, this does not return any query that is strictly less likely than the best query.

SQL Query	#Result
SELECT DISTINCT name FROM adult WHERE education = 'Bachelors' AND occupation = 'Craft-repair' AND hoursperweek >= 36 AND hoursperweek <= 40 AND age >= 46 AND age <= 47	8
SELECT DISTINCT name FROM adult WHERE race = 'White' AND sex = 'Female' AND nativecountry = 'United-States' AND relationship = 'Other-relative' AND occupation = 'Machine-op-inspct' AND workclass = 'Private'	11
SELECT DISTINCT name FROM adult WHERE occupation = 'Craft-repair' AND workclass = 'Private' AND age >= 65 AND age <= 68 AND relationship = 'Husband'	12
SELECT DISTINCT name FROM adult WHERE maritalstatus = 'Divorced' AND capitalgain >= 7298 AND capitalgain <= 10520 AND hoursperweek >= 40 AND hoursperweek <= 44 AND relationship = 'Not-in-family'	14
SELECT DISTINCT name FROM adult WHERE capitalgain >= 4101 AND capitalgain <= 4650 AND workclass = 'Private' AND age >= 41 AND age <= 44	14
SELECT DISTINCT name FROM adult WHERE occupation = 'Protective-serv' AND hoursperweek >= 45 AND hoursperweek <= 48	44
SELECT DISTINCT name FROM adult WHERE education = '10th' AND race = 'White' AND fnlwgt >= 334113 AND fnlwgt <= 403468	48
SELECT DISTINCT name FROM adult WHERE nativecountry = 'United-States' AND hoursperweek >= 43 AND hoursperweek <= 45 AND race = 'White' AND fnlwgt >= 106541 AND fnlwgt <= 118876	126
SELECT DISTINCT name FROM adult WHERE race = 'White' AND education = 'Bachelors' AND nativecountry = 'United-States' AND capitalgain >= 6097 AND capitalgain <= 7688 AND maritalstatus = 'Married-civ-spouse' AND relationship = 'Husband'	128
SELECT DISTINCT name FROM adult WHERE education = 'Bachelors' AND capitalloss >= 1848 AND capitalloss <= 1980	182

Figure A.4: First 10 benchmark queries for the Adult dataset.

A.3 Benchmark Queries, Source of Datasets, and Additional Dataset Details for Evaluation SQuID

Figures A.2– A.5 show the benchmark queries used to evaluate SQuID. For obtaining a downsized database `sm-IMDb`, we dropped persons with less than 2 affiliated movies and/or who have too many semantic information missing, and movies that have no cast information. We produced two upsized databases: one with dense associations `bd-IMDb`, and the other with sparse associations `bs-IMDb`. The database `bd-IMDb` contains duplicate entries for all movies, persons, and companies (with different primary keys), and the associations among persons and movies are duplicated to produce more dense associations. For example, if person `P1` appeared in movie `M1` in IMDb, i.e., `(P1,M1)` exists in IMDb's

SQL Query	#Result
SELECT DISTINCT name FROM adult WHERE sex = 'Male' AND nativecountry = 'United-States' AND capitalloss >= 1848 AND capitalloss <= 1887	203
SELECT DISTINCT name FROM adult WHERE education = 'Doctorate' AND maritalstatus = 'Married-civ-spouse' AND nativecountry = 'United-States'	223
SELECT DISTINCT name FROM adult WHERE education = 'HS-grad' AND workclass = 'Private' AND hoursperweek >= 45 AND hoursperweek <= 46 AND relationship = 'Husband'	241
SELECT DISTINCT name FROM adult WHERE capitalgain >= 7688 AND capitalgain <= 8614	343
SELECT DISTINCT name FROM adult WHERE education = 'Bachelors' AND maritalstatus = 'Never-married' AND workclass = 'Private' AND hoursperweek >= 40 AND hoursperweek <= 43 AND race = 'White'	563
SELECT DISTINCT name FROM adult WHERE education = 'HS-grad' AND nativecountry = 'United-States' AND occupation = 'Machine-op-inspct' AND race = 'White'	777
SELECT DISTINCT name FROM adult WHERE nativecountry = 'United-States' AND age >= 60 AND age <= 62	798
SELECT DISTINCT name FROM adult WHERE fnlwgt >= 271962 AND fnlwgt <= 288781	912
SELECT DISTINCT name FROM adult WHERE maritalstatus = 'Married-civ-spouse' AND fnlwgt >= 221366 AND fnlwgt <= 259301	1340
SELECT DISTINCT name FROM adult WHERE maritalstatus = 'Never-married' AND fnlwgt >= 185624 AND fnlwgt <= 211177	1404

Figure A.5: Last 10 benchmark queries for the Adult dataset.

`castinfo`, we added a duplicate person `P2`, a duplicate movie `M2`, and 3 new associations, `(P1,M2)`, `(P2,M2)`, and `(P2,M1)`, to `bd-IMDb`'s `castinfo`. For `bs-IMDb`, we only duplicated the old associations, i.e., we added `P2` and `M2` in a similar manner, but only added `(P2,M2)` in `castinfo`.

A.4 SQuID Parameters

We empirically evaluate how the parameters of SQuID contribute to its performance. Figure A.9 lists the four most important parameters of the system, along with brief descriptions. We proceed to discuss our observations regarding the impact of these parameters.

ρ. The base filter prior parameter ρ defines SQuID's tendency towards including filters. Small ρ makes SQuID less likely to include a filter, which prevents overfitting and favors recall. In contrast, large ρ makes SQuID more likely to include a filter, which may result in overfitting but will result in higher precision. A low ρ helps eliminate coincidental filters—which prevents overfitting—particularly with very few example tuples. However,

Title	Source
IMDb dataset	https://datasets.imdbws.com/
DBLP dataset	https://data.mendeley.com/datasets/3p9w84t5mr
Adult dataset	https://archive.ics.uci.edu/ml/datasets/adult
Physically strong actors	https://www.imdb.com/list/ls050159844/
Top 1000 Actors and Actresses*	http://www.imdb.com/list/ls058011111/
Sci-Fi Cinema in the 2000s	http://www.imdb.com/list/ls000097375/
Funny Actors	https://www.imdb.com/list/ls000025701/
100 Random Comedy Actors	https://www.imdb.com/list/ls000791012/
BEST COMEDY ACTORS	https://www.imdb.com/list/ls000076773/
115 funniest actors	https://www.imdb.com/list/ls051583078/
Top 35 Male Comedy Actors	https://www.imdb.com/list/ls006081748/
Top 25 Funniest Actors Alive	https://www.imdb.com/list/ls056878567/
the top funniest actors in hollywood today	https://www.imdb.com/list/ls007041954/
Google knowledge graph: Actors: Comedy	https://www.google.com/search?q=funny+actors
The Best Movies of All Time*	https://www.ranker.com/crowdranked-list/the-best-movies-of-all-time
Top H-Index for Computer Science & Electronics*	http://www.guide2research.com/scientists/

Figure A.6: Source of datasets and lists used in this work. * denotes the lists that are used as popularity mask.

with sufficient example tuples, coincidental filters are naturally removed, as with more examples, it is unlikely to keep observing a coincidental similarity. Therefore, the effect of ρ diminishes, i.e., any value of ρ works just as well, when the number of examples is large. Figure A.10(a) shows the effect of varying the value of ρ for a few benchmark queries on the IMDb dataset. While a low ρ favors some queries (e.g., `IQ2` and `IQ16`), it causes accuracy degradation—due to underfitting or overgeneralization—for other queries (e.g., `IQ3`, `IQ4`, and `IQ11`), particularly when the number of examples are very few. A high ρ works better for `IQ3`, `IQ4`, and `IQ11`; but with very few example tuples, it overfits for other queries (e.g., `IQ2`).

In general, there is a natural tradeoff between overfitting (high value for ρ) and overgeneralization (low value for ρ), and we found empirically that a moderate value of ρ (e.g., 0.1) works best on average, especially with small number of examples. When we observe sufficient examples, then all ρ values eventually converge to the same level of accuracy.

γ. The domain coverage penalty parameter γ specifies SQuID's leniency towards filters with large domain coverage. Low γ reduces the penalty on filters with large domain coverage, and a high γ increases the penalty. Figure A.10(b) shows the effect of varying γ. Very low value for γ favors some queries (`IQ3`, `IQ4`, `IQ11`) but also causes accuracy degradation for some other queries (`IQ2`, `IQ16`), where a high γ works better. Like ρ, it is

IMDb & variations				
IMDb			bd-IMDb	
	DB size	633 MB	DB size	1926 MB
	#Relations	15	#Relations	15
	Precomputed DB size	2310 MB	Precomputed DB size	5971 MB
	Precomputation time	150 min	Precomputation time	370 min
	`person`	6,150,949	`person`	12,301,898
Relation	`movie`	976,719	`movie`	1,953,438
Cardinality	`castinfo`	14,915,325	`castinfo`	59,661,300
bs-IMDb			sm-IMDb	
	DB size	1330 MB	DB size	75 MB
	#Relations	15	#Relations	15
	Precomputed DB size	4831 MB	Precomputed DB size	317 MB
	Precomputation time	351 min	Precomputation time	14 min
	`person`	12,301,898	`person`	65,865
Relation	`movie`	1,953,438	`movie`	335,705
Cardinality	`castinfo`	29,830,650	`castinfo`	1,364,890

Figure A.7: Description of different variations of the IMDb dataset.

	DBLP		Adult	
	DB size	22 MB	DB size	4 MB
	#Relations	14	#Relations	1
	Precomputed DB size	98 MB	Precomputed DB size	5 MB
	Precomputation time	42 min	Precomputation time	3 min
	`author`	126,094		
Relation	`publication`	148,521	`adult`	32,561
Cardinality	`authortopub`	416,445		

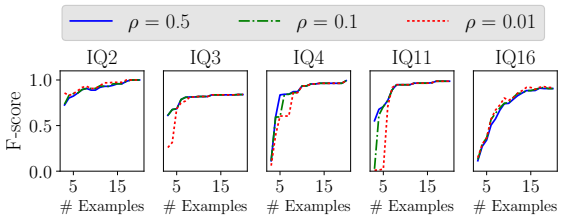
Figure A.8: Description of the DBLP and the Adult datasets.

also a tradeoff, and, empirically, we found moderate values of γ (e.g., 2) to work well on average.

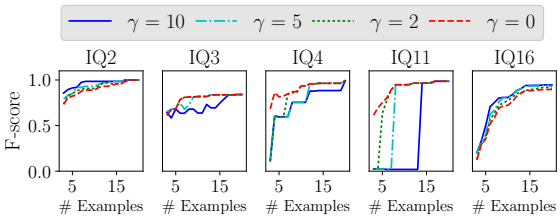
τ_a. The association strength threshold τ_a is required to define the association strength impact $\alpha(\phi)$ (Section 3.3.2.2). Figure A.10(c) illustrates the effect of different values of τ_a on the benchmark query `IQ5` on the IMDb dataset. With very few example tuples, high τ_a is preferable, since it helps discard filters with coincidentally strong associations. Similar to other parameters, with increased number of example tuples, the effect of τ_a diminishes, as it is unlikely for a coincidental filter to survive the association strength threshold if it is not intended after all. In general, it is advised to not set τ_a to a very high value as some intended similarities might have weak associations (e.g., find all actors who *occasionally*

Parameter	Default Value	Description
ρ (Base filter prior parameter)	0.1	Controls SQuID's tendency towards including a filter.
γ (Domain coverage penalty parameter)	2	Penalizes filters with large domain coverage.
τ_a (Association strength threshold)	5	Eliminates filters with weak associations with other entities.
τ_s (Skewness threshold)	2.0	Identifies skewed distributions, which contain interesting filters as outliers.

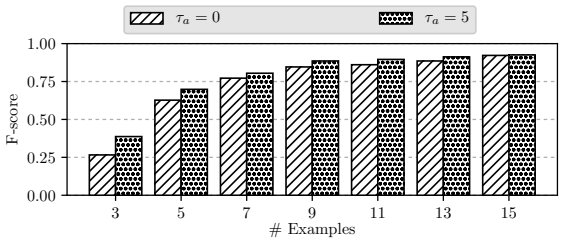
Figure A.9: List of SQuID parameters with description.



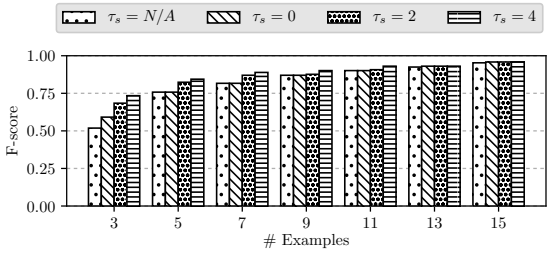
(a) Effect of different values for ρ.



(b) Effect of different values for γ.



(c) Effect of different values for τ_a for `IQ5` (IMDb).



(d) Effect of different values for τ_s for `IQ1` (IMDb).

Figure A.10: Effect of system parameters on SQuID's performance for a few benchmark queries over the IMDb dataset.

appear in comedies); in such cases, having a very high value of τ_a will discard the desired filter with a relatively weak association strength. Ultimately, with sufficient examples, even the lowest possible value for τ_a, which is 0, would work just as well.

τ_s. The skewness threshold τ_s is required to classify an association strength distribution as skewed or not (Section 3.3.2.2). Figure A.10(d) illustrates the effect of different values of τ_s on the benchmark query `IQ1` (Find the entire cast of the movie `Pulp Fiction`) on the

IMDb dataset. $\tau_s = N/A$ refers to the experiment where outlier impact was not taken into account (i.e., $\lambda(\phi) = 1$ for all filters). In this query, there were a number of unintended derived filters involving `movie certificate` (e.g., PG, G, R, X, etc.) for different countries, and a high τ_s helped SQuID get rid of those. We also found a high τ_s to be very useful when we could not use a high τ_a due to the nature of the query intent that requires filters with low association strengths (e.g., `IQ3`). However, too high value for τ_s is also not desirable, since it will underestimate some moderately skewed distributions and discard intended filters. Empirically, we found that moderate τ_s (e.g., 2 and 4) to work well on average.

Key Takeaway. The key takeaway from the experiments on parameter sensitivity is that SQuID's dependency on parameters diminishes as the number of examples increases. One interesting point to note is that unlike other approaches that require hyperparameter tuning for specific workloads (e.g., machine learning models), SQuID does not depend heavily on the parameter values and its dependency on the parameters diminishes as the number of examples increases. The purpose of gauging SQuID's sensitivity to different parameter values is to demonstrate how the specific parameter values have little to no impact even with a moderate number of example tuples (e.g., 15). Nevertheless, this analysis presents a guideline on how to tune SQuID's parameters when the workload is expected to contain only a small number of example tuples for each user intent, where overfitting must be prevented to allow for generalization. Otherwise, it is safe to set all the parameters to values that encourage overfitting when sufficient example tuples can be expected.

A.5 Study Design

A.5.1 Dataset and Baseline

In our comparative user studies, we studied how users perceive SQuID, compared to the traditional SQL querying mechanism, over a variety of subjective and objective data exploration tasks.

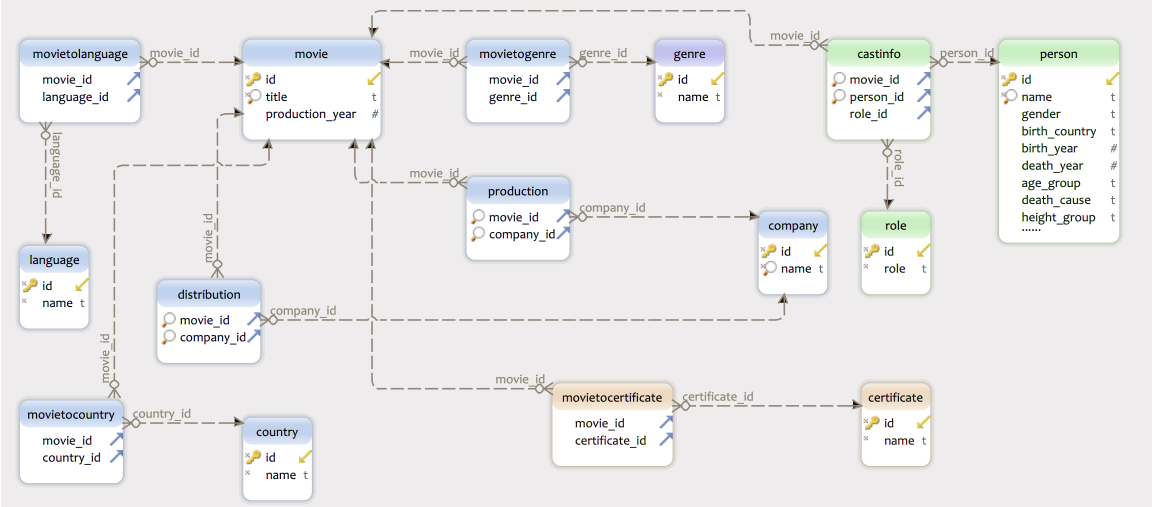


Figure A.11: Complete schema of the IMDb database with 8 main relations: `movie`, `person`, `genre`, `language`, `country`, `company`, `role`, and `certificate`; and 7 connecting relations that associate the main relations: `castinfo`, `movietogenre`, `movietolanguage`, `distribution`, `movietocountry`, `movietocertificate`, and `production`.

