Example-Driven Query Intent Discovery: Abductive Reasoning using Semantic Similarity

Anna Fariha
College of Information and Computer Sciences
University of Massachusetts Amherst
afariha@cs.umass.edu

Alexandra Meliou
College of Information and Computer Sciences
University of Massachusetts Amherst
ameli@cs.umass.edu

ABSTRACT

Traditional relational data interfaces require precise structured queries over potentially complex schemas. These rigid data retrieval mechanisms pose hurdles for non-expert users, who typically lack language expertise and are unfamiliar with the details of the schema. *Query by Example* (QBE) methods offer an alternative mechanism: users provide examples of their intended query output and the QBE system needs to infer the intended query. However, these 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. In this paper, we present SQuID, a system that performs *semantic similarity-aware query intent discovery*. Our work makes the following contributions: (1) We design an end-to-end system that automatically formulates select-project-join queries in an *open-world* setting, with optional group-by aggregation and intersection operators; a much larger class than prior QBE techniques. (2) We express the problem of query intent discovery using a *probabilistic abduction model*, that infers a query as the most likely explanation of the provided examples. (3) We introduce the notion of an *abduction-ready* database, which precomputes semantic properties and related statistics, allowing SQuID to achieve real-time performance. (4) We present an extensive empirical evaluation on three real-world datasets, including user-intent case studies, demonstrating 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.

PVLDB Reference Format:
Anna Fariha, Alexandra Meliou. Example-Driven Query Intent Discovery: Abductive Reasoning using Semantic Similarity. *PVLDB*, 12(11): 1262-1275, 2019.
DOI: https://doi.org/10.14778/3342263.3342266

1. INTRODUCTION

Database technology has expanded drastically, and its audience has broadened, bringing on a new set of usability requirements. A significant group of current database users are non-experts, such as data enthusiasts and occasional users. These non-expert users want to explore data, but lack the expertise needed to do so. Traditional database technology was not designed with this group of users in mind, and hence poses hurdles to these non-expert users. Traditional query interfaces allow data retrieval through well-structured queries. To write such queries, one needs expertise in the query language (typically SQL) and knowledge of the, potentially complex, database schema. Unfortunately, occasional users typically lack both. Query by Example (QBE) offers an alternative retrieval mechanism, where users specify their intent by providing example tuples for their query output [43].

Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342266

academics

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

research

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

Figure 1: Excerpt of the CS Academics data. Dan Suciu and Sam Madden (in bold), both have research interests in data management.

Unfortunately, traditional QBE systems [49, 46, 14] for relational databases make a strong and oversimplifying 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.[1] This ignores the richer context in the data that can help identify the intended query more effectively.

Example 1.1. In Figure 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 `data management` researchers. However, a QBE system, that looks for queries based only 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. The 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'
```

[1] More nuanced QBE systems exist, but typically place additional requirements or significant restrictions over the supported queries (Figure 3).

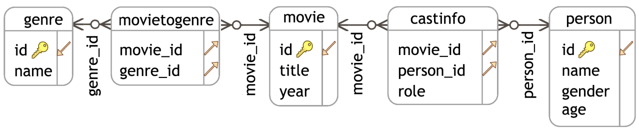


Figure 2: Partial IMDb schema with entity relations `movie` and `person`, and semantic property relation `genre`. Relations `castinfo` and `movietogenre` associate entities and semantic properties.

Example 1.1 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 1.1). Direct attributes capture intent that is *explicit*, precisely specified by the particular attribute values. However, sometimes query intent is more vague, and not expressible by explicit semantic similarity alone. In such cases, the semantic similarity of the example tuples is *implicit*, captured through deeper associations with other entities in the data (e.g., type and quantity of movies an actor appears in).

Example 1.2. The IMDb dataset contains a wealth of information related to the movies and entertainment industry. We query the IMDb dataset (Figure 2) with a QBE system, using two different sets of examples:

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

`ET1` contains the names of three actors from a public list of "physically strong" actors[2]; `ET2` contains the names of three actors from a public list of "funny" actors[3]. `ET1` and `ET2` represent different query intents (*strong* actors and *funny* actors, respectively), but 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 labels an actor as "strong" or "funny". Nevertheless, the database encodes these associations implicitly, in the number and type of movies an actor appears in.[4]

Standard QBE systems typically produce queries that are too general, and fail to capture nuanced query intents, such as the ones in Examples 1.1 and 1.2. Some prior approaches attempt to refine the queries based on additional, external information, such as external ontologies [36], provenance information of the example tuples [14], and user feedback on multiple (typically a large number) system-generated examples [11, 35, 16]. Other work relies on a *closed-world* assumption[5] to produce more expressive queries [35, 54, 62] and thus requires complete examples of input databases and output results. Providing such external information is typically complex and tedious for a non-expert.[6]

In contrast with prior approaches, 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 additional external information*, beyond the initial set of example tuples. SQuID, our semantic similarity-aware query intent discovery framework [22], 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 for the observed results (example tuples). We explain how SQuID uses these insights to handle the challenging scenario of Example 1.2.

Legend

QBE: Query by Example
QRE: Query Reverse Engineering
DX: Data Exploration
KG: Knowledge Graph
!: with significant restrictions

			query class				semi-join	implicit property	scalable	open-world	additional requirements
			join	projection	selection	aggregation					
QBE	relational	**SQuID**	✓	✓	✓	✓	✓	✓	✓	✓	
		Bonifati et al. [11]	✓	✓	!		✓		✓	✓	user feedback
		QPlain [14]	✓	✓	✓		✓		✓	✓	provenance input
		Shen et al. [49]	✓	✓					✓	✓	
		FastTopK [46]	✓	✓					✓	✓	
	KG	Arenas et al. [4]	✓	✓	!		✓		✓	✓	
		SPARQLByE [15]	✓	✓	!		✓		✓	✓	negative examples
		GQBE [28]	✓	✓	!		✓		✓	✓	
		QBEES [41]	✓	✓	!		✓		✓	✓	
QRE	relational	PALEO-J [45]	✓	✓	✓	✓			✓		top-k queries only
		SQLSynthesizer [62]	✓	✓	✓	✓	✓				schema knowledge
		Scythe [54]	✓	✓	✓	✓	✓				schema knowledge
		Zhang et al. [61]	✓						✓	✓	
		REGAL [51]		✓	✓	✓			✓		
		REGAL+ [52]	✓	✓	✓	✓					
		FastQRE [31]	✓	✓					✓	✓	
		QFE [35]	✓	✓	✓						user feedback
		TALOS [53]	✓	✓	✓	✓	✓		✓		
DX	rel.	AIDE [16]			✓				✓	✓	user feedback
		REQUEST [23]			✓				✓	✓	user feedback

Figure 3: SQuID captures complex intents and more expressive queries than prior work in the open-world setting.

Example 1.3. We query the IMDb dataset with SQuID, using the example tuples in `ET2` (Example 1.2). 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
```

In this paper, we make the following contributions:

- We design an end-to-end system, SQuID, that automatically formulates select-project-join queries with optional group-by aggregation and intersection operators (SPJ_{AI}) based on few user-provided example tuples. SQuID does not require the users to have any knowledge of the database schema or the query language. In contrast with existing approaches, SQuID does not need any additional user-provided information, and achieves very high precision with very few examples in most cases.

[2]https://www.imdb.com/list/ls050159844
[3]https://www.imdb.com/list/ls000025701
[4]"Strong" actors frequently appear in action movies, and "funny" actors in comedies.
[5]In the closed-world setting, a tuple not specified as an example output is assumed to be excluded from the query result.
[6]Figure 3 provides a summary exposition of prior work, and contrasts with our contributions. We detail this classification and metrics in our technical report [21] and discuss the related work in Section 8.