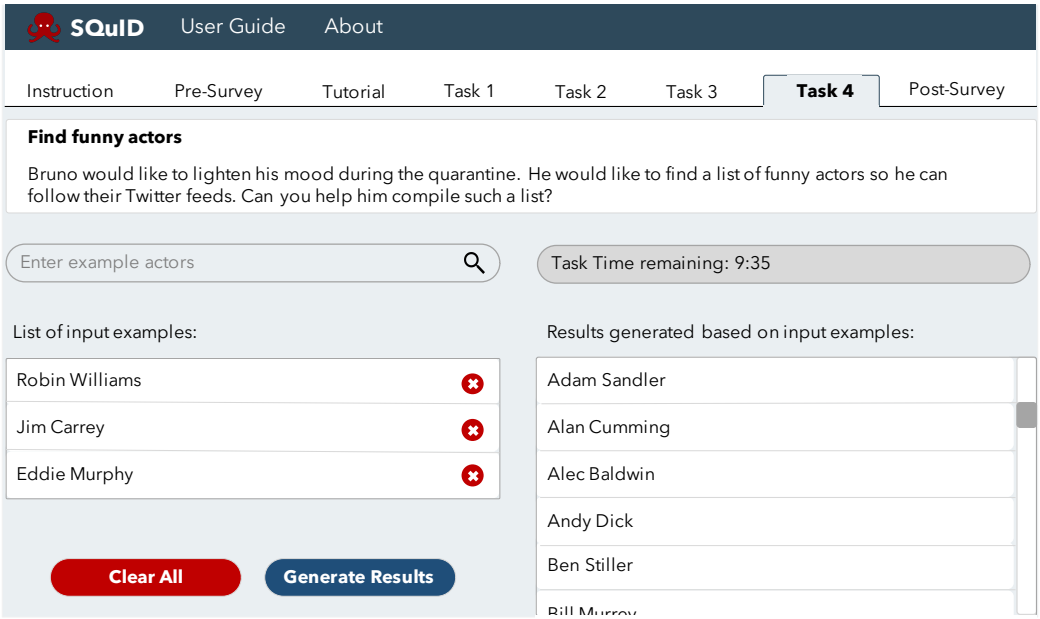


Figure A.12: The graphical user interface of SQUID used in our user study. The task description is at the top. The left panel allows the users to provide examples with an autocompletion feature. SQUID infers the user's intended query from the examples, executes it, and shows the results in the right panel. We only show the first five results (alphabetically).

We now provide an overview of the dataset we used in our studies and a brief description of SQL, the baseline which we compare SQUID against.

A.5.2 Dataset

For our comparative user studies, our goal was to emulate data exploration tasks in a controlled experiment setting. Generally, people explore data they are interested in and within a domain they are somewhat familiar with. Moreover, data exploration with QBE expects some basic domain familiarity, as users need to be able to provide examples. Therefore, our goal in selecting a dataset was to identify a domain of general interest, where most study participants can be expected to have a basic level of domain familiarity. Furthermore, the dataset needs to be sufficiently large to emulate the practical challenges that users face during data exploration. We selected the IMDb database (discussed in Section 3.6.1), which satisfies these goals, as the IMDb website has over 83 million registered users and about 927 million yearly page visits.[1]

A.5.3 Structured Query Language (SQL)

The traditional way to query a relational database is to write a query in structured query language (SQL). SQL is one of the most widely used programming languages for handling structured data (54.7% developers use SQL [214]). It is specifically designed to query relational databases and has been used for over 50 years. SQL is a declarative query language and is primarily based on relational algebra. The SQL language consists of several elements such as clauses, expressions, predicates, statements, integrity constraints, etc. SQL has been implemented by different developers—such as Oracle, Microsoft SQL, MySQL, PostgreSQL, etc.—slightly differently; however, fundamentally, they all work the same way. For our comparative user studies, we picked PostgreSQL, which is a free and open-source relational database management system.

[1]IMDb.com Analytics: `www.similarweb.com/website/imdb.com/`

Relational databases usually organize data in a *normalized* form, to avoid redundancy. This is in contrast with the flat data format where all attributes of an entity are stored together within the same row. For example, the detailed schema of the IMDb database, split in 15 relational tables, is shown in Figure A.11. Here, the relation `movie` contains only three attributes about movies: a numerical record `id` (called *primary key*), a text attribute specifying the `title` of the movie, and the `production year` of the movie. However, information about associated genres of a movie is not present in the `movie` table. To figure out the genres of a movie, one would need to write a SQL query to `JOIN` the tables `movie`, `movietogenre`, and `genre`. The query would also need to specify the *logic* behind this join, i.e., which rows in the `genre` table are relevant to a particular movie in the `movie` table. SQL is a relatively simple language with a limited set of operators (e.g., `SELECT`, `PROJECT`, `JOIN`, etc.). While this simplicity enables the users to learn quickly how to express easy intents using SQL (e.g., the SQL query `SELECT title FROM movie` would retrieve all movie titles), it comes at the cost that complex intents are hard to express in SQL. Specifically, the restrictions in the data organization (normalized schema) and the simplicity of the SQL operators make complex tasks harder to translate in SQL: it requires the users to specify the entire data retrieval logic. Overall, writing a successful SQL query for a data exploration tasks requires several skills: (1) familiarity with the database schema, (2) understanding of the table semantics, (3) understanding of the SQL operators, (4) knowledge of the SQL syntax, and (5) expertise in translating task intents to SQL.

A.5.4 Study Design

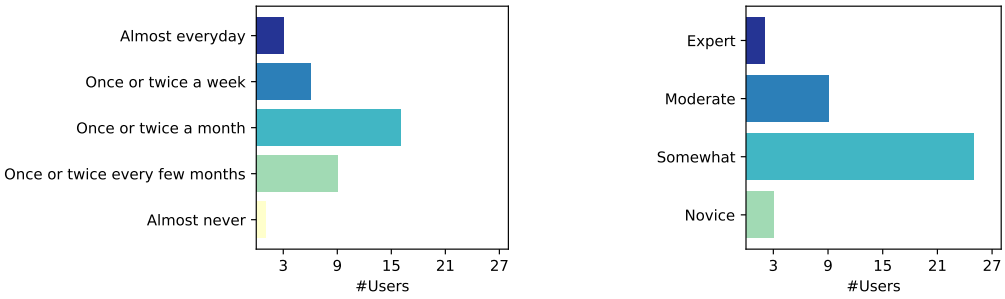
In our user studies, our goal was to quantitatively compare the efficacy and efficiency of SQuID and SQL over a variety of data exploration tasks, while also gathering qualitative feedback from users regarding their experiences with the systems. To this end, we opted for two separate comparative user studies: (1) a controlled experiment study, with a fixed set of tasks, over a group of participants of sufficient size to support quantitative evaluation; (2) an

interview study, with a flexible set of tasks, over a small group to gather qualitative user feedback. Due to the situation caused by the ongoing COVID-19 pandemic, both studies were conducted online: the controlled experiment was conducted through a website, hosted on our university servers, and the interview study was conducted over Zoom.

For both studies, we provided the database schema (Figure A.11) and a graphical user interface with a text box, where the participants could write SQL queries to interact with a PostgreSQL database. For SQUID, we provided a graphical user interface to allow the participants to interact with the system (Figure A.12). We now proceed to describe the settings, design choices, and methods of our comparative user studies. We first describe our controlled experiment study over a user group of 35 participants, followed by our interview study with a smaller group of 7 interviewees.

A.5.4.1 Study 1: Controlled Experiment Study

Participants. For our controlled experiment study, we recruited students who were enrolled in an undergraduate computer science course on Data Management Systems at our university during the Spring 2020 semester. The course offers an introduction to data management systems and the SQL language. This ensured that our study participants would have basic familiarity with SQL, which is required to compare the two systems: SQUID and SQL. We invited all 89 students enrolled in this course to take part in the study and 35 of them agreed to participate. We offered extra credit for study participation; students who opted to not participate were given alternative opportunities for extra credit. We labeled these participants P1–P35. The average grade the participants achieved in the course was 86.3 (out of 100), with a minimum grade of 45, and a maximum grade of 100; the standard deviation of the grades was 9.87. This indicates a broad range in our participants' SQL skills, which was one of our goals. While all of them had prior experience and exposure, some had only very basic skills (and failed the class) and some achieved advanced skills.



(a) Movie-watching frequency. **(b)** Overall knowledge of movies and actors.

Figure A.13: Domain knowledge of the participants.

The distribution of self-reported movie-watching frequency among the participants of our controlled-experiment study is shown in Figure A.13(a), with the most common response being 'once or twice a month', followed by 'once or twice every few months'. The responses regarding actor and movie familiarity are summarized in Figure A.13(b): a vast majority of the participants (25 out of 35) reported that they were 'somewhat' familiar with movies and actors. This validates our choice of the IMDb database for conducting the study, as indeed, we observed sufficient domain knowledge among the participants. Regarding SQL expertise, all 35 participants reported being very familiar with easy SQL queries and 34 reported being very familiar with moderately complex SQL queries. When asked regarding familiarity with complex SQL queries, 27 participants reported being very familiar, 6 were unsure, and 2 were not familiar.

Tasks and task-assignment mechanism. We designed 4 data exploration tasks over the IMDb database. Our goal was to observe what challenges a set of diverse tasks poses to the participants and how the challenges vary based on the subjectivity of the tasks and the mechanism (SQUID or SQL) used to solve the tasks. To this end, we designed two objective tasks: (1) to find *Disney* movies and (2) to find *Marvel* movies; and two subjective tasks: (1) to find *funny* actors and (2) to find *strong* and muscular actors. We provided a detailed description for each task to the study participants. Each participant was assigned all of the four tasks in the sequence: Disney, Marvel, funny, and strong. This

order was enforced to ensure that they perform easier, objective tasks first, and then move to more complex, subjective tasks. Our randomized task-system pairings made sure that for each task, about half of the participants use SQUID while the other half use SQL. For each user, we randomized which system (SQUID or SQL) they are allowed to use for each task. Everyone did the tasks—Disney, Marvel, funny, strong—in that order, but there were two possible system assignment orders: (a) SQUID, SQL, SQUID, SQL, or (b) SQL, SQUID, SQL, SQUID. Each participant was randomly given one of these assignments. This resulted in randomized task-system pairings, with the constraint that each participant must solve one objective and one subjective task using SQL and the remaining two tasks (also one objective and one subjective) using SQUID. This mechanism also eliminated any potential order bias with respect to the treatment system as half of the participants interacted with SQUID before SQL, while the other half interacted with SQL before SQUID. Each participant used either SQUID or SQL to solve each task, but never both.

Study procedure. This study was conducted online and the participants took the study over the Internet on a specific website, hosted on our university servers. We sent out the URL of the website during recruitment. At the beginning of the study, participants were asked a series of questions about their familiarity with SQL. The questions asked the participants to provide answers using a 5-point Likert-scale ranging from "Not familiar (1)" to "Very familiar (5)". Next, there was a question asking them at what frequency they watch movies, followed by a questions about overall movie and actor familiarity where participants could select multiple options. After this survey, participants were given an interactive tutorial, which was divided into two sections, walking them through the steps to obtain results with both SQUID and SQL. The tutorial took about 2–5 minutes to complete. After the tutorial, the participants started the tasks. They had 10 minutes for each task, but could finish before the time was up if they chose. Participants were asked to avoid using Internet search, but if they did, they were encouraged to report it. After each task, the participants were asked to answer a post-task survey with two questions:

the first one was about the difficulty of the task where the participants had to provide answers using a 5-point Likert-scale ranging from "Very difficult (1)" to "Very easy (5)"; and the second one was about their satisfaction with the results where the participants had to provide answers using a 5-point Likert-scale ranging from "Very unsatisfied (1)" to "Very satisfied (5)". After completing all four tasks, the participants were asked to answer four survey questions: the first one was regarding their preferences between SQUID and SQL where the participants had to provide answers using a 5-point Likert-scale ranging from "Definitely SQL (1)" to "Definitely SQUID (5)"; the second one was about usability comparison between SQL and SQUID where the participants had to provide answers using a 5-point Likert-scale ranging from "SQL was a lot easier (1)" to "SQUID was a lot easier (5)"; the third one was about satisfaction with results obtained using SQUID where the participants had to provide answers using a 5-point Likert-scale ranging from "Very unsatisfied (1)" to "Very satisfied (5)"; and the fourth one was about accuracy of the results obtained using SQL where the participants had to provide answers using a 5-point Likert-scale ranging from "Very inaccurate (1)" to "Very accurate (5)".

Data collection and analysis. During the study, we collected all survey responses and all inputs the participants provided to the systems. Specifically, for SQL, we collected all their queries, including any intermediate queries that they used to reach their final query; for SQUID, we collected all the examples they provided, along with the revision history (addition or removal of examples). We stored all this information in JSON format. During our data analysis, we extracted the JSON data programmatically through Python scripts and implemented custom functions to programmatically analyze the data. To quantitatively evaluate the tasks performed by the participants, we compared their results against the ground-truth results. We collected the ground-truth data from publicly available lists on the IMDb website. For the objective tasks (Marvel and Disney), we determined the ground truth by selecting one list for each. For the subjective tasks (funny and strong), we compiled a list by combining seven different lists for each. We selected lists that meet the following

Interviewee ID	Gender	Country of origin	Program level	SQL expertise	Area of specialization
I1	Female	Greece	2nd year PhD	Medium	Data management
I2	Male	India	3rd year PhD	Low	Natural language processing
I3	Male	Hong Kong	2nd year MS	High	Systems
I4	Female	China	5th year PhD	High	Data privacy
I5	Male	India	4th year PhD	High	Theory and data management
I6	Female	Japan	2nd year PhD	Medium	Data privacy
I7	Male	USA	4th year PhD	High	Data privacy

Figure A.14: Demographic and experience details of the interviewees who participated in our interview study.

criteria: (1) they have a number of entries that is representative of the task (e.g., there are more than five Marvel movies, thus the list should contain more than five entries), (2) they are frequently viewed, and (3) they contain entries that match the task objectives. For instance, we collected a list of 300 funny actors, which was compiled from 7 shorter lists of funny actors. One of these lists, titled "Funny Actors", has over 400,000 views, and includes 60 well-known comedians including Jim Carrey, Robin Williams, Eddie Murphy, Mel Brooks, and Will Ferrell [110].

A.5.4.2 Study 2: Interview Study

We conducted a comparative interview study to gain richer insights on users' behavior, their preferences, and any issues they faced while solving the data exploration tasks using both systems.

Interviewees. We recruited 7 interviewees for this study by targeting a diverse set of computer science graduate students directly working or collaborating with the data management research lab at our university. Out of the 7 interviewees, 4 were male and 3 were female; 6 of them were international students; and their ages ranged from 25 to 30 years old. All of them had experience using SQL for at least one year; however, their expertise varied from moderate to expert. We label the interviewees I1–I7. We provide details on the interviewees who participated in our interview study in Figure A.14.

Tasks. For this study, we asked the interviewees to pick one objective task from the following list: (1) Disney movies, (2) Marvel movies, (3) animation movies, (4) sci-fi

movies, (5) action movies, (6) movies by an actor of their own choice, or (7) movies by a country of their own choice. We also asked them to select one subjective task form this list: (1) funny actors, (2) physically strong actors, or (3) serious actors. The variety of tasks allowed interviewees to pick tasks based on their interests and enabled us to observe how the two systems compare over a variety of data exploration tasks. This study was within-subject, i.e., all of the interviewees were required to use both the mechanisms (SQUID and SQL) to solve each task.

Study procedure. For each interview, two of our research team members were present, one as primary to lead the interview and ask questions and another as secondary to take notes and ask potential follow-up questions. At the beginning of the study, we provided them the URL of the study website over the chat feature of Zoom. During the study, the interviewees first completed an interactive tutorial and then they were asked to pick two tasks. The interviewees were then asked to solve each task using both SQUID and SQL, so that they can directly contrast the two systems. We asked them to complete each task first using SQUID and then using SQL, so that the examples they would provide while using SQUID would be free from biases due to observing the results from their SQL query outputs. We did not expose the query that SQUID generates through the SQUID interface, and, thus, avoided biases when the interviewees were completing the SQL tasks. The interviewees followed a think-aloud protocol and shared their screen over Zoom during the study. They were observed by two interviewers who also asked open-ended questions to the interviewees on completion of each of the two tasks using both systems. The questions were intended to gather information on which of the two systems the interviewees prefer, under what circumstances they prefer one over the other, and the justification of why they do so. They were also asked what challenges they faced while using the systems and whether some particular task exacerbated these challenges. Finally, they were asked what type of results they prefer during data exploration: specific or generic.