- SQuID infers the *semantic similarity* of the example tuples, and models query intent using a collection of *basic* and *derived* semantic property *filters* (Section 3). Prior work has explored the use of semantic similarity in knowledge graph retrieval tasks [63, 41, 28]. However, these prior systems do not directly apply to the relational domain, and do not model implicit semantic similarities, derived from aggregating properties of affiliated entities (e.g., number of comedy movies an actor appeared in).
- We express the problem of query intent discovery using a *probabilistic abduction model* (Section 4). This model allows SQuID to identify the semantic property filters that represent the most probable intent given the examples.
- SQuID achieves real-time performance through an offline strategy that pre-computes semantic properties and related statistics to construct an *abduction-ready* database (Section 5). 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 6). We prove the correctness of the abduction algorithm in Theorem 1.
- Our empirical evaluation includes three real-world datasets, 41 queries covering a broad range of complex intents and structures, and three case studies (Section 7). We further compare with TALOS [53], a state-of-the-art system that captures very expressive queries, but in a closed-world setting. We show that SQuID is more accurate at capturing intent and produces better queries, often reducing the number of predicates by orders of magnitude. We also empirically show that SQuID outperforms a semi-supervised machine learning system [19], which learns classification models from positive examples and unlabeled data.

2. 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.

2.1 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 example tuples can involve a huge search space. 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 and practical impact 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})[7], 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 [49] or only rank them according to the degree of input containment, when the example tuples are not fully contained by the query output [46]. 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.

[7]The SPJ_{AI} queries derived by SQuID limit joins to key-foreign key joins, and conjunctive selection predicates of the form `attribute OP value`, where $\texttt{OP} \in \{=, \geq, \leq\}$ and `value` is a constant.

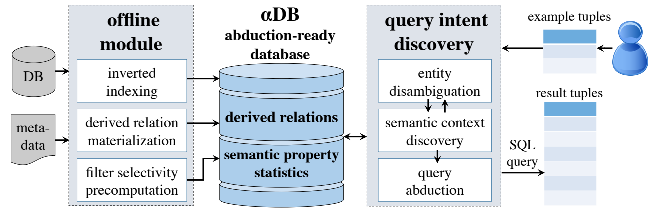


Figure 4: SQuID includes an offline module, which constructs an *abduction-ready* database (αDB) and precomputes statistics of semantic properties. During normal operation, SQuID's query intent discovery module interacts with the αDB to identify the semantic context of the examples and abduces the most likely 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 1.2). Existing QBE solutions focus on the query structure and are thus ill-equipped to capture nuanced intent. 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 $\mathcal{D}$.

Definition 2.1 (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:
- $E \subseteq Q(\mathcal{D})$
- $Q = \text{argmax}_q \, Pr(q|E)$

More informally, 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 $Pr(Q|E)$.

2.2 Abductive Reasoning

SQuID solves the query intent discovery problem (Definition 2.1) using *abduction*. Abduction or abductive reasoning [40, 30, 10, 5] 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. So, a deductive approach would produce all possible queries that contain the example tuples in their results, and it would guarantee that the intended query is one of them. However, the set of valid queries is typically extremely large, growing exponentially with the number of properties and the size of the data domain. In our work, we model query intent discovery as an abduction problem and apply abductive inference to discover the most likely query intent. More formally, given two possible candidate queries, Q and Q', we infer Q as the intended query if $Pr(Q|E) > Pr(Q'|E)$.

Example 2.1. Consider again the scenario of Example 1.1. SQuID identifies that the two example tuples share the semantic context `interest = data management`. `Q1` and `Q2` both contain the example tuples in their result set. However, the probability that two tuples

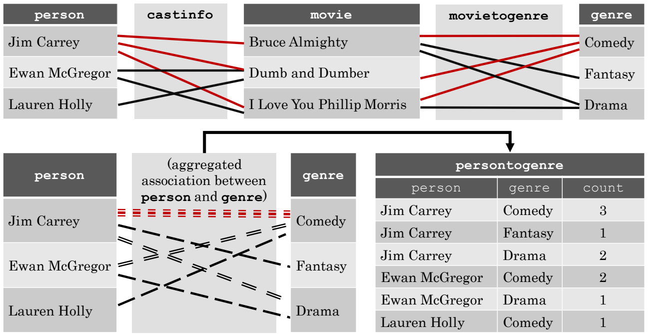


Figure 5: A `genre` value (e.g., `genre=Comedy`) is a basic semantic property of a `movie` (through the `movietogenre` relation). A `person` is associated with `movie` entities (through the `castinfo` relation); aggregates of basic semantic properties of movies are *derived semantic properties* of `person`, 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`.)