Data collection and analysis. We conducted the interviews over a video conferencing tool and recorded the sessions. The 7 interviews summed to 467 minutes. On average, each interview lasted about 67 minutes, with the shortest interview lasting 43 minutes and the longest one lasting 77 minutes. Upon completion, we replayed the interview recordings, manually transcribed the responses, and stored them as plain text in a spreadsheet, resulting in 119 responses in total. We thematically analyzed the responses using our coding software (spreadsheet). Two independent coders from our team independently coded the data. The following six themes emerged after several rounds of analysis: (1) struggle in task understanding, (2) struggle in familiarizing oneself with the schema while using SQL, (3) difficulties with writing syntactically correct SQL queries, (4) struggle with solving vague/subjective tasks using SQL, (5) struggle due to lack of domain familiarity while using SQUID, and (6) preference between precision and recall of the results. There was a 98% match between the two researchers' theme assignments.

A.6 Detailed Qualitative Feedback from Interview Study

We now report the results of our interview study and describe six main themes that emerged from our analysis.

Studying the schema is challenging, even for SQL experts. All seven of our interviewees from the interview study commented that it was difficult to become acquainted with the database schema. "As a user, I have to *explore* the schema", I1 said. I1 continued, "The query itself was not complicated. It was time consuming to get familiar with the schema itself. Even for experienced users, reading through the schema and getting acquainted to [it] ...takes time." When asked about the comparison in difficulty between writing the SQL query and understanding the schema, I3 said "Looking at the schema diagram was harder. I kept going back and forth trying to understand it." Understanding the schema may be complicated not only because it can be difficult to learn what keys connect the tables,

but also because it may be hard to interpret the structure of the individual tables. I5 said, "I think it was pretty hard because I was not sure where to look for comedy based on actors. I was thinking that [the] Role [table] might have the attribute, but it didn't. Then I had to go through joining five tables!"

SQL requires stricter syntax, which makes writing queries difficult. All interviewees struggled to a varying degree to write a SQL query because of different issues; e.g., some of them could not figure out the correct spelling of attributes. For instance, one queried for the genres 'scifi' and 'comedic', neither of which exist in the database. I4 said, "The difficult part was to get the accurate predicate for the query, and I had to [explore the database] for that." SQL requires strict string matching, which can be extremely difficult to overcome for someone who is unfamiliar with the database constants and SQL syntax. While it is possible to query a table and view its content to see how the names are spelled, very few interviewees did this. It appears that the ability to write a SQL query is based on experience and recent exposure to SQL. Interviewees noted that they do not use SQL on a daily basis—some even said they had not used SQL in months—thus, it was difficult to recall specific syntactic rules. For instance, two of the interviewees—who had relatively lower SQL expertise—could not remember the requirements for joining tables. I7 had to use Google to help with this syntax, and I2 did not recall that SQL could join more than two tables. I5 said, "I was making a lot of mistakes about where to have the underscores, where to not have underscores, and for those things I had to look through the [schema] multiple times." An interesting note, I6 spent the vast majority (over 9 minutes) attempting to find the name 'Japan' in the database, and spent less than 1 minute writing the actual query. SQUID reduces the need to recall exact spelling by providing an auto-completion feature as the user types examples. Although it does not provide an auto-correct if the name is misspelled, the auto-completion feature allows the users to type what they know and scroll through the suggestions until they find the intended name. We observed several of our interviewees initially spelled a movie name incorrectly, but they were helped by the

auto-completion feature. For example, I2 initially typed 'Spiderman' in the search bar, but the title is spelled 'Spider-Man' in the database. I2 was able to correct the spelling when he typed 'Spider' in the search box and autocomplete showed the entire title. The search bar also helped I5 who noted, "If I was missing some spellings, there were some suggestions."

SQL requires parameter tuning for subjective tasks; SQuID alleviates this. Some exploration tasks can be subjective and inherently vague, e.g., defining a "funny" actor *precisely*. How many comedies, exactly, does an actor have to star in before they are considered funny? These questions have no clear answers, and such parameters can vary from person to person and from day to day. In practice, it may be very difficult, if not impossible, to think of objective measures for a subjective concept, which makes subjective tasks very complicated to specify with SQL. I2 said, "Even if I forget about syntax ... figuring out how to go about writing the pseudocode query for funny actors [is difficult]". One of the most common blunders of interviewees who used SQL to find "funny" actors was to query all actors who had been in some comedy movie. I3 was the first to acknowledge this. "I had to play around with a lot of smaller queries," he said, "to get the one that I eventually had, which I was still not satisfied with. It seems like I pulled many actors and actresses that happened to be in some comedy." I3 elaborated, "Vague tasks are generally a lot more open to interpretation. Coding up a query that meets someone's vague specifications [is] hard ... It was very hard to nail down what the correct definition of funny is." I4 also recognized that vague tasks are difficult to define. She even said, "This probably isn't a query that I should write in SQL!" She continued, "strong and muscular are very vague descriptors, and SQL needs clear rules. I have to use genre as a proxy, and that makes the query very nasty."

On the other hand, SQuID can interpret complex parameters without any involvement from the user, sparing them the mental burden of defining and implementing a complex query. I4 also said, "In order to write a SQL query, you need to understand the schema well, know your data well, and know your question well ... But if the task is exploratory

and you only have a vague idea in mind, like 'strong actors' . . . it would be very hard, if not impossible, to write a SQL query." Indicating how SQuID helped in the subjective tasks, I3 said "SQuID is a lot more user-oriented. You could just put in some actor names and it would infer what you really want."

SQuID produces precise results, which is preferred for data exploration. We asked interviewees whether they would prefer a long list that includes all relevant names, but may also include many irrelevant names (high-recall) or a shorter list that includes exclusively relevant names with very few irrelevant names, but may miss some relevant names (high-precision). Six out of seven interviewees reported that they would prefer having a shorter list with higher precision, while one interviewee had no preference. "I think I'm okay with not having all Marvel movies listed here," I2 said, "but I definitely don't want anything outside of Marvel movies. It's fine that [the results] are missing some Thor movies. I wouldn't have liked it if there were movies from DC [Comics] in here." Comparing the SQL results to the SQuID results, I5 said, "I think the [SQuID] results were not too few but not too many. It was easily understandable, and I could actually see if these were actors I was looking for . . . The [SQL] results were just too many, and most of the names I didn't know, so it was not easy to find the names that I was looking for." I6 said, "I prefer a shorter list because if there are too many movies listed, then probably, it would be overwhelming and I could not say if the results are right."

SQuID's interactivity helps users to enrich examples. Three interviewees mentioned that the results produced by SQuID helped them think of more examples in an iterative process. I6, who struggled to think of examples, was able to think of only three sci-fi movies, but when she saw 'Avatar' in the list of results, many other ideas came to her mind. Even if the intermediate results (the first or second round of results generated) were not all intended, some of them were useful in reminding the interviewees of relevant examples. For instance, I2 said, "SQuID was [nice] because it was slightly interactive. I could look

at the results and update my examples." During a task, I7 said, "[The results are] useful because now I can use Guardians of the Galaxy." I7 later added, "I think when I gave the first few examples, it gave me some results and that helped me think of more that I was looking for, and it eventually did complete the task." SQUID's results reminded the interviewees of examples that had not been on their mind, but were nonetheless relevant. I3 said, "I saw the movie Transformers, and that's something I had in my mind, but it did not occur to me when I was entering the examples. There were a bunch of other movie names [like that]." Since SQUID can provide serendipitous, but helpful, intermediate results, the user's lack of domain familiarity can still be alleviated to some extent.

Domain familiarity is crucial to evaluate the results, for both SQUID and SQL. Since SQUID requires a basic familiarity with the domain, for those who struggle to think of even one relevant example, like I6, SQUID presents a unique challenge. All interviewees could easily think of a few examples that fit the task, but they struggled beyond that. I7 said, "It was very easy to come up with two or three, but the more examples I had to give the harder it became". Two interviewees suggested that SQUID adopt an interactive system where it would ask the user whether or not a particular result was relevant on a case-by-case basis. This could alleviate some of the difficulty of thinking of relevant examples.

Furthermore, users who possess very little knowledge of the domain may be unable to recognize the results, and thus would be incapable of verifying them. But this is true for both SQUID and SQL. It was not uncommon for the interviewees to tell us that they could hardly recognize the names in the results, especially for SQL. I1, for instance, said, "Honestly, I don't recognize any of the results." This, apparently, was partly due to the large number of results returned by SQL, where there is a high chance that there will be unfamiliar names. Most people are only familiar with a relatively small subset of actors, rather than the entirety of the IMDb database. This made it difficult for the users to evaluate the results produced by both SQUID and SQL.

A.6.1 Discussion

We summarize significant findings from the quantitative and qualitative analysis of our comparative user studies below:

SQUID alleviates SQL pain points: schema complexity, semantic translation, and syntax. From our interviews, we identified three key pain points of the traditional SQL querying mechanism, all of which are removed when using SQUID:

Schema complexity. One significant difficulty that we observed during the use of SQL was the requirement of schema understanding. To issue a SQL query over a relational database, the user must first familiarize themselves with the database schema [22, 294]. The schema is often complex, like the IMDb schema shown in Figure A.11, and understanding it requires significant effort. The user also needs to correctly specify the constant values (e.g., `Comedy` and not `Comedic`), name of the relations (e.g., `movietogenre` and not `movie_to_genre`), and name of the attributes (e.g., `id` and not `movie_id`) in the SQL query. Moreover, some attributes reside in the main relation (e.g., `person.name`) while others reside in a different relation (e.g., names of a movie's genres reside in the relation `genre` and not in the relation `movie`). From a closer look at some of the user-issued SQL queries, we observed futile efforts to guess keywords, incorrectly trying values such as "comedic", "superhero comics", and "funny", which do not exist in the database and result in syntax or semantic errors. In structured databases, if one does not know the exact keywords, they end up issuing an incorrect SQL query, which returns an empty result. In contrast, SQUID frees the user from this overhead as it leverages the database content and schema and associates it automatically with the user-provided examples.

Semantic translation. After studying the schema, the next task was to translate the task's semantics formally to a language (e.g., SQL) that computational systems understand. While this is relatively easy for objective tasks (e.g., finding all movies produced by Disney), the same is not true for subjective tasks (e.g., finding all "funny" actors). As our

qualitative feedback indicates, expressing subjective or vague tasks is hard in any formal language, not only in SQL. For example, for the task of finding all "funny" actors, even the SQL experts struggled to encode the concept "funny" in SQL. Many participants wrote a SQL query to retrieve all actors who appeared in at least one movie whose genre is `Comedy`. However, upon observing the output of such an ill-formed query, they were not satisfied with the results. This is because appearing in only one comedy movie does not necessarily make an actor funny. Usually, actors who appear in "many" comedy movies are considered funny. The key struggle here is to figure out what is the right threshold for "many", i.e., in *how many* comedy movies should an actor appear to be considered "funny". In contrast, SQuID is able to discover these implicit constants from the user-provided examples. For retrieving funny actors, SQuID learns from the user-provided examples what is the usual number of comedy movies all the example actors appeared in, and subsequently, uses that number to define the notion of "many". For instance, for Example 3.3, SQuID inferred that appearing in 40 comedy movies is sufficient for an actor to be considered funny. This parameter (40) was automatically inferred based on the user-provided examples: SQuID automatically discovered that each example actor appeared in 40 or more comedy movies in the IMDb database.

Language syntax. SQL is a programming language with several operators and keywords, and similar to all programming languages, SQL also requires strict syntax. While issuing a SQL query, even a minor syntactic error will result in complete failure and will return no result. Moreover, the syntax error messages that the SQL engine provides are often ambiguous and confusing to novice users. We observed that one of our interviewees could not recall the correct syntax of the `JOIN` operation. This stringent requirement of syntax poses significant hurdles to novice and even intermediate SQL users. In contrast, SQuID completely bypasses SQL, eliminating this challenge.

SQuID is generally more effective than SQL and boosts efficiency. In our controlled experiments, we noted that SQuID is generally more effective than SQL in deriving ac-

curate results. For objective tasks, we found that SQUID outperforms SQL in all three correctness metrics—precision, recall, and F1 score. However, it is important to highlight that our interviewees noted that SQUID is particularly useful and preferable to SQL for *subjective* tasks. This does not contradict our quantitative analysis. While SQL has higher recall than SQUID for subjective tasks, SQUID achieves much higher F1 scores, because SQL's precision for these tasks is close to 0. This is because an extremely general SQL query (e.g., one that returns all the data) may have very high recall, but it will not be useful for the exploration task that expects targeted results. Furthermore, SQUID significantly boosts the user's efficiency in data exploration. This was confirmed by our controlled experiment study where we found that participants achieved their goal much faster (in about 200 fewer seconds) and with less effort (with about 4 fewer attempts) while using SQUID compared to SQL.

SQUID's pain point and remedies: lack of domain expertise. Lack of domain knowledge is a handicap for SQUID, as it requires at least a few initial examples for its inference. This is a general issue with all query-by-example mechanisms [124, 294]. However, even when the user lacks domain knowledge, they can use alternative mechanisms—such as keyword search, Internet search, or very basic SQL queries (when the user has some SQL familiarity)—to come up with some initial examples. In contrast, when a user does not know SQL, learning it from scratch takes significant time and effort. While SQUID's by-example paradigm can help both expert and novice users alike, in general, programming-by-example systems are most beneficial when domain knowledge outweighs technical knowledge and experience [284]; otherwise, a hybrid system is more desirable. However, lack of domain knowledge is a problem for SQL as well. Without basic knowledge over the data domain (e.g., what are the entities and what are their properties), understanding the schema can be harder. Furthermore, without sufficient domain knowledge, debugging SQL queries, i.e., validating whether the user-issued SQL queries are correct or not, based on the results, is also challenging.

SQuID promotes serendipitous discovery, aiding users in data exploration. SQuID is *interactive* in a sense that the users can revise their examples based on the results and even use some of the results as examples in the next iteration. A number of interviewees mentioned that by looking at the results that SQuID generated from their initial examples, they were able to come up with new examples. Moreover, when their examples contained some unintentional bias—e.g., while retrieving Disney movies, they only provided examples of recent movies—they were able to receive implicit feedback of that bias by SQuID as the results SQuID generated reflected the same bias. This feedback mechanism helped them revise their examples accordingly. In contrast, SQL does not offer such interactivity or feedback mechanism. While some interviewees used subqueries of the main query to view some intermediate results, this was just for the purpose of verifying the correctness of the main query. In contrast, SQuID's natural interaction and feedback mechanism offers additional help to the users. This makes SQuID particularly suitable for the task of data exploration. SQuID often promotes *serendipity* in the results—providing a good balance between *exploration* (serendipitous, surprising, and novel discovery) and *exploitation* (similar to the examples)—which is a desired property during data exploration.

SQuID is particularly useful for solving complex and subjective tasks. The specific properties of SQuID, specifically interactivity, providing feedback, and promoting serendipitous discovery, make it a significantly better choice for solving subjective tasks that are usually ambiguous and vague, and are very hard to solve using SQL. For example, in our studies, we used "strong actors" or "funny actors" as two examples of subjective tasks. Participants of both our controlled experiment study and interview study found thinking of examples easier than expressing their intent using SQL, especially for subjective tasks. Our results indicate that SQuID provides an easier mechanism for data retrieval and helps users overcome the difficulty of writing overly complex SQL queries for subjective tasks. In contrast, for objective tasks, we found both SQuID and SQL equally effective, given the user has basic SQL expertise.

Trust on a system depends on prior exposure, expertise, type of the tasks, and system explainability. During our controlled experiment, we wanted to measure how much the participants trust the mechanism that produces the results by asking the questions: "how well do you think SQUID did in generating the desired results?" and "how accurate were the SQL results?" While some participants reported that they were more satisfied with the results produced by SQUID than SQL, interestingly, many of them reported that they prefer SQL over SQUID even though they generally did better with SQUID (Figure 3.20(d)). This result is in line with prior work that compared a PBE tool against traditional shell-scripting and found that despite performing better using the PBE tool, users tend to trust the traditional shell-scripting more [284]. We validated this by checking against ground-truth results where SQUID groups achieved results with higher precision (more specific) and F1 score (more accurate), as shown in Figure 3.16.