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 identified semantic context is high (1.0). Assuming that `Q1` and `Q2` have equal priors ($Pr(Q1) = Pr(Q2)$), then from Bayes' rule $Pr(Q2|E) > Pr(Q1|E)$.

2.3 Solution Sketch

At the core of SQUID is an *abduction-ready database*, αDB (Figure 4). The αDB (1) increases SQUID's efficiency by storing precomputed associations and statistics, and (2) 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 2.2. The IMDb database has, among others, relations `person` and `genre` (Figure 2). SQUID's αDB stores a derived semantic property that associates the two entity types in a new relation, `persontogenre(person.id, genre.id, count)`, which stores how many movies of each genre each person appeared in. SQUID derives this relation through joins with `castinfo` and `movietogenre`, and aggregation (Figure 5). Then, the SPJ_{AI} query `Q4` (Example 1.3) 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_{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. Section 3 formalizes SQUID's model of query intent as a combination of the base query Q^* and a set of semantic property filters. Then, Section 4 analyzes the probabilistic abduction model that SQUID uses to solve the query intent discovery problem (Definition 2.1).

After the formal models, we describe the system components of SQUID. Section 5 describes the offline module, which is responsible for making the database abduction-ready, by precomputing semantic properties and statistics in derived relations. Section 6 describes the query intent discovery module, which abduces the most likely intent as an SPJ query on the αDB.

3. 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`). 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 1.1) would capture the structure of the examples, the semantic context may rely on other relations (e.g., `research`, as in `Q2` of Example 1.1). 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 6.2. 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 1.1 are:

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

3.1 Semantic Properties and Filters

Semantic properties encode characteristics of an entity, and are of two types: (1) A *basic semantic property* is affiliated with an entity directly. In the IMDb schema of Figure 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 semantic property of an associated entity. In Example 2.2, 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[8] or a value range for attribute A associated with entities in R. 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 6, the filters $\phi_{\langle \texttt{gender}, \texttt{Male}, \perp \rangle}$ and $\phi_{\langle \texttt{age}, [50,90], \perp \rangle}$ 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., $\sigma_{\texttt{gender=Male}}(\texttt{person})$ and $\sigma_{50 \leq \texttt{age} \leq 90}(\texttt{person})$. For derived properties, filters specify conditions on the association across different entities. In Example 2.2, for `person` entities, the filter $\phi_{\langle \texttt{genre}, \texttt{Comedy}, 30 \rangle}$ 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., $\sigma_{\texttt{genre=Comedy} \wedge \texttt{count} \geq 30}(\texttt{persontogenre})$.

3.2 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 examples satisfy.

[8] 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.

person			
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 6: Sample database with example tuples

Definition 3.1 (Filter validity)**.** Given a database $\mathcal{D}$, an example tuple set E, and a base query Q^*, a filter ϕ is valid if and only if $Q^{\{\phi\}}(\mathcal{D}) \supseteq E$, where $Q^{\{\phi\}} = (Q^*, \{\phi\})$.

Figure 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 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 $\forall \phi_i \in \Phi\ \phi_i$ 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. Out of all valid filters, SQuID focuses on *minimal* valid filters, which have the tightest bounds.[9]

Definition 3.2 (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 not valid. 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 not valid.

In the example of Figure 6, $\phi_{\langle \texttt{age},[50,90],\perp\rangle}$ is a minimal filter and $\phi_{\langle \texttt{age},[40,90],\perp\rangle}$ is not.

4. PROBABILISTIC ABDUCTION MODEL

We now revisit the problem of Query Intent Discovery (Definition 2.1), and recast it based on our model of query intent (Section 3). Specifically, Definition 2.1 aims to discover an SPJ_{AI} query Q; this is reduced to an equivalent SPJ query Q^φ on the αDB (as in Example 2.2). SQuID's task is to find the query Q^φ that maximizes the posterior probability $Pr(Q^\varphi|E)$, 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.