Since the participants performed better when using SQUID compared to SQL, we interpret their preference for SQL to be due to three possible sources of bias: (1) *Familiarity:* The participants were at the time taking a course on relational databases and SQL, which may have artificially increased their confidence in their SQL skills. They had prior experience with SQL, but were experiencing SQUID for the first time through the study. (2) *Explainability:* SQL exposes the precise mechanism (the code) that produces the results, while we did not provide participants with an explanation of the inner workings of SQUID nor exposed the query it produces. (3) *Domain expertise:* Low domain expertise poses a hurdle in producing examples for SQUID; we posit that the users may consider SQL a more versatile mechanism for such circumstances.

We further investigated the issue of trust during our interview study by asking all our interviewees the question: "Which of these two systems, SQUID or SQL, do you trust more?" We expected SQL experts to trust SQL more, but did not observe any strong trend. Rather, the interviewees mentioned that for objective tasks, they were more confident about the SQL queries they wrote, and hence, they trusted SQL more. In contrast, for

the subjective tasks, they reported that they trusted the results produced by SQUID more, as for the subjective tasks, the most common complaint was that SQL produced too many results (less specific) and perhaps retrieved the entire database content. Ultimately, SQUID can also provide explanations, by exposing the SQL query it synthesizes in order to generate the results and the underlying mechanism used to synthesize the query. We shed more light on this in the future work.

SQUID is easy to learn. A desired property for any system is *learnability*: how easy it is to get used to the system. From our study, we found that it was very easy for the participants to learn how to use SQUID almost instantly. SQUID's interface is intuitive and both novices and experts learned how to use it, just by observing its behavior. In contrast, when participants did not know how to write certain classes of SQL queries, they simply gave up and mentioned that they cannot express their logic in SQL. This is particularly significant considering that all our study participants and interviewees had prior exposure to and experience with SQL, while this was their first experience using SQUID.

APPENDIX B

CONFORMANCE CONSTRAINTS

B.1 Proof of Lemma 5.2

Proof. Pick β_1, β_2 s.t. $\beta_1^2 + \beta_2^2 = 1$ and the following equation holds:

$$\texttt{sign}(\rho_{F_1,F_2})\beta_1\sigma(F_1(D)) + \beta_2\sigma(F_2(D)) = 0 \tag{B.1}$$

Let t be any tuple that is incongruous w.r.t. $\langle F_1, F_2\rangle$. Now, we compute how far t is from the mean of the projection F on D:

$$\begin{aligned}
|\Delta F(t)| &= |F(t) - \mu(F(D))| \\
&= |\beta_1 F_1(t) + \beta_2 F_2(t) - \mu(\beta_1 F_1(D) + \beta_2 F_2(D))| \\
&= |\beta_1 \Delta F_1(t) + \beta_2 \Delta F_2(t)| \\
&= |\beta_1 \Delta F_1(t)| + |\beta_2 \Delta F_2(t)|
\end{aligned}$$

The last step is correct only when $\beta_1\Delta F_1(t)$ and $\beta_2\Delta F_2(t)$ are of same sign. We prove this by cases:

(Case 1). $\rho_{F_1,F_2} \geq \frac{1}{2}$. In this case, β_1 and β_2 are of different signs due to Equation B.1. Moreover, since t is incongruous w.r.t. $\langle F_1, F_2\rangle$, $\Delta F_1(t)$ and $\Delta F_2(t)$ are of different signs. Hence, $\beta_1\Delta F_1(t)$ and $\beta_2\Delta F_2(t)$ are of same sign.

(Case 2). $\rho_{F_1,F_2} \leq -\frac{1}{2}$. In this case, β_1 and β_2 have the same sign due to Equation B.1. Moreover, since t is incongruous w.r.t. $\langle F_1, F_2\rangle$, $\Delta F_1(t)$ and $\Delta F_2(t)$ are of same sign. Hence, $\beta_1\Delta F_1(t)$ and $\beta_2\Delta F_2(t)$ are of same sign.

Next, we compute the variance of F on D:

$$\begin{aligned}\sigma(F(D))^2 &= \frac{1}{|D|}\sum_{t\in D}(\beta_1\Delta F_1(t)+\beta_2\Delta F_2(t))^2 \\ &= \beta_1^2\sigma(F_1(D))^2+\beta_2^2\sigma(F_2(D))^2 \\ &\qquad +2\beta_1\beta_2\rho_{F_1,F_2}\sigma(F_1(D))\sigma(F_2(D)) \\ &= \beta_1^2\sigma(F_1(D))^2+\beta_1^2\sigma(F_1(D))^2-2\beta_1^2|\rho_{F_1,F_2}|\sigma(F_1(D))^2 \\ &= 2\beta_1^2\sigma(F_1(D))^2(1-|\rho_{F_1,F_2}|)\end{aligned}$$

Hence, $\sigma(F(D)) = \sqrt{2(1-|\rho_{F_1,F_2}|)}|\beta_1|\sigma(F_1(D))$, which is also equal to $\sqrt{2(1-|\rho_{F_1,F_2}|)}|\beta_2|\sigma(F_2(D))$. Since $\sqrt{2(1-|\rho_{F_1,F_2}|)}| \leq 1$, and since $|\beta_k| < 1$, we conclude that $\sigma(F(D)) < \sigma(F_k(D))$. Now, we compute $\frac{|\Delta F(t)|}{\sigma(F(D))}$ next using the above derived facts about $|\Delta F(t)|$ and $\sigma(F(D))$.

$$\begin{aligned}\frac{|\Delta F(t)|}{\sigma(F(D))} &> \frac{|\beta_1\Delta F_1(t)|}{\sqrt{2(1-|\rho_{F_1,F_2}|)}|\beta_1|\sigma(F_1(D))} \\ &= \frac{|\Delta F_1(t)|}{\sqrt{2(1-|\rho_{F_1,F_2}|)}\sigma(F_1(D))} \geq \frac{|\Delta F_1(t)|}{\sigma(F_1(D))}\end{aligned}$$

The last step uses the fact that $|\rho_{F_1,F_2}| \geq \frac{1}{2}$. Similarly, we also get $\frac{|\Delta F(t)|}{\sigma(F(D))} > \frac{|\Delta F_2(t)|}{\sigma(F_2(D))}$. Hence, ϕ_F is stronger than both ϕ_{F_1} and ϕ_{F_2} on d, using Lemma 5.1. This completes the proof. □

B.2 Proof of Theorem 5.1

Proof. First, we use Lemma 5.2 on F_i, F_j to construct F. We initialize $I := \{i, j\}$. Next, we repeatedly do the following: We iterate over all F_k, where $k \notin I$, and check if $|\rho_{F,F_k}| \geq \frac{1}{2}$ for some k. If yes, we use Lemma 5.2 (on F and F_k) to update F to be the new projection returned by the lemma. We update $I := I \cup \{k\}$, and continue the iterations. If $|\rho_{F,F_k}| < \frac{1}{2}$ for all $k \notin I$, then we stop. The final F and index set I can easily be seen to satisfy the claims of the theorem. □

B.3 Proof of Theorem 5.2

We first provide some additional details regarding the statement of the theorem. Since standard deviation is not scale invariant, if there is no constraint on the norm of the linear projections, then it is possible to scale down the linear projections to make their standard deviations arbitrarily small. Therefore, claim (1) can not be proved for *any* linear projection, but only linear projections whose 2-norm is not too "small". Hence, we restate the theorem with some additional technical conditions.

Given a numerical dataset D, let $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ be the set of linear projections returned by Algorithm 2. Let $\sigma^* = \min_k^K \sigma(F_k(D))$. WLOG, assume $\sigma^* = \sigma(F_1(D))$ where $F_1 = \vec{A}^T\vec{w^*}$. Assume that the attribute mean is zero for all attributes in D (call this Condition 1). Then,

(1) $\sigma^* \leq \sigma(F(D))$ for every possible linear projection F whose 2-norm is sufficiently large, i.e., we require $||\vec{w}|| \geq 1$ for $F = \vec{A}^T\vec{w}$. If we do not assume Condition 1, then the requirement changes to $||\vec{w}|| \geq ||\vec{w}^{*e}|| - \mu(D^T\vec{w})||$. Here $\vec{w}^{*e}$ is the vector constructed by augmenting a dimension to $\vec{w}^*$ to turn it to an eigenvector of $D^{eT}D^e$ where $D^e = [\vec{1}; D]$.

(2) $\forall F_j, F_k \in \mathcal{F}$ s.t. $F_j \neq F_k$, $\rho_{F_j,F_k} = 0$. If we do not assume Condition 1, then the correlation coefficient is close to 0 for those F_j, F_k whose corresponding eigenvalues are much smaller than $|D|$.

Proof. The proof uses the following facts:

(Fact 1) If we add a constant c to each element of a set S of real values to get a new set S', then $\sigma(S) = \sigma(S')$.

(Fact 2) The Courant-Fischer min-max theorem [147] states that the vector $\vec{w}$ that minimizes $||M\vec{w}||/||\vec{w}||$ is the eigenvector of M^TM corresponding to the lowest eigenvalue (for any matrix M).

(Fact 3) Since $D'_N := [\vec{1}; D_N]$, by definition: $\sigma(D_N\vec{w}) = \frac{||D'_N\vec{w'}||}{\sqrt{|D|}}$, where $\vec{w}' = \begin{bmatrix} -\mu(D_N\vec{w}) \\ \vec{w} \end{bmatrix}$

(Fact 4) By the definition of variance, $\sigma(S)^2 = ||S||^2 - \mu(S)^2$.

Let $F = \vec{A}^T\vec{w}$ be an arbitrary linear projection. Since D is numerical, $D = D_N$. Let D^e denote D'_N. (We use the superscript e to denote the augmented vector/matrix).

$$
\begin{array}{lll}
\sigma(D^T\vec{w})^2 & & \\
= & \sigma(D^T\vec{w} - \vec{1}\mu)^2 & \text{(Fact 1), } \mu = \mu(D^T\vec{w}) \\
= & \sigma(D^{eT}\vec{w}^e)^2 & \text{where } \vec{w}^e = \begin{bmatrix} -\mu \\ \vec{w} \end{bmatrix} \\
= & \frac{||D^{eT}\vec{w}^e||^2}{|D|} & \text{(Fact 3)} \\
\geq & \frac{||D^{eT}\vec{w}^{*e}||^2 \cdot ||\vec{w}^e||^2}{|D| \cdot ||\vec{w}^{*e}||^2} & \text{(Fact 2)} \\
= & (\sigma(D^{eT}\vec{w}^{*e})^2 + b^2) \cdot \frac{||\vec{w}^e||^2}{||\vec{w}^{*e}||^2} & \text{(Fact 4), } b = \mu(D^{eT}\vec{w}^{*e}) \\
= & (\sigma(D^T\vec{w}^* + c)^2 + b^2) \cdot \frac{||\vec{w}^e||^2}{||\vec{w}^{*e}||^2} & \text{Expand } D^{eT}\vec{w}^{*e} \\
= & (\sigma(D^T\vec{w}^*)^2 + b^2) \cdot \frac{||\vec{w}^e||^2}{||\vec{w}^{*e}||^2} & \text{(Fact 1)} \\
= & (\sigma^{*2} + b^2) \cdot \frac{||\vec{w}^e||^2}{||\vec{w}^{*e}||^2} & \text{definition of } \sigma^* \\
\geq & \sigma^{*2} & \text{by assumption } \frac{||\vec{w}^e||^2}{||\vec{w}^{*e}||^2} \geq 1
\end{array}
$$

For the last step, we use the technical condition that the norm of the extension of $\vec{w}$ (extended by augmenting the mean over $D\vec{w}$) is at least as large as the norm of extension of $\vec{w}^*$ (extended to make it an eigenvector of $D^{eT}D^e$). When Condition 1 holds, $||\vec{w}^e||^2 = ||\vec{w}||^2$ (because $\mu(F(D))$ will be 0 and therefore, $\vec{w}^e = \begin{bmatrix} 0 \\ \vec{w} \end{bmatrix}$), and $||\vec{w}^{*e}||^2 = 1$ (for the same reason), and hence $\frac{||\vec{w}^e||^2}{||\vec{w}^{*e}||^2} \geq 1$.

For part (2) of the claim, let $F_i = \vec{A}^T\vec{w}_i$ for all i, where $\vec{w}_i$ are the coefficients of the linear projection F_i. Let $c_i = \mu(F_i(D))$.

(Fact 5) If Condition 1 holds, $\forall i\ c_i = 0$.

By construction of F_i's, we know that w_i can be extended to be an eigenvector $\begin{bmatrix} d_i \\ \vec{w_i} \end{bmatrix}$ of $D^{eT}D^e$ (with corresponding eigenvalue λ_i). In general,

(Fact 6) It is easy to work out that $d_i = \frac{-c_i}{1-\frac{\lambda_i}{|D|}}$.

Thus, we have:

$$
\begin{array}{lll}
\rho_{F_j,F_k} & & \\
= & \frac{\sum_{t\in D}\Delta F_j(t)\Delta F_k(t)}{|D|\sigma(F_j(D))\sigma(F_k(D))} & \text{(definition of } \rho) \\
= & \frac{(D\vec{w_j}-c_j\vec{1})^T(D\vec{w_k}-c_k\vec{1})}{|D|\sigma(F_j(D))\sigma(F_k(D))} & \\
= & \frac{(D^e\vec{w}_j^e)^TD^e\vec{w}_k^e}{|D|\sigma(F_j(D))\sigma(F_k(D))} & w_i^e = \begin{bmatrix} -c_i \\ \vec{w_i} \end{bmatrix} \\
= & \frac{\vec{w}_j^{eT}D^{eT}D^e\vec{w}_k^e}{|D|\sigma(F_j(D))\sigma(F_k(D))} & \\
= & \frac{\vec{w}_j^{eT}\lambda_k\vec{w}_k^e}{|D|\sigma(F_j(D))\sigma(F_k(D))} & \text{(Fact 5,6), } D^{eT}D^e\vec{w}_k^e = \lambda_k\vec{w}_k^e \\
= & 0 & \text{(eigenvectors are orthogonal)}
\end{array}
$$

When Condition 1 does not hold, Fact 5 would not hold, but Fact 6 continues to hold, and hence by continuity, if $|\frac{\lambda_i}{|D|}|$ is close to 0, then d_i will be close to $-c_i$, and ρ_{F_j,F_k} will be close to 0. □

APPENDIX C

DEBUGGING DATA SYSTEMS

C.1 Proof of Theorem 6.1

Proof. After the first intervention, we get at least $\left(\log\binom{N}{D} - \log\binom{N-S_1}{D} + 1\right)$ bits of information. Suppose that there are m interventions. Since after retrieving all information, the remaining information should be ≤ 0:

$$\log\binom{N}{D} - \sum_{i=1}^{m}\left(\log\binom{N-(i-1)S_1}{D} - \log\binom{N-iS_1}{D} + 1\right) \leq 0$$

$$\implies \log\binom{N-mS_1}{D} - m \leq 0$$

$$\implies m \geq \log\frac{(N-mS_1)!}{D!(N-mS_1-D)!}$$

$$\implies m \geq \log\frac{(N-mS_1)^D}{D!} \quad [\frac{(N-mS_1)!}{(N-mS_1-D)!} \approx (N-mS_1)^D]$$

$$\implies m \geq D\log(N-mS_1) - \log(D!)$$

$$\implies m \geq D\log N(1-\frac{mS_1}{N}) - \log(D!)$$

$$\implies m \geq D\log N + D\log(1-\frac{mS_1}{N}) - \log(D!)$$

Since $\log(1-x) \approx -x$ for small x; we assume $\frac{mS_1}{N}$ to be small:

$$\implies m \geq D\log N - \frac{mDS_1}{N} - \log(D!)$$

$$\implies m\left(1+\frac{DS_1}{N}\right) \geq D\log N - \log(D!)$$

$$\implies m\left(1+\frac{DS_1}{N}\right) \geq \log\frac{N^D}{D!}$$

$$\implies m\left(1+\frac{DS_1}{N}\right) \geq \log\frac{N!}{D!(N-D)!} \quad [N^D \approx \frac{N!}{(N-D)!}]$$

$$\implies m \geq \frac{\log\binom{N}{D}}{1+\frac{DS_1}{N}}$$

□

C.2 Proof of Theorem 6.2

Proof. Since at least S_2 predicates are discarded during each causal predicate discovery, and there are D causal predicates, we compute the upper bound of the number of required interventions:

$$
\begin{aligned}
&\sum_{i=1}^{D} \log\Big(N - (i-1)S_2\Big) \\
=&\sum_{i=1}^{D} \log\Big(N\big(1 - \frac{(i-1)S_2}{N}\big)\Big) \\
=&\sum_{i=1}^{D} \log N + \sum_{i=1}^{D} \log\Big(1 - \frac{(i-1)S_2}{N}\Big) \\
\approx&\sum_{i=1}^{D} \log N - \sum_{i=1}^{D} \frac{(i-1)S_2}{N} \quad [\log(1-x) \approx -x \text{ for small } x] \\
=&D \log N - \frac{D(D-1)S_2}{2N}
\end{aligned}
$$

□

BIBLIOGRAPHY

[1] Abedjan, Ziawasch, Golab, Lukasz, and Naumann, Felix. Profiling relational data: a survey. *VLDB J. 24*, 4 (2015), 557–581.

[2] Abouzied, Azza, Angluin, Dana, Papadimitriou, Christos H., Hellerstein, Joseph M., and Silberschatz, Avi. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013* (2013), ACM, pp. 49–60.

[3] Achlioptas, Panos, Diamanti, Olga, Mitliagkas, Ioannis, and Guibas, Leonidas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392* (2017).

[4] Adaboost classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`.

[5] Agarwal, Abhishek, Jaggi, Sidharth, and Mazumdar, Arya. Novel impossibility results for group-testing. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018* (2018), pp. 2579–2583.

[6] Agarwal, Swati, Sureka, Ashish, Mittal, Nitish, Katyal, Rohan, and Correa, Denzil. DBLP records and entries for key computer science conferences, 2016.

[7] Aggarwal, Charu C. A framework for change diagnosis of data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003* (2003), ACM, pp. 575–586.

[8] Agrawal, Sanjay, Chaudhuri, Surajit, and Das, Gautam. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002* (2002), IEEE Computer Society, pp. 5–16.

[9] Airlines dataset. `http://kt.ijs.si/elena_ikonomovska/data.html`, 2009.

[10] Akbik, Alan, Blythe, Duncan, and Vollgraf, Roland. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics* (2018), pp. 1638–1649.

[11] Aker, Ahmet, Cohn, Trevor, and Gaizauskas, Robert J. Multi-document summarization using a* search and discriminative learning. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL* (2010), ACL, pp. 482–491.

[12] Alvaro, Peter, Rosen, Joshua, and Hellerstein, Joseph M. Lineage-driven fault injection. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), pp. 331–346.

[13] Alzate, Carlos, and Suykens, Johan AK. Kernel component analysis using an epsilon-insensitive robust loss function. *IEEE Transactions on Neural Networks 19*, 9 (2008), 1583–1598.

[14] Arenas, Marcelo, Diaz, Gonzalo I., and Kostylev, Egor V. Reverse engineering SPARQL queries. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016* (2016), ACM, pp. 239–249.

[15] Arieli, Ofer, Bruynooghe, Maurice, Denecker, Marc, and Nuffelen, Bert Van. Coherent integration of databases by abductive logic programming. *CoRR abs/1107.0030* (2011).

[16] Attariyan, Mona, Chow, Michael, and Flinn, Jason. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012* (2012), pp. 307–320.

[17] Attariyan, Mona, and Flinn, Jason. Automating configuration troubleshooting with dynamic information flow analysis. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings* (2010), pp. 237–250.

[18] Attariyan, Mona, and Flinn, Jason. Automating configuration troubleshooting with confaid. *login Usenix Mag. 36*, 1 (2011).

[19] Baah, George K., Podgurski, Andy, and Harrold, Mary Jean. Causal inference for statistical fault localization. In *Proceedings of the 19th International Symposium on Software Testing and Analysis* (New York, NY, USA, 2010), ISSTA '10, ACM, pp. 73–84.

[20] Baah, George K., Podgurski, Andy, and Harrold, Mary Jean. Mitigating the confounding effects of program dependences for effective fault localization. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011* (2011), pp. 146–156.

[21] Bai, Yechao, Wang, Qingsi, Lo, Chun, Liu, Mingyan, Lynch, Jerome P., and Zhang, Xinggan. Adaptive bayesian group testing: Algorithms and performance. *Signal Processing 156* (2019), 191–207.

[22] Baik, Christopher, Jin, Zhongjun, Cafarella, Michael J., and Jagadish, H. V. Duoquest: A dual-specification system for expressive SQL queries. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), pp. 2319–2329.

[23] Bailis, Peter, Fekete, Alan, Franklin, Michael J, Ghodsi, Ali, Hellerstein, Joseph M, and Stoica, Ion. Feral concurrency control: An empirical investigation of modern application integrity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), ACM, pp. 1327–1342.

[24] Bailis, Peter, Gan, Edward, Madden, Samuel, Narayanan, Deepak, Rong, Kexin, and Suri, Sahaana. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data* (New York, NY, USA, 2017), SIGMOD '17, ACM, pp. 541–556.

[25] Baldassini, Leonardo, Johnson, Oliver, and Aldridge, Matthew. The capacity of adaptive group testing. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013* (2013), pp. 2676–2680.

[26] Ball, Thomas, and Larus, James R. Optimally profiling and tracing programs. *ACM Trans. Program. Lang. Syst. 16*, 4 (1994), 1319–1360.

[27] Barceló, Pablo, and Romero, Miguel. The complexity of reverse engineering problems for conjunctive queries. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy* (2017), vol. 68 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 7:1–7:17.

[28] Barddal, Jean Paul, Gomes, Heitor Murilo, Enembreck, Fabrício, and Pfahringer, Bernhard. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software 127* (2017), 278–294.

[29] Barowy, Daniel W, Berger, Emery D, and Zorn, Benjamin. Excelint: Automatically finding spreadsheet formula errors. *Proceedings of the ACM on Programming Languages 2*, OOPSLA (2018), 1–26.

[30] Barowy, Daniel W., Gochev, Dimitar, and Berger, Emery D. CheckCell: data debugging for spreadsheets. In *OOPSLA* (2014), pp. 507–523.

[31] Barowy, Daniel W., Gulwani, Sumit, Hart, Ted, and Zorn, Benjamin G. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015* (2015), pp. 218–228.

[32] Beautiful soup. https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[33] Bekker, Jessa, and Davis, Jesse. Positive and unlabeled relational classification through label frequency estimation. In *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers* (2017), vol. 10759 of *Lecture Notes in Computer Science*, Springer, pp. 16–30.

[34] Bekker, Jessa, and Davis, Jesse. Estimating the class prior in positive and unlabeled data through decision tree induction. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (2018), AAAI Press, pp. 2712–2719.

[35] Bekker, Jessa, and Davis, Jesse. Learning from positive and unlabeled data: A survey. *CoRR abs/1811.04820* (2018).

[36] Bellamy, Rachel K. E., Dey, Kuntal, Hind, Michael, Hoffman, Samuel C., Houde, Stephanie, Kannan, Kalapriya, Lohia, Pranay, Martino, Jacquelyn, Mehta, Sameep, Mojsilovic, Aleksandra, Nagar, Seema, Ramamurthy, Karthikeyan Natesan, Richards, John T., Saha, Diptikalyan, Sattigeri, Prasanna, Singh, Moninder, Varshney, Kush R., and Zhang, Yunfeng. AI fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM J. Res. Dev. 63*, 4/5 (2019), 4:1–4:15.

[37] Bertossi, Leopoldo E., and Salimi, Babak. Causes for query answers from databases: Datalog abduction, view-updates, and integrity constraints. *Int. J. Approx. Reason. 90* (2017), 226–252.

[38] Bias in amazon hiring. https://becominghuman.ai/amazons-sexist-ai-recruiting-tool-how-did-it-go-so-wrong-e3d14816d98e.

[39] Bifet, Albert, and Gavaldà, Ricard. Learning from time-changing data with adaptive windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA* (2007), SIAM, pp. 443–448.

[40] Bifet, Albert, Holmes, Geoff, Kirkby, Richard, and Pfahringer, Bernhard. MOA: massive online analysis. *J. Mach. Learn. Res. 11* (2010), 1601–1604.

[41] Blei, David M., Ng, Andrew Y., and Jordan, Michael I. Latent dirichlet allocation. *J. Mach. Learn. Res. 3* (2003), 993–1022.

[42] Bleifuß, Tobias, Kruse, Sebastian, and Naumann, Felix. Efficient denial constraint discovery with hydra. *Proc. VLDB Endow. 11*, 3 (2017), 311–323.

[43] Bohannon, Philip, Fan, Wenfei, Geerts, Floris, Jia, Xibei, and Kementsietsidis, Anastasios. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007* (2007), IEEE Computer Society, pp. 746–755.

[44] Bonifati, Angela, Ciucanu, Radu, and Staworko, Slawek. Learning join queries from user examples. *ACM Trans. Database Syst. 40*, 4 (2016), 24:1–24:38.

[45] Bonifati, Angela, Comignani, Ugo, Coquery, Emmanuel, and Thion, Romuald. Interactive mapping specification with exemplar tuples. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017* (2017), ACM, pp. 667–682.

[46] Bovenzi, Antonio, Cotroneo, Domenico, Pietrantuono, Roberto, and Russo, Stefano. On the aging effects due to concurrency bugs: A case study on MySQL. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering* (2012), IEEE, pp. 211–220.

[47] Brachmann, Mike, Bautista, Carlos, Castelo, Sonia, Feng, Su, Freire, Juliana, Glavic, Boris, Kennedy, Oliver, Müeller, Heiko, Rampin, Rémi, Spoth, William, et al. Data debugging and exploration with vizier. In *Proceedings of the 2019 International Conference on Management of Data* (2019), pp. 1877–1880.

[48] Breck, Eric, Polyzotis, Neoklis, Roy, Sudip, Whang, Steven Euijong, and Zinkevich, Martin. Data validation for machine learning. In *Conference on Systems and Machine Learning (SysML). https://www. sysml. cc/doc/2019/167. pdf* (2019).

[49] Brucato, Matteo, Beltran, Juan Felipe, Abouzied, Azza, and Meliou, Alexandra. Scalable package queries in relational database systems. *Proc. VLDB Endow. 9*, 7 (2016), 576–587.

[50] Bu, Li, Alippi, Cesare, and Zhao, Dongbin. A pdf-free change detection test based on density difference estimation. *IEEE Trans. Neural Netw. Learning Syst. 29*, 2 (2018), 324–334.

[51] Cadamuro, Gabriel, Gilad-Bachrach, Ran, and Zhu, Xiaojin. Debugging machine learning models. In *ICML Workshop on Reliable Machine Learning in the Wild* (2016).

[52] Cardiovascular disease: `https://www.kaggle.com/sulianova/cardiovascular-disease-dataset`, Feb 2020.

[53] Caruccio, Loredana, Deufemia, Vincenzo, and Polese, Giuseppe. On the discovery of relaxed functional dependencies. In *Proceedings of the 20th International Database Engineering & Applications Symposium* (2016), pp. 53–61.

[54] Casalicchio, Giuseppe, Molnar, Christoph, and Bischl, Bernd. Visualizing the feature importance for black box models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2018), Springer, pp. 655–670.

[55] Chapelle, Olivier, Schlkopf, Bernhard, and Zien, Alexander. *Semi-Supervised Learning*, 1st ed. The MIT Press, 2010.

[56] Chen, Mike Y., Kiciman, Emre, Fratkin, Eugene, Fox, Armando, and Brewer, Eric. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of IEEE DSN* (USA, 2002), DSN'02, IEEE, pp. 595–604.

[57] Chen, Wei-Fan, Syed, Shahbaz, Stein, Benno, Hagen, Matthias, and Potthast, Martin. Abstractive snippet generation. In *Proceedings of The Web Conference 2020* (2020), pp. 1309–1319.

[58] Cheng, Hong, Lo, David, Zhou, Yang, Wang, Xiaoyin, and Yan, Xifeng. Identifying bug signatures using discriminative graph mining. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009* (2009), pp. 141–152.

[59] Chilimbi, Trishul M., Liblit, Ben, Mehra, Krishna K., Nori, Aditya V., and Vaswani, Kapil. HOLMES: effective statistical debugging via efficient path profiling. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings* (2009), pp. 34–44.

[60] Chirigati, Fernando, Doraiswamy, Harish, Damoulas, Theodoros, and Freire, Juliana. Data polygamy: The many-many relationships among urban spatio-temporal data sets. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016* (2016), pp. 1011–1025.

[61] Chu, Xu, Ilyas, Ihab F., and Papotti, Paolo. Discovering denial constraints. *Proc. VLDB Endow. 6*, 13 (2013), 1498–1509.

[62] Chu, Xu, Ilyas, Ihab F., and Papotti, Paolo. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013* (2013), IEEE Computer Society, pp. 458–469.

[63] Azure Cosmos DB. `https://docs.microsoft.com/en-us/azure/cosmos-db/`.

[64] Cosmos DB bug. `https://github.com/Azure/azure-cosmos-dotnet-v3/pull/713`.

[65] Cypher, Allen. Eager: Programming repetitive tasks by example. In *Readings in human–computer interaction*. Elsevier, 1995, pp. 804–810.

[66] Dasu, Tamraparni, Krishnan, Shankar, Venkatasubramanian, Suresh, and Yi, Ke. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Symp. on the Interface of Statistics, Computing Science, and Applications* (2006).

[67] Daumé III, Hal. Bayesian query-focused summarization. *arXiv preprint arXiv:0907.1814* (2009).

[68] de Mello, Rodrigo Fernandes, Vaz, Yule, Ferreira, Carlos Henrique Grossi, and Bifet, Albert. On learning guarantees to unsupervised concept drift detection on data streams. *Expert Syst. Appl. 117* (2019), 90–102.

[69] de Souza, Vinícius M. A., Silva, Diego Furtado, Gama, João, and Batista, Gustavo E. A. P. A. Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015* (2015), SIAM, pp. 873–881.

[70] Dean, Jeffrey, and Ghemawat, Sanjay. Mapreduce: simplified data processing on large clusters. *Commun. ACM 51*, 1 (2008), 107–113.

[71] Denouden, Taylor, Salay, Rick, Czarnecki, Krzysztof, Abdelzad, Vahdat, Phan, Buu, and Vernekar, Sachin. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *CoRR abs/1812.02765* (2018).

[72] Deutch, Daniel, and Gilad, Amir. Qplain: Query by explanation. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016* (2016), IEEE Computer Society, pp. 1358–1361.

[73] Diaz, Gonzalo I., Arenas, Marcelo, and Benedikt, Michael. Sparqlbye: Querying RDF data by example. *Proc. VLDB Endow. 9*, 13 (2016), 1533–1536.

[74] Dimitriadou, Kyriaki, Papaemmanouil, Olga, and Diao, Yanlei. AIDE: an active learning-based approach for interactive data exploration. *IEEE Trans. Knowl. Data Eng. 28*, 11 (2016), 2842–2856.

[75] Dong, Xin Luna, Gabrilovich, Evgeniy, Heitz, Geremy, Horn, Wilko, Murphy, Kevin, Sun, Shaohua, and Zhang, Wei. From data fusion to knowledge fusion. *Proc. VLDB Endow. 7*, 10 (2014), 881–892.

[76] dos Reis, Denis Moreira, Flach, Peter A., Matwin, Stan, and Batista, Gustavo E. A. P. A. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016* (2016), ACM, pp. 1545–1554.

[77] Drosos, Ian, Barik, Titus, Guo, Philip J., DeLine, Robert, and Gulwani, Sumit. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020* (2020), pp. 1–12.

[78] Drosou, Marina, and Pitoura, Evaggelia. Ymaldb: exploring relational databases via result-driven recommendations. *VLDB J. 22*, 6 (2013), 849–874.

[79] Du, Ding-Zhu, and Hwang, Frank K. *Combinatorial group testing and its applications*. World Scientific, Singapore River Edge, N.J, 1993.

[80] Du, Dingzhu, Hwang, Frank K, and Hwang, Frank. *Combinatorial group testing and its applications*, vol. 12. World Scientific, 2000.

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

[82] DUC dataset. `https://duc.nist.gov/data.html`.

[83] Eirinaki, Magdalini, Abraham, Suju, Polyzotis, Neoklis, and Shaikh, Naushin. Querie: Collaborative database exploration. *IEEE Trans. Knowl. Data Eng. 26*, 7 (2014), 1778–1790.

[84] Elkan, Charles, and Noto, Keith. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008* (2008), ACM, pp. 213–220.

[85] Fabbri, Alexander R, Li, Irene, She, Tianwei, Li, Suyi, and Radev, Dragomir R. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749* (2019).

[86] Faithfull, William J., Diez, Juan José Rodríguez, and Kuncheva, Ludmila I. Combining univariate approaches for ensemble change detection in multivariate data. *Information Fusion 45* (2019), 202–214.

[87] Fan, Ju, Li, Guoliang, and Zhou, Lizhu. Interactive SQL query suggestion: Making databases user-friendly. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany* (2011), IEEE Computer Society, pp. 351–362.

[88] Fan, Wenfei, Geerts, Floris, and Jia, Xibei. A revival of integrity constraints for data cleaning. *Proc. VLDB Endow. 1*, 2 (2008), 1522–1523.