4.1 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[10], from hereon simply referred to as filters, where ϕ_i encodes the semantic context x_i. 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.

[9]Bounds can be derived in different ways, potentially informed by the result set cardinality. However, we found that the choice of the tightest bounds works well in practice.
[10]We omit $\langle A, V, \theta\rangle$ in the filter notation when the context in clear.

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^*
$x = (p, \|E\|)$	Semantic context of E for p
$\mathcal{X} = \{x_1, x_2, \dots\}$	Set of semantic contexts

Figure 7: Summary of notations

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

4.2 Modeling Query Posterior

We first analyze the probabilistic model for a *fixed base query* Q^* and then generalize the model in Section 4.3. We use $Pr_*(a)$ as a shorthand for $Pr(a|Q^*)$. We model the query posterior $Pr_*(Q^\varphi|E)$, using Bayes' rule:

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

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

$$\begin{aligned} Pr_*(Q^\varphi|E) &= \frac{Pr_*(E, \mathcal{X}|Q^\varphi)Pr_*(Q^\varphi)}{Pr_*(E)} \\ &= \frac{Pr_*(E|\mathcal{X}, Q^\varphi)Pr_*(\mathcal{X}|Q^\varphi)Pr_*(Q^\varphi)}{Pr_*(E)} \end{aligned}$$

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

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

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

$$Pr_*(Q^\varphi|E) = \frac{Pr_*(\mathcal{X}|Q^\varphi)Pr_*(Q^\varphi)}{Pr_*(\mathcal{X})} \tag{1}$$

In Equation 1, we have modeled 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, $Pr_*(\mathcal{X}|Q^\varphi)$. We proceed to analyze each of these components.

4.2.1 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 $x_i \in \mathcal{X}$:

$$Pr_*(\mathcal{X}) \propto \psi(\Phi) \tag{2}$$

Selectivity $\boldsymbol{\psi(\phi)}$. The selectivity of a filter ϕ denotes the portion of tuples from the result of the base query Q^* that satisfy ϕ:

$$\psi(\phi) = \frac{|Q^{\{\phi\}}(\mathcal{D})|}{|Q^*(\mathcal{D})|}$$

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 6, $\phi_{\langle \texttt{gender}, \texttt{Male}, \perp \rangle}$ is more selective than $\phi_{\langle \texttt{age}, [50,90], \perp \rangle}$, 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 2.

4.2.2 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[11], we reduce the query prior to a product of probabilities of filter events:

$$Pr_*(Q^\varphi) = Pr_*(\bigcap_{\phi \in \Phi} \widetilde{\phi}) = \prod_{\phi \in \Phi} Pr_*(\widetilde{\phi}) \tag{3}$$

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.

Domain selectivity impact $\boldsymbol{\delta(\phi)}$. 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 $\delta \in (0, 1]$; $\delta(\phi)$ is equal to 1 for filters that do not exceed a predefined ratio in the coverage of their domain, and decreases for filters that exceed this threshold.[12]

Association strength impact $\boldsymbol{\alpha(\phi)}$. 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 label filters with θ lower than a threshold τ_α as insignificant, and set $\alpha(\phi) = 0$. All other filters, including basic filters, have $\alpha(\phi) = 1$.

Outlier impact $\boldsymbol{\lambda(\phi)}$. While $\alpha(\phi)$ characterizes the impact of association strength on a filter individually, $\lambda(\phi)$ characterizes its impact in consideration with other derived filters over the same attribute. Figure 8 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 are more significant than the other filters of the same family (higher association strength). 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.