[89] Fan, Wenfei, Geerts, Floris, Jia, Xibei, and Kementsietsidis, Anastasios. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst. 33*, 2 (2008), 6:1–6:48.

[90] Fan, Wenfei, Geerts, Floris, Lakshmanan, Laks V. S., and Xiong, Ming. Discovering conditional functional dependencies. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China* (2009), pp. 1231–1234.

[91] Fariha, Anna, Brucato, Matteo, Haas, Peter J., and Meliou, Alexandra. SuDocu: Summarizing Documents by Example. *Proc. VLDB Endow. 13*, 12 (2020), 2861–2864.

[92] Fariha, Anna, Brucato, Matteo, Haas, Peter J., and Meliou, Alexandra. SuDocu: `http://sudocu.cs.umass.edu/demo/`, 2020.

[93] Fariha, Anna, Cousins, Lucy, Mahyar, Narges, and Meliou, Alexandra. Example-driven user intent discovery: Empowering users to cross the SQL barrier through query by example. *CoRR abs/2012.14800* (2020).

[94] Fariha, Anna, and Meliou, Alexandra. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *Proc. VLDB Endow. 12*, 11 (2019), 1262–1275.

[95] Fariha, Anna, and Meliou, Alexandra. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *CoRR abs/1906.10322* (2019).

[96] Fariha, Anna, Nath, Suman, and Meliou, Alexandra. Causality-guided adaptive interventional debugging. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), ACM, pp. 431–446.

[97] Fariha, Anna, Nath, Suman, and Meliou, Alexandra. Causality-guided adaptive interventional debugging. *CoRR abs/2003.09539* (2020).

[98] Fariha, Anna, Tiwari, Ashish, Meliou, Alexandra, Radhakrishna, Arjun, and Gulwani, Sumit. CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD Conference 2021, online conference [Xi'an, Shaanxi, China], June 20-25, 2021* (2021).

[99] Fariha, Anna, Tiwari, Ashish, Radhakrishna, Arjun, and Gulwani, Sumit. ExTuNe: Explaining Tuple Non-conformance. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), ACM, pp. 2741–2744.

[100] Fariha, Anna, Tiwari, Ashish, Radhakrishna, Arjun, Gulwani, Sumit, and Meliou, Alexandra. Conformance constraint discovery: Measuring trust in data-driven systems. *CoRR abs/2003.01289* (2020).

[101] Fariha, Anna, Tiwari, Ashish, Radhakrishna, Arjun, Gulwani, Sumit, and Meliou, Alexandra. Conformance Constraint Discovery: Measuring Trust in Data-Driven Systems. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD Conference 2021, online conference [Xi'an, Shaanxi, China], June 20-25, 2021* (2021).

[102] Fellows, Michael R, Fomin, Fedor V, Lokshtanov, Daniel, Rosamond, Frances, Saurabh, Saket, and Villanger, Yngve. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences 78*, 3 (2012), 707–719.

[103] Fernandes, Cristina G, Schmidt, Tina Janne, and Taraz, Anusch. On minimum bisection and related cut problems in trees and tree-like graphs. *Journal of Graph Theory 89*, 2 (2018), 214–245.

[104] Feser, John K, Chaudhuri, Swarat, and Dillig, Isil. Synthesizing data structure transformations from input-output examples. *ACM SIGPLAN Notices 50*, 6 (2015), 229–239.

[105] Feyzi, Farid, and Parsa, Saeed. Inforence: Effective fault localization based on information-theoretic analysis and statistical causal inference. *CoRR abs/1712.03361* (2017).

[106] Flashfill. `https://support.microsoft.com/en-us/office/using-flash-fill-in-excel-3f9bcf1e-db93-4890-94a0-1578341f73f7`.

[107] Fraser, Gordon, and Arcuri, Andrea. Whole test suite generation. *IEEE Transactions on Software Engineering 39*, 2 (2013), 276–291.

[108] Frermann, Lea, and Klementiev, Alexandre. Inducing document structure for aspect-based summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), pp. 6263–6273.

[109] Frid, Emma, Gomes, Celso, and Jin, Zeyu. Music creation by example. In *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020* (2020), pp. 1–13.

[110] Funny actors. `https://www.imdb.com/list/ls000025701`.

[111] Gaber, Mohamed Medhat, and Yu, Philip S. Classification of changes in evolving data streams using online clustering result deviation. In *Proc. Of International Workshop on Knowledge Discovery in Data Streams* (2006).

[112] Galhotra, Sainyam, Fariha, Anna, Lourenço, Raoni, Freire, Juliana, Meliou, Alexandra, and Srivastava, Divesh. DataExposer: Exposing Disconnect between Data and Systems. *CoRR abs/2105.06058* (2021).

[113] Galhotra, Sainyam, Khurana, Udayan, Hassanzadeh, Oktie, Srinivas, Kavitha, Samulowitz, Horst, and Qi, Miao. Automated feature enhancement for predictive modeling using external knowledge. In *2019 International Conference on Data Mining Workshops (ICDMW)* (2019), IEEE, pp. 1094–1097.

[114] Gama, João, Medas, Pedro, Castillo, Gladys, and Rodrigues, Pedro Pereira. Learning with drift detection. In *Advances in Artificial Intelligence - SBIA* (2004), pp. 286–295.

[115] Gama, João, Zliobaite, Indre, Bifet, Albert, Pechenizkiy, Mykola, and Bouchachia, Abdelhamid. A survey on concept drift adaptation. *ACM Comput. Surv. 46*, 4 (2014), 44:1–44:37.

[116] Garey, Michael R, Johnson, David S, and Stockmeyer, Larry. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing* (1974), pp. 47–63.

[117] Ge, Xiaoyu, Xue, Yanbing, Luo, Zhipeng, Sharaf, Mohamed A., and Chrysanthis, Panos K. REQUEST: A scalable framework for interactive construction of exploratory queries. In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016* (2016), IEEE Computer Society, pp. 646–655.

[118] Gebaly, Kareem El, Agrawal, Parag, Golab, Lukasz, Korn, Flip, and Srivastava, Divesh. Interpretable and informative explanations of outcomes. *Proc. VLDB Endow. 8*, 1 (2014), 61–72.

[119] Getoor, Lise, and Taskar, Ben. Introduction to statistical relational learning, 2007.

[120] Glerum, Kirk, Kinshumann, Kinshuman, Greenberg, Steve, Aul, Gabriel, Orgovan, Vince R., Nichols, Greg, Grant, David, Loihle, Gretchen, and Hunt, Galen C. Debugging in the (very) large: ten years of implementation and experience. In *SOSP* (2009), pp. 103–116.

[121] Godefroid, Patrice, Levin, Michael Y., and Molnar, David A. Automated whitebox fuzz testing. In *Proceedings of NDSS* (2008), p. 151–166.

[122] Google vision racism. `https://algorithmwatch.org/en/story/google-vision-racism/`.

[123] Gudmundsdottir, Helga, Salimi, Babak, Balazinska, Magdalena, Ports, Dan R. K., and Suciu, Dan. A demonstration of interactive analysis of performance measurements with viska. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017* (2017), pp. 1707–1710.

[124] Gulwani, Sumit. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011* (2011), pp. 317–330.

[125] Gulwani, Sumit. Programming by examples - and its applications in data wrangling. In *Dependable Software Systems Engineering*. IOS Press, 2016, pp. 137–158.

[126] Gulwani, Sumit. Programming by examples: Applications, algorithms, and ambiguity resolution. In *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings* (2016), pp. 9–14.

[127] Gulwani, Sumit, and Jain, Prateek. Programming by examples: Pl meets ml. In *Asian Symposium on Programming Languages and Systems* (2017), Springer, pp. 3–20.

[128] Gulzar, Muhammad Ali, Wang, Siman, and Kim, Miryung. Bigsift: Automated debugging of big data analytics in data-intensive scalable computing. In *Proceedings of ESEC/FSE* (New York, NY, USA, 2018), ESEC/FSE 2018, ACM, pp. 863–866.

[129] Guo, Chuan, Pleiss, Geoff, Sun, Yu, and Weinberger, Kilian Q. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1321–1330.

[130] Haghighi, Aria, and Vanderwende, Lucy. Exploring content models for multi-document summarization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (2009), pp. 362–370.

[131] Hailpern, Brent, and Santhanam, Padmanabhan. Software debugging, testing, and verification. *IBM Systems Journal 41*, 1 (2002), 4–12.

[132] Halpern, Joseph Y., and Pearl, Judea. Causes and explanations: A structural-model approach: Part 1: Causes. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001* (2001), pp. 194–202.

[133] Han, Jialong, Zheng, Kai, Sun, Aixin, Shang, Shuo, and Wen, Ji-Rong. Discovering neighborhood pattern queries by sample answers in knowledge base. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016* (2016), IEEE Computer Society, pp. 1014–1025.

[134] Han, Seungjae, Shin, Kang G, and Rosenberg, Harold A. Doctor: An integrated software fault injection environment for distributed real-time systems. In *Proceedings of 1995 IEEE International Computer Performance and Dependability Symposium* (1995), IEEE, pp. 204–213.

[135] He, Yeye, Ganjam, Kris, Lee, Kukjin, Wang, Yue, Narasayya, Vivek R., Chaudhuri, Surajit, Chu, Xu, and Zheng, Yudian. Transform-data-by-example (TDE): extensible data transformation in excel. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018* (2018), pp. 1785–1788.

[136] Heise, Arvid, Quiané-Ruiz, Jorge-Arnulfo, Abedjan, Ziawasch, Jentzsch, Anja, and Naumann, Felix. Scalable discovery of unique column combinations. *Proc. VLDB Endow. 7*, 4 (2013), 301–312.

[137] Hellerstein, Joseph M. Quantitative data cleaning for large databases.

[138] Hendrycks, Dan, and Gimpel, Kevin. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings* (2017).

[139] Henzinger, Thomas A, Jhala, Ranjit, Majumdar, Rupak, and Sutre, Grégoire. Software verification with blast. In *International SPIN Workshop on Model Checking of Software* (2003), Springer, pp. 235–239.

[140] Herlihy, Maurice. A methodology for implementing highly concurrent data objects (abstract). *Operating Systems Review 26*, 2 (1992), 12.

[141] Hinton, Geoffrey E, and Salakhutdinov, Ruslan R. Reducing the dimensionality of data with neural networks. *science 313*, 5786 (2006), 504–507.

[142] Hitchcock, Christopher. Conditioning, intervening, and decision. *Synthese 193* (03 2015).

[143] Ho, Shen-Shyang. A martingale framework for concept change detection in time-varying data streams. In *ICML* (2005), pp. 321–327.

[144] Holler, Christian, Herzig, Kim, and Zeller, Andreas. Fuzzing with code fragments. In *Proceedings of USENIX Security Symposium* (2012), pp. 445–458.

[145] Hong, Kai, Conroy, John M, Favre, Benoit, Kulesza, Alex, Lin, Hui, and Nenkova, Ani. A repository of state of the art and competitive baseline summaries for generic news summarization. In *LREC* (2014), Citeseer, pp. 1608–1616.

[146] Hooi, Bryan, and Faloutsos, Christos. Branch and border: Partition-based change detection in multivariate time series. In *SDM* (2019), pp. 504–512.

[147] Horn, Roger A., and Johnson, Charles R. *Matrix Analysis*, 2nd ed. Cambridge University Press, New York, NY, USA, 2012.

[148] House prices: Advanced regression techniques: Advanced regression techniques. `https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data`, Feb 2020.

[149] Hristidis, Vagelis, and Papakonstantinou, Yannis. DISCOVER: keyword search in relational databases. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002* (2002), Morgan Kaufmann, pp. 670–681.

[150] Huhtala, Yka, Kärkkäinen, Juha, Porkka, Pasi, and Toivonen, Hannu. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal 42*, 2 (1999), 100–111.

[151] Hwang, Frank K. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association 67*, 339 (1972), 605–608.

[152] Ibm aif 360. `https://aif360.mybluemix.net/`.

[153] Ienco, Dino, Bifet, Albert, Pfahringer, Bernhard, and Poncelet, Pascal. Change detection in categorical evolving data streams. In *Symposium on Applied Computing, SAC* (2014), pp. 792–797.

[154] Ilyas, Ihab F, Markl, Volker, Haas, Peter, Brown, Paul, and Aboulnaga, Ashraf. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), pp. 647–658.

[155] Ilyas, Ihab F., Markl, Volker, Haas, Peter J., Brown, Paul, and Aboulnaga, Ashraf. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004* (2004), pp. 647–658.

[156] Imdb dataset. `https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`.

[157] Inala, Jeevana Priya, and Singh, Rishabh. Webrelate: integrating web data with spreadsheets using examples. *Proceedings of the ACM on Programming Languages 2*, POPL (2017), 1–28.

[158] Iqbal, Md Shahriar, Krishna, Rahul, Javidian, Mohammad Ali, Ray, Baishakhi, and Jamshidi, Pooyan. Cadet: A systematic method for debugging misconfigurations using counterfactual reasoning.

[159] Is amazon same-day delivery service racist? the christian science monitor. `https://www.csmonitor.com/Business/2016/0423/Is-Amazon-same-day-delivery-service-racist`.

[160] Jayaram, Nandish, Gupta, Mahesh, Khan, Arijit, Li, Chengkai, Yan, Xifeng, and Elmasri, Ramez. GQBE: querying knowledge graphs by example entity tuples. 1250–1253.

[161] Jensen, David D., Burroni, Javier, and Rattigan, Matthew J. Object conditioning for causal inference. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019* (2019), p. 393.

[162] Jha, Susmit. Trust, resilience and interpretability of AI models. In *Numerical Software Verification - 12th International Workshop, NSV@CAV* (2019), pp. 3–25.

[163] Jha, Susmit, Gulwani, Sumit, Seshia, Sanjit A, and Tiwari, Ashish. Oracle-guided component-based program synthesis. In *2010 ACM/IEEE 32nd International Conference on Software Engineering* (2010), vol. 1, IEEE, pp. 215–224.

[164] Jiang, Heinrich, Kim, Been, Guan, Melody Y., and Gupta, Maya R. To trust or not to trust A classifier. In *NeurIPS* (2018), pp. 5546–5557.

[165] Jiang, Lilong, and Nandi, Arnab. Snaptoquery: Providing interactive feedback during exploratory query specification. *Proc. VLDB Endow. 8*, 11 (2015), 1250–1261.

[166] Jiang, Lingxiao, and Su, Zhendong. Context-aware statistical debugging: from bug predictors to faulty control flow paths. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)* (2007), pp. 184–193.

[167] Jin, Guoliang, Thakur, Aditya V., Liblit, Ben, and Lu, Shan. Instrumentation and sampling strategies for cooperative concurrency bug isolation. In *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA* (2010), pp. 241–255.

[168] Jin, Zhongjun. *Democratizing Self-Service Data Preparation through Example Guided Program Synthesis*. PhD thesis, 2020.

[169] Johnson, Brittany, Brun, Yuriy, and Meliou, Alexandra. Causal testing: Finding defects' root causes. In *ICSE* (2020).

[170] Johnson, Noah M., Caballero, Juan, Chen, Kevin Zhijie, McCamant, Stephen, Poosankam, Pongsin, Reynaud, Daniel, and Song, Dawn. Differential slicing: Identifying causal execution differences for security applications. In *IEEE Symposium on Security and Privacy* (2011), IEEE Computer Society, pp. 347–362.

[171] Jones, James A., and Harrold, Mary Jean. Empirical evaluation of the tarantula automatic fault-localization technique. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA* (2005), pp. 273–282.

[172] Kafka - a distributed streaming platform. `https://kafka.apache.org/`.

[173] Kafka bug. `https://github.com/confluentinc/confluent-kafka-dotnet/issues/279`.

[174] Kakas, Antonis C. Abduction. In *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017, pp. 1–8.

[175] Kalashnikov, Dmitri V., Lakshmanan, Laks V. S., and Srivastava, Divesh. Fastqre: Fast query reverse engineering. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018* (2018), ACM, pp. 337–350.

[176] Kanawati, Ghani A., Kanawati, Nasser A., and Abraham, Jacob A. FERRARI: A flexible software-based fault and error injection system. *IEEE Trans. Computers 44*, 2 (1995), 248–260.

[177] Karaletsos, Theofanis, Belongie, Serge, and Rätsch, Gunnar. Bayesian representation learning with oracle constraints. *arXiv preprint arXiv:1506.05011* (2015).

[178] Karbasi, Amin, and Zadimoghaddam, Morteza. Sequential group testing with graph constraints. In *2012 IEEE Information Theory Workshop, Lausanne, Switzerland, September 3-7, 2012* (2012), pp. 292–296.

[179] Kasikci, Baris, Cui, Weidong, Ge, Xinyang, and Niu, Ben. Lazy diagnosis of in-production concurrency bugs. In *SOSP* (2017), ACM, pp. 582–598.

[180] Kawahara, Yoshinobu, and Sugiyama, Masashi. Change-point detection in time-series data by direct density-ratio estimation. In *SDM* (2009), pp. 389–400.

[181] Keerthi, S Sathiya, and Lin, Chih-Jen. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation 15*, 7 (2003), 1667–1689.

[182] Khan, Shehroz S, and Madden, Michael G. A survey of recent trends in one class classification. In *AICS* (2009), Springer, pp. 188–197.

[183] Khot, Tushar, Natarajan, Sriraam, and Shavlik, Jude W. Relational one-class classification: A non-parametric approach. In *AAAI* (2014), pp. 2453–2459.

[184] Khoussainova, Nodira, Kwon, YongChul, Balazinska, Magdalena, and Suciu, Dan. SnipSuggest: Context-aware autocompletion for SQL. *Proc. VLDB Endow. 4*, 1 (2010), 22–33.

[185] Kifer, Daniel, Ben-David, Shai, and Gehrke, Johannes. Detecting change in data streams. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004* (2004), Morgan Kaufmann, pp. 180–191.

[186] Kini, Dileep, and Gulwani, Sumit. Flashnormalize: Programming by examples for text normalization. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (2015), pp. 776–783.

[187] Koudas, Nick, Saha, Avishek, Srivastava, Divesh, and Venkatasubramanian, Suresh. Metric functional dependencies. In *2009 IEEE 25th International Conference on Data Engineering* (2009), IEEE, pp. 1275–1278.

[188] Kriegel, Hans-Peter, Kröger, Peer, Schubert, Erich, and Zimek, Arthur. Outlier detection in arbitrarily oriented subspaces. In *2012 IEEE 12th international conference on data mining* (2012), IEEE, pp. 379–388.

[189] Krishna, Rahul, Iqbal, Md Shahriar, Javidian, Mohammad Ali, Ray, Baishakhi, and Jamshidi, Pooyan. CADET: A systematic method for debugging misconfigurations using counterfactual reasoning. *CoRR abs/2010.06061* (2020).

[190] Kruse, Sebastian, and Naumann, Felix. Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment 11*, 7 (2018), 759–772.

[191] Küçük, Yigit, Henderson, Tim A. D., and Podgurski, Andy. Improving fault localization by integrating value and predicate based causal inference techniques. *CoRR abs/2102.06292* (2021).

[192] Kulesza, Todd, Burnett, Margaret, Wong, Weng-Keen, and Stumpf, Simone. Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th international conference on intelligent user interfaces* (2015), pp. 126–137.

[193] Kumar, Amresh, Kiran, M, and Prathap, BR. Verification and validation of mapreduce program model for parallel k-means algorithm on hadoop cluster. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)* (2013), IEEE, pp. 1–8.

[194] Kuncheva, Ludmila I., and Faithfull, William J. PCA feature extraction for change detection in multidimensional unlabeled data. *IEEE Trans. Neural Netw. Learning Syst. 25*, 1 (2014), 69–80.

[195] Lam, Wing, Godefroid, Patrice, Nath, Suman, Santhiar, Anirudh, and Thummalapenta, Suresh. Root causing flaky tests in a large-scale industrial setting. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (New York, NY, USA, 2019), ISSTA 2019, ACM, pp. 101–111.

[196] Lamport, Leslie. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (1978), 558–565.

[197] Langer, Philipp, and Naumann, Felix. Efficient order dependency detection. *VLDB J. 25*, 2 (2016), 223–241.

[198] Le, Vu, and Gulwani, Sumit. Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014* (2014), pp. 542–553.

[199] Lee, Doris Jung Lin, and Parameswaran, Aditya G. The case for a visual discovery assistant: A holistic solution for accelerating visual data exploration. *IEEE Data Eng. Bull. 41*, 3 (2018), 3–14.

[200] Lee, Tak Yeon, Dugan, Casey, and Bederson, Benjamin B. Towards understanding human mistakes of programming by example: an online user study. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces* (2017), pp. 257–261.

[201] Leesatapornwongsa, Tanakorn, Lukman, Jeffrey F, Lu, Shan, and Gunawi, Haryadi S. Taxdc: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. *ACM SIGPLAN Notices 51*, 4 (2016), 517–530.

[202] Lev, Guy, Shmueli-Scheuer, Michal, Herzig, Jonathan, Jerbi, Achiya, and Konopnicki, David. Talksumm: A dataset and scalable annotation method for scientific paper summarization based on conference talks. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers* (2019), Association for Computational Linguistics, pp. 2125–2131.

[203] Li, Hao, Chan, Chee-Yong, and Maier, David. Query from examples: An iterative, data-driven approach to query construction. *Proc. VLDB Endow. 8*, 13 (2015), 2158–2169.

[204] Li, Tongxin, Chan, Chun Lam, Huang, Wenhao, Kaced, Tarik, and Jaggi, Sidharth. Group testing with prior statistics. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014* (2014), pp. 2346–2350.

[205] Liblit, Ben, Naik, Mayur, Zheng, Alice X., Aiken, Alexander, and Jordan, Michael I. Scalable statistical bug isolation. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005* (2005), pp. 15–26.

[206] Lieberman, Henry. Programming by example (introduction). *Communications of the ACM 43*, 3 (2000), 72–74.

[207] Lim, Lipyeow, Wang, Haixun, and Wang, Min. Semantic queries by example. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013* (2013), ACM, pp. 347–358.

[208] Lin, Chin-Yew. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out* (2004), pp. 74–81.

[209] Lin, Hui, and Bilmes, Jeff A. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA* (2010), The Association for Computational Linguistics, pp. 912–920.

[210] Liu, Bing, Dai, Yang, Li, Xiaoli, Lee, Wee Sun, and Yu, Philip S. Building text classifiers using positive and unlabeled examples. In *ICDM* (2003), pp. 179–188.

[211] Liu, Bo, Qi, Zhengwei, Wang, Bin, and Ma, Ruhui. Pinso: Precise isolation of concurrency bugs via delta triaging. In *ICSME* (2014), IEEE Computer Society, pp. 201–210.

[212] Liu, Chao, Fei, Long, Yan, Xifeng, Han, Jiawei, and Midkiff, Samuel P. Statistical debugging: A hypothesis testing-based approach. *IEEE Trans. Software Eng. 32*, 10 (2006), 831–848.

[213] Liu, Haopeng, Lu, Shan, Musuvathi, Madan, and Nath, Suman. What bugs cause production cloud incidents? In *Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS 2019, Bertinoro, Italy, May 13-15, 2019* (2019), pp. 155–162.

[214] Liu, Shanhong. Most used languages among software developers globally 2020, Jun 2020.

[215] Livshits, Ester, Heidari, Alireza, Ilyas, Ihab F., and Kimelfeld, Benny. Approximate denial constraints. *CoRR abs/2005.08540* (2020).

[216] Lourenço, Raoni, Freire, Juliana, and Shasha, Dennis E. Bugdoc: Algorithms to debug computational processes. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), ACM, pp. 463–478.

[217] Lu, Haochuan, Xu, Huanlin, Liu, Nana, Zhou, Yangfan, and Wang, Xin. Data sanity check for deep learning systems via learnt assertions. *CoRR abs/1909.03835* (2019).

[218] Lu, Shan, Park, Soyeon, Hu, Chongfeng, Ma, Xiao, Jiang, Weihang, Li, Zhenmin, Popa, Raluca A., and Zhou, Yuanyuan. MUVI: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007* (2007), pp. 103–116.

[219] Lu, Shan, Park, Soyeon, Seo, Eunsoo, and Zhou, Yuanyuan. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2008, Seattle, WA, USA, March 1-5, 2008* (2008), pp. 329–339.

[220] Luo, Qingzhou, Hariri, Farah, Eloussi, Lamyaa, and Marinov, Darko. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2014), ACM, pp. 643–653.

[221] Manevitz, Larry M, and Yousef, Malik. One-class svms for document classification. *JMLR 2*, Dec (2001), 139–154.

[222] Manning, Christopher D., Raghavan, Prabhakar, and Schütze, Hinrich. *Introduction to information retrieval*. Cambridge University Press, 2008.

[223] Marchi, Fabien De, Lopes, Stéphane, and Petit, Jean-Marc. Unary and n-ary inclusion dependency discovery in relational databases. *J. Intell. Inf. Syst. 32*, 1 (2009), 53–73.

[224] Marinescu, Paul D, and Candea, George. Lfi: A practical and general library-level fault injector. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks* (2009), IEEE, pp. 379–388.

[225] Martin, Tveten, and Glad, Ingrid K. Online detection of sparse changes in high-dimensional data streams using tailored projections. *arXiv preprint arXiv:1908.02029* (2019).

[226] Mayer, Mikaël, Soares, Gustavo, Grechkin, Maxim, Le, Vu, Marron, Mark, Polozov, Oleksandr, Singh, Rishabh, Zorn, Benjamin G., and Gulwani, Sumit. User interaction models for disambiguation in programming by example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, UIST 2015, Charlotte, NC, USA, November 8-11, 2015* (2015), pp. 291–301.

[227] Meliou, Alexandra, Gatterbauer, Wolfgang, Moore, Katherine F., and Suciu, Dan. WHY so? or WHY no? functional causality for explaining query answers. In *Proceedings of the Fourth International VLDB workshop on Management of Uncertain Data (MUD 2010) in conjunction with VLDB 2010, Singapore, September 13, 2010* (2010), vol. WP10-04 of *CTIT Workshop Proceedings Series*, Centre for Telematics and Information Technology (CTIT), University of Twente, The Netherlands, pp. 3–17.

[228] Meliou, Alexandra, Gatterbauer, Wolfgang, Nath, Suman, and Suciu, Dan. Tracing data errors with view-conditioned causality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011* (2011), pp. 505–516.

[229] Meliou, Alexandra, Roy, Sudeepa, and Suciu, Dan. Causality and explanations in databases. *Proc. VLDB Endow. 7*, 13 (2014), 1715–1716.

[230] Menzies, Tim. Applications of abduction: knowledge-level modelling. *Int. J. Hum. Comput. Stud. 45*, 3 (1996), 305–335.

[231] Mesbah, Ali, Van Deursen, Arie, and Roest, Danny. Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering 38*, 1 (2011), 35–53.

[232] Metzger, Steffen, Schenkel, Ralf, and Sydow, Marcin. QBEES: query-by-example entity search in semantic knowledge graphs based on maximal aspects, diversity-awareness and relaxation. *J. Intell. Inf. Syst. 49*, 3 (2017), 333–366.

[233] Miao, Zhengjie, Zeng, Qitian, Li, Chenjie, Glavic, Boris, Kennedy, Oliver, and Roy, Sudeepa. CAPE: explaining outliers by counterbalancing. *Proc. VLDB Endow. 12*, 12 (2019), 1806–1809.

[234] Micenková, Barbora, Ng, Raymond T., Dang, Xuan-Hong, and Assent, Ira. Explaining outliers by subspace separability. In *ICDM* (2013), pp. 518–527.

[235] Mihalcea, Rada, and Tarau, Paul. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (Barcelona, Spain, July 2004), Association for Computational Linguistics, pp. 404–411.

[236] Mobile prices: `https://www.kaggle.com/iabhishekofficial/mobile-price-classification`, Feb 2020.

[237] Mordelet, Fantine, and Vert, J-P. A bagging svm to learn from positive and unlabeled examples. *Pattern Recognition Letters 37* (2014), 201–209.

[238] Móro, Róbert, and Bieliková, Mária. Personalized text summarization based on important terms identification. In *23rd International Workshop on Database and Expert Systems Applications, DEXA 2012, Vienna, Austria, September 3-7, 2012* (2012), IEEE Computer Society, pp. 131–135.

[239] Mottin, Davide, Lissandrini, Matteo, Velegrakis, Yannis, and Palpanas, Themis. Exemplar queries: a new way of searching. *VLDB J. 25*, 6 (2016), 741–765.

[240] Muşlu, Kıvanç, Brun, Yuriy, and Meliou, Alexandra. Data debugging with continuous testing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (2013), pp. 631–634.

[241] MySQL. `https://www.mysql.com/`.

[242] Nallapati, Ramesh, Zhou, Bowen, dos Santos, Cícero Nogueira, Gülçehre, Çaglar, and Xiang, Bing. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016* (2016), ACL, pp. 280–290.

[243] Narayan, Shashi, Cohen, Shay B., and Lapata, Mirella. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *CoRR abs/1808.08745* (2018).

[244] Neelakantan, Arvind, Roth, Benjamin, and McCallum, Andrew. Compositional vector space models for knowledge base completion. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers* (2015), The Association for Computer Linguistics, pp. 156–166.

[245] Npgsql - .net access to postgresql. https://www.npgsql.org/.

[246] Npgsql bug. https://github.com/npgsql/npgsql/issues/2485.

[247] Oldenburg, Lennart, Zhu, Xiangfeng, Ramasubramanian, Kamala, and Alvaro, Peter. Fixed it for you: Protocol repair using lineage graphs. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings* (2019).

[248] Ouyang, Zhenzheng, Gao, Yuhai, Zhao, Zipeng, and Wang, Tao. Study on the classification of data streams with concept drift. In *International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)* (2011), vol. 3, IEEE, pp. 1673–1677.

[249] Pan, Victor Y., and Chen, Zhao Q. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA* (1999), ACM, pp. 507–516.

[250] Panev, Kiril, Weisenauer, Nico, and Michel, Sebastian. Reverse engineering top-k join queries. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings* (2017), vol. P-265 of *LNI*, GI, pp. 61–80.

[251] Papenbrock, Thorsten, Ehrlich, Jens, Marten, Jannik, Neubert, Tommy, Rudolph, Jan-Peer, Schönberg, Martin, Zwiener, Jakob, and Naumann, Felix. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment 8*, 10 (2015), 1082–1093.

[252] Papenbrock, Thorsten, Kruse, Sebastian, Quiané-Ruiz, Jorge-Arnulfo, and Naumann, Felix. Divide & conquer-based inclusion dependency discovery. *Proceedings of the VLDB Endowment 8*, 7 (2015), 774–785.

[253] Parameswaran, Aditya. Democratizing data science and lessons learned along the way. In *Proceedings of the VLDB 2020 PhD Workshop co-located with the 46th International Conference on Very Large Databases (VLDB 2020), ONLINE, August 31 - September 4, 2020* (2020).

[254] Parnin, Chris, and Orso, Alessandro. Are automated debugging techniques actually helping programmers? In *Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011, Toronto, ON, Canada, July 17-21, 2011* (2011), ACM, pp. 199–209.

[255] Pearl, Judea. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA, 2000.

[256] Pearl, Judea. The algorithmization of counterfactuals. *Ann. Math. Artif. Intell. 61*, 1 (2011), 29–39.

[257] Pearl, Judea, and Verma, Thomas. A theory of inferred causation. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Cambridge, MA, USA, April 22-25, 1991* (1991), pp. 441–452.

[258] Pena, Eduardo HM, de Almeida, Eduardo C, and Naumann, Felix. Discovery of approximate (and exact) denial constraints. *Proceedings of the VLDB Endowment 13*, 3 (2019), 266–278.

[259] Physically strong actors. `https://www.imdb.com/list/ls050159844`.

[260] Psallidas, Fotis, Ding, Bolin, Chakrabarti, Kaushik, and Chaudhuri, Surajit. S4: top-k spreadsheet-style search for query discovery. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), ACM, pp. 2001–2016.

[261] Pu, Yewen, Ellis, Kevin, Kryven, Marta, Tenenbaum, Josh, and Solar-Lezama, Armando. Program synthesis with pragmatic communication.

[262] Python rexpy package. `https://tdda.readthedocs.io/en/v1.0.30/rexpy.html`.

[263] Qahtan, Abdulhakim Ali, Alharbi, Basma, Wang, Suojin, and Zhang, Xiangliang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015* (2015), ACM, pp. 935–944.

[264] Qahtan, Abdulhakim Ali, Tang, Nan, Ouzzani, Mourad, Cao, Yang, and Stonebraker, Michael. ANMAT: automatic knowledge discovery and error detection through pattern functional dependencies. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019* (2019), ACM, pp. 1977–1980.

[265] Qahtan, Abdulhakim Ali, Tang, Nan, Ouzzani, Mourad, Cao, Yang, and Stonebraker, Michael. Pattern functional dependencies for data cleaning. *Proc. VLDB Endow. 13*, 5 (2020), 684–697.

[266] Qian, Li, Cafarella, Michael J., and Jagadish, H. V. Sample-driven schema mapping. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012* (2012), ACM, pp. 73–84.

[267] Qin, Xuedi, Chai, Chengliang, Luo, Yuyu, Tang, Nan, and Li, Guoliang. Interactively discovering and ranking desired tuples without writing SQL queries. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), ACM, pp. 2745–2748.