We model the outlier impact $\lambda(\phi)$ of a filter using the *skewness* of the association strength distribution within the family of derived filters sharing the same attribute. Our assumption is that highly-skewed, heavy-tailed distributions (Case A) are likely to contain the significant (intended) filters as *outliers*. We set $\lambda(\phi) = 1$ for a derived filter whose association strength is an outlier in the association strength distribution of filters of the same family. We also set $\lambda(\phi) = 1$ for basic filters. All other filters get $\lambda(\phi) = 0$.[12]

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 8: Examples of outlier impact. In Case A, filters ϕ_1 and ϕ_2 are interesting, whereas no filter is interesting in Case B.

4.2.3 Semantic Context Posterior

The semantic context posterior $Pr_*(\mathcal{X}|Q^\varphi)$ 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 $\mathcal{X}$:

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

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

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

Recall that ϕ_i encodes the semantic context x_i (Section 4.1). We assume that x_i is conditionally independent of any $\widetilde{\phi}_j, i \neq j$, given $\widetilde{\phi}_i$ (this always holds for $\widetilde{\phi}_i = \phi_i$):

$$Pr_*(\mathcal{X}|Q^\varphi) = \prod_{i=1}^n Pr_*(x_i|\widetilde{\phi}_i) \tag{4}$$

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

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

$\boldsymbol{Pr_*(x_i|\bar{\phi}_i)}$: 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, $Pr_*(x_i|\bar{\phi}_i) \approx \psi(\phi_i)^{|E|}$.

Using Equations (1)–(4), we derive the final form of the query posterior (where K is a normalization constant):

$$Pr_*(Q^\varphi|E) = \frac{K}{\psi(\Phi)} \prod_{\phi_i \in \Phi} \left(Pr_*(\widetilde{\phi}_i) Pr_*(x_i|\widetilde{\phi}_i) \right)$$

$$= \frac{K}{\psi(\Phi)} \prod_{\phi_i \in \varphi} Pr_*(\phi_i) Pr_*(x_i|\phi_i) \prod_{\phi_i \notin \varphi} Pr_*(\bar{\phi}_i) Pr_*(x_i|\bar{\phi}_i) \tag{5}$$

4.3 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., $Pr(Q^*|Q^\varphi) = 1$. Hence:

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

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

5. OFFLINE ABDUCTION PREPARATION

In this section, we discuss system considerations to perform query intent discovery *efficiently*. SQuID employs an offline module that performs several pre-computation steps to make the database *abduction-ready*. The abduction-ready database (αDB) augments

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

[12] Details on the computation of $\delta(\phi)$ and $\lambda(\phi)$ are in our technical report [21].

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.

Entity lookup. SQUID's goal is to discover the most likely query, 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* [49], built over all text attributes and stored in the αDB, to perform fast lookups, matching the provided example data to entities in the database.

Semantic property discovery. To reason about intent, SQUID first needs to determine what makes the examples similar. It 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 PK-FK 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 may involve several joins and aggregations and performing them at runtime would be prohibitive.[13] For example, SQUID computes the `persontogenre` relation (Figure 5) and stores it in the αDB with the SQL query below:

```
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. It uses (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 derived properties up to a certain pre-defined 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 we found these are not common in practice. SQUID generally 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.

Smart selectivity computation. For basic filters over categorical values, SQUID stores the selectivity for each value. For numeric ranges, SQUID precomputes 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 then derive the selectivity of a filter with any value range as:

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

[13] The data cube [25] can serve as an alternative mechanism to model the αDB data, but is much less efficient compared to the αDB (details are in our technical report [21]).

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 "$A = v$".

6. QUERY INTENT DISCOVERY

During normal operation, SQUID receives example tuples from a user, consults the αDB, and infers the most likely query intent (Definition 2.1). 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.

6.1 Entity and Context Discovery

SQUID's probabilistic abduction model (Section 4) 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.

6.1.1 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 examples are typically few, SQUID can determine the right mappings by considering all combinations.

6.1.2 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

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 $Pr_*(Q^{\varphi}|E)$ is maximized