[268] Quiñonero-Candela, Joaquin, Sugiyama, Masashi, Lawrence, Neil D, and Schwaighofer, Anton. *Dataset shift in machine learning*. Mit Press, 2009.

[269] Random forest classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[270] Rawal, Kaivalya, Kamar, Ece, and Lakkaraju, Himabindu. Can I still trust you?: Understanding the impact of distribution shifts on algorithmic recourses. *CoRR abs/2012.11788* (2020).

[271] Raza, Mohammad, and Gulwani, Sumit. Disjunctive program synthesis: A robust approach to programming by example. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (2018), pp. 1403–1412.

[272] Reimers, Nils, and Gurevych, Iryna. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019* (2019), Association for Computational Linguistics, pp. 3980–3990.

[273] Ren, Zhaochun, Liang, Shangsong, Meij, Edgar, and de Rijke, Maarten. Personalized time-aware tweets summarization. In *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013* (2013), ACM, pp. 513–522.

[274] Rezig, El Kindi, Brahmaroutu, Ashrita, Tatbul, Nesime, Ouzzani, Mourad, Tang, Nan, Mattson, Timothy G., Madden, Samuel, and Stonebraker, Michael. Debugging large-scale data science pipelines using dagger. *Proc. VLDB Endow. 13*, 12 (2020), 2993–2996.

[275] Rezig, El Kindi, Cao, Lei, Simonini, Giovanni, Schoemans, Maxime, Madden, Samuel, Tang, Nan, Ouzzani, Mourad, and Stonebraker, Michael. Dagger: A data (not code) debugger. In *CIDR 2020, 10th Conference on Innovative Data Systems Research, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings* (2020).

[276] Ribeiro, Marco Túlio, Singh, Sameer, and Guestrin, Carlos. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016* (2016), ACM, pp. 1135–1144.

[277] Ribeiro, Marco Túlio, Singh, Sameer, and Guestrin, Carlos. Anchors: High-precision model-agnostic explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (2018), AAAI Press, pp. 1527–1535.

[278] Rößler, Jeremias, Fraser, Gordon, Zeller, Andreas, and Orso, Alessandro. Isolating failure causes through test case generation. In *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012* (2012), ACM, pp. 309–319.

[279] Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[280] Rush, Alexander M., Chopra, Sumit, and Weston, Jason. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015* (2015), The Association for Computational Linguistics, pp. 379–389.

[281] Rutkowski, Leszek, Jaworski, Maciej, Pietruczuk, Lena, and Duda, Piotr. A new method for data stream mining based on the misclassification error. *IEEE Trans. Neural Netw. Learning Syst. 26*, 5 (2015), 1048–1059.

[282] Salimi, Babak, Parikh, Harsh, Kayali, Moe, Getoor, Lise, Roy, Sudeepa, and Suciu, Dan. Causal relational learning. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), pp. 241–256.

[283] Salimi, Babak, Rodriguez, Luke, Howe, Bill, and Suciu, Dan. Interventional fairness: Causal database repair for algorithmic fairness. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019* (2019), ACM, pp. 793–810.

[284] Santolucito, Mark, Goldman, Drew, Weseley, Allyson, and Piskac, Ruzica. Programming by example: Efficient, but not "helpful". In *9th Workshop on Evaluation and Usability of Programming Languages and Tools, PLATEAU@SPLASH 2018, November 5, 2018, Boston, Massachusetts, USA* (2018), pp. 3:1–3:10.

[285] Santolucito, Mark, Hallahan, William T., and Piskac, Ruzica. Live programming by example. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019* (2019).

[286] Saria, Suchi, and Subbaswamy, Adarsh. Tutorial: Safe and reliable machine learning. *CoRR abs/1904.07204* (2019).

[287] Sarma, Anish Das, Parameswaran, Aditya G., Garcia-Molina, Hector, and Widom, Jennifer. Synthesizing view definitions from data. In *Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings* (2010), ACM International Conference Proceeding Series, ACM, pp. 89–103.

[288] Scheines, Richard, Spirtes, Peter, Glymour, Clark, Meek, Christopher, and Richardson, Thomas. The tetrad project: Constraint based aids to causal model specification. *Multivariate Behavioral Research 33*, 1 (1998), 65–117.

[289] Schelter, Sebastian, Rukat, Tammo, and Bießmann, Felix. Learning to validate the predictions of black box classifiers on unseen data. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), pp. 1289–1299.

[290] Schölkopf, Bernhard, Smola, Alexander J, Bach, Francis, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[291] Sentiment 140 dataset. `https://www.kaggle.com/kazanova/sentiment140`.

[292] Sethi, Tegjyot Singh, and Kantardzic, Mehmed M. On the reliable detection of concept drift from streaming unlabeled data. *Expert Syst. Appl. 82* (2017), 77–99.

[293] Sethi, Tegjyot Singh, Kantardzic, Mehmed M., and Arabmakki, Elaheh. Monitoring classification blindspots to detect drifts from unlabeled data. In *IEEE International Conference on Information Reuse and Integration, IRI* (2016), pp. 142–151.

[294] Shen, Yanyan, Chakrabarti, Kaushik, Chaudhuri, Surajit, Ding, Bolin, and Novik, Lev. Discovering queries based on example tuples. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014* (2014), ACM, pp. 493–504.

[295] Shu, Gang, Sun, Boya, Podgurski, Andy, and Cao, Feng. MFL: method-level fault localization with causal inference. In *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013* (2013), IEEE Computer Society, pp. 124–133.

[296] Shvachko, Konstantin, Kuang, Hairong, Radia, Sanjay, and Chansler, Robert. The hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010* (2010), pp. 1–10.

[297] Siddiqui, Tarique, Kim, Albert, Lee, John, Karahalios, Karrie, and Parameswaran, Aditya G. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *Proc. VLDB Endow. 10*, 4 (2016), 457–468.

[298] Siddiqui, Tarique, Luh, Paul, Wang, Zesheng, Karahalios, Karrie, and Parameswaran, Aditya G. Shapesearch: A flexible and efficient system for shape-based exploration of trendlines. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), pp. 51–65.

[299] Song, Shaoxu, and Chen, Lei. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst. 36*, 3 (2011), 16:1–16:41.

[300] Song, Xiuyao, Wu, Mingxi, Jermaine, Christopher M., and Ranka, Sanjay. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007* (2007), ACM, pp. 667–676.

[301] Spirtes, Peter, Glymour, Clark, and Scheines, Richard. *Causation, Prediction, and Search, Second Edition*. Adaptive computation and machine learning. MIT Press, 2000.

[302] Srinivasan, Ashwin. The aleph manual `https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html`.

[303] Subbaswamy, Adarsh, Schulam, Peter, and Saria, Suchi. Preventing failures due to dataset shift: Learning predictive models that transport. In *International Conference on Artificial Intelligence and Statistics, AISTATS* (2019), pp. 3118–3127.

[304] Sumner, William N., and Zhang, Xiangyu. Algorithms for automatically computing the causal paths of failures. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings* (2009), vol. 5503 of *Lecture Notes in Computer Science*, Springer, pp. 355–369.

[305] Szlichta, Jaroslaw, Godfrey, Parke, Golab, Lukasz, Kargar, Mehdi, and Srivastava, Divesh. Effective and complete discovery of order dependencies via set-based axiomatization. *Proc. VLDB Endow. 10*, 7 (2017), 721–732.

[306] Sztyler, Timo, and Stuckenschmidt, Heiner. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)* (2016), pp. 1–9.

[307] Tan, Wei Chit, Zhang, Meihui, Elmeleegy, Hazem, and Srivastava, Divesh. Reverse engineering aggregation queries. *Proc. VLDB Endow. 10*, 11 (2017), 1394–1405.

[308] Tan, Wei Chit, Zhang, Meihui, Elmeleegy, Hazem, and Srivastava, Divesh. REGAL+: reverse engineering SPJA queries. *Proc. VLDB Endow. 11*, 12 (2018), 1982–1985.

[309] Tax, David M. J., and Müller, Klaus-Robert. Feature extraction for one-class classification. In *ICANN/ICONIP* (2003), pp. 342–349.

[310] Thakur, Aditya V., Sen, Rathijit, Liblit, Ben, and Lu, Shan. Cooperative crug isolation. In *Proceedings of the International Workshop on Dynamic Analysis: held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2009), WODA 2009, Chicago, IL, USA, July, 2009.* (2009), pp. 35–41.

[311] Tiwari, Ashish, Dutertre, Bruno, Jovanovic, Dejan, de Candia, Thomas, Lincoln, Patrick, Rushby, John M., Sadigh, Dorsa, and Seshia, Sanjit A. Safety envelope for security. In *International Conference on High Confidence Networked Systems (part of CPS Week), HiCoNS* (2014), pp. 85–94.

[312] Tran, Quoc Trung, Chan, Chee Yong, and Parthasarathy, Srinivasan. Query reverse engineering. *VLDB J. 23*, 5 (2014), 721–746.

[313] Tsymbal, Alexey. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin 106*, 2 (2004), 58.

[314] Tsymbal, Alexey, Pechenizkiy, Mykola, Cunningham, Padraig, and Puuronen, Seppo. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *IEEE International Symposium on Computer-Based Medical Systems (CBMS)* (2006), pp. 679–684.

[315] Tveten, Martin. Which principal components are most sensitive to distributional changes? *CoRR abs/1905.06318* (2019).

[316] Vahabzadeh, Arash, Fard, Amin Milani, and Mesbah, Ali. An empirical study of bugs in test code. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015* (2015), pp. 101–110.

[317] Vapnik, Vladimir, Golowich, Steven E, and Smola, Alex J. Support vector method for function approximation, regression estimation and signal processing. In *NeurIPS* (1997), pp. 281–287.

[318] Varma, Paroma, Iter, Dan, De Sa, Christopher, and Ré, Christopher. Flipper: A systematic approach to debugging training sets. In *Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics* (2017), pp. 1–5.

[319] Varshney, Kush R. Trustworthy machine learning and artificial intelligence. *ACM Crossroads 25*, 3 (2019), 26–29.

[320] Varshney, Kush R., and Alemzadeh, Homa. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big Data 5*, 3 (2017), 246–255.

[321] Völske, Michael, Potthast, Martin, Syed, Shahbaz, and Stein, Benno. Tl;dr: Mining reddit to learn automatic summarization. In *Proceedings of the Workshop on New Frontiers in Summarization, NFiS@EMNLP 2017, Copenhagen, Denmark, September 7, 2017* (2017), Association for Computational Linguistics, pp. 59–63.

[322] Wang, Chenglong, Cheung, Alvin, and Bodík, Rastislav. Synthesizing highly expressive SQL queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017* (2017), ACM, pp. 452–466.

[323] Wang, Heng, and Abraham, Zubin. Concept drift detection for imbalanced stream data. *CoRR abs/1504.01044* (2015).

[324] Wang, Richard C., and Cohen, William W. Language-independent set expansion of named entities using the web. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA* (2007), IEEE Computer Society, pp. 342–350.

[325] Wang, Xiaolan, Dong, Xin Luna, and Meliou, Alexandra. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), pp. 1231–1245.

[326] Wang, Xinyu, Gulwani, Sumit, and Singh, Rishabh. FIDEX: filtering spreadsheet data using examples. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016* (2016), pp. 195–213.

[327] Weeratunge, Dasarath, Zhang, Xiangyu, Sumner, William N., and Jagannathan, Suresh. Analyzing concurrency bugs using dual slicing. In *Proceedings of the Nineteenth International Symposium on Software Testing and Analysis, ISSTA 2010, Trento, Italy, July 12-16, 2010* (2010), ACM, pp. 253–264.

[328] Weiss, Yaacov Y., and Cohen, Sara. Reverse engineering spj-queries from examples. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017* (2017), ACM, pp. 151–166.

[329] Wong, W Eric, and Debroy, Vidroha. A survey of software fault localization. *Department of Computer Science, University of Texas at Dallas, Tech. Rep. UTDCS-45 9* (2009).

[330] Woodward, James. *Making Things Happen: A Theory of Causal Explanation*. Oxford scholarship online. Oxford University Press, 2003.

[331] Word grabbag. `http://wordgrabbag.com`.

[332] Wu, Eugene, and Madden, Samuel. Scorpion: Explaining away outliers in aggregate queries. *Proc. VLDB Endow. 6*, 8 (2013), 553–564.

[333] Wu, Weiyuan, Flokas, Lampros, Wu, Eugene, and Wang, Jiannan. Complaint-driven training data debugging for query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020), pp. 1317–1334.

[334] Xiang, Rongjing, and Neville, Jennifer. Pseudolikelihood EM for within-network relational learning. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy* (2008), IEEE Computer Society, pp. 1103–1108.

[335] Xin, Bin, Sumner, William N., and Zhang, Xiangyu. Efficient program execution indexing. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008* (2008), pp. 238–248.

[336] Xu, Songhua, Jiang, Hao, and Lau, Francis C. M. User-oriented document summarization through vision-based eye-tracking. In *Proceedings of the 14th International Conference on Intelligent User Interfaces, IUI 2009, Sanibel Island, Florida, USA, February 8-11, 2009* (2009), ACM, pp. 7–16.

[337] Yan, Jing Nathan, Schulte, Oliver, Zhang, Mohan, Wang, Jiannan, and Cheng, Reynold. SCODED: statistical constraint oriented data error detection. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), pp. 845–860.

[338] Yasunaga, Michihiro, Kasai, Jungo, Zhang, Rui, Fabbri, Alexander R., Li, Irene, Friedman, Dan, and Radev, Dragomir R. Scisummnet: A large annotated corpus and content-impact models for scientific paper summarization with citation networks. *CoRR abs/1909.01716* (2019).

[339] Yessenov, Kuat, Tulsiani, Shubham, Menon, Aditya Krishna, Miller, Robert C., Gulwani, Sumit, Lampson, Butler W., and Kalai, Adam. A colorful approach to text processing by example. In *The 26th Annual ACM Symposium on User Interface Software and Technology, UIST'13, St. Andrews, United Kingdom, October 8-11, 2013* (2013), pp. 495–504.

[340] Yu, Hwanjo, Han, Jiawei, and Chang, Kevin Chen-Chuan. PEBL: positive example based learning for web page classification using SVM. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada* (2002), ACM, pp. 239–248.

[341] Yu, Shujian, Abraham, Zubin, Wang, Heng, Shah, Mohak, Wei, Yantao, and Príncipe, José C. Concept drift detection and adaptation with hierarchical hypothesis testing. *J. Franklin Institute 356*, 5 (2019), 3187–3215.

[342] Yuan, Ding, Luo, Yu, Zhuang, Xin, Rodrigues, Guilherme Renna, Zhao, Xu, Zhang, Yongle, Jain, Pranay, and Stumm, Michael. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014* (2014), USENIX Association, pp. 249–265.

[343] Zeller, Andreas. Yesterday, my program worked. today, it does not. why? In *Software Engineering - ESEC/FSE'99, 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Toulouse, France, September 1999, Proceedings* (1999), pp. 253–267.

[344] Zeng, Zhong, Lee, Mong-Li, and Ling, Tok Wang. Answering keyword queries involving aggregates and GROUPBY on relational databases. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016* (2016), OpenProceedings.org, pp. 161–172.

[345] Zhang, Meihui, Elmeleegy, Hazem, Procopiuc, Cecilia M., and Srivastava, Divesh. Reverse engineering complex join queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013* (2013), ACM, pp. 809–820.

[346] Zhang, Sai, and Sun, Yuyin. Automatically synthesizing SQL queries from input-output examples. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013* (2013), IEEE, pp. 224–234.

[347] Zhang, Xiangling, Chen, Yueguo, Chen, Jun, Du, Xiaoyong, Wang, Ke, and Wen, Ji-Rong. Entity set expansion via knowledge graphs. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017* (2017), ACM, pp. 1101–1104.

[348] Zhang, Xingxing, Lapata, Mirella, Wei, Furu, and Zhou, Ming. Neural latent extractive document summarization. *arXiv preprint arXiv:1808.07187* (2018).

[349] Zhang, Yunjia, Guo, Zhihan, and Rekatsinas, Theodoros. A statistical perspective on discovering functional dependencies in noisy data. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (2020), pp. 861–876.

[350] Zheng, Alice X., Jordan, Michael I., Liblit, Ben, Naik, Mayur, and Aiken, Alex. Statistical debugging: Simultaneous identification of multiple bugs. In *Proceedings of ICML* (New York, NY, USA, 2006), ICML'06, ACM, pp. 1105–1112.

[351] Zheng, Alice X., Rish, Irina, and Beygelzimer, Alina. Efficient test selection in active diagnosis via entropy approximation. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005* (2005), p. 675.

[352] Zloof, Moshé M. Query-by-example: the invocation and definition of tables and forms. In *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA* (1975), ACM, pp. 1–24.