```
1 𝒳 = {x_1, x_2, ...}                         // semantic contexts in E
2 φ = ∅
3 foreach φ_i ∈ Φ do
4     include_{φ_i} = Pr_*(φ_i)Pr_*(x_i|φ_i)   // from Equation (5)
5     exclude_{φ_i} = Pr_*(φ̄_i)Pr_*(x_i|φ̄_i)   // from Equation (5)
6     if include_{φ_i} > exclude_{φ_i} then
7         φ = φ ∪ {φ_i}
8 return Q^φ
```

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 semantic 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 have appeared in 3 and 5 `Comedy` movies, SQUID will produce the context $(\langle \texttt{genre}, \texttt{Comedy}, 3\rangle, 2)$.

6.2 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`). 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).

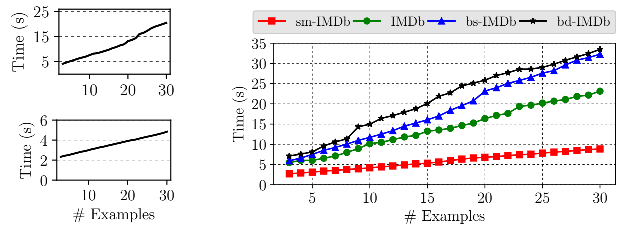
While the number of candidate SPJ queries grows exponentially in the number of minimum valid filters ($2^{|\Phi|}$), we prove that we can make decisions on including or excluding each filter independently. Algorithm 1 iterates over the set of minimal valid filters Φ and decides to include a filter only if its addition to the query increases the query posterior probability (lines 6-7). Our query abduction algorithm has $O(|\Phi|)$ time complexity and is guaranteed to produce the query Q^{φ} that maximizes the query posterior.

Theorem 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 $Pr_*(Q^{\varphi}|E)$ is maximized.[14]

7. EXPERIMENTS

In this section, we present an extensive experimental evaluation of SQUID over three real-world datasets, with a total of 41 benchmark queries of varying complexities. Our results show that SQUID is scalable and effective, even with a small number of example tuples. Our evaluation includes 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.

[14]Proof is provided in our technical report [21].



(a) (b)

Figure 9: Average abduction time over the benchmark queries in (a) IMDb (top), DBLP (bottom), and (b) 4 versions of the IMDb dataset in different sizes.

7.1 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.

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. We summarize the datasets and queries below and provide detailed description in our technical report [21].

IMDb (633 MB): The dataset contains 15 relations with information on movies, cast members, film studios, etc. We designed a set of 16 benchmark queries 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).

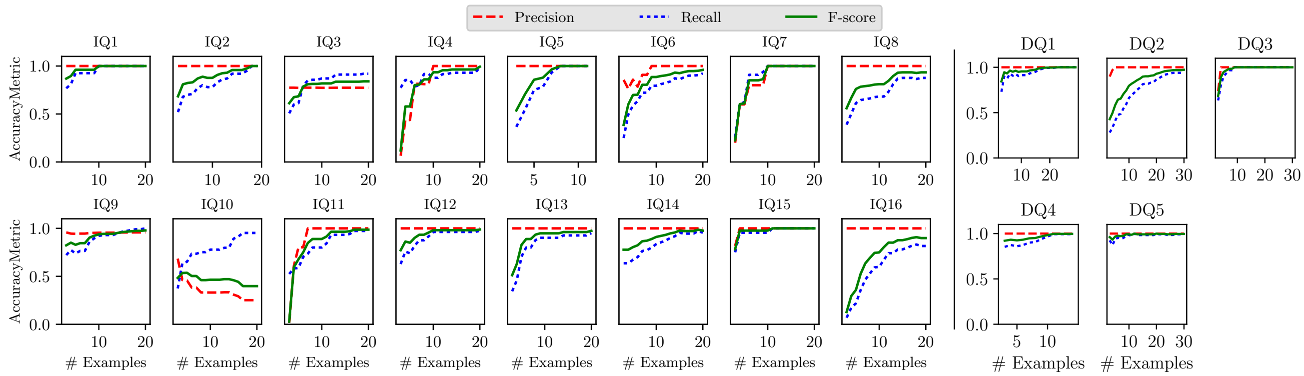
DBLP (22 MB): We used a subset of the DBLP data [2], with 14 relations, and 16 years (2000–2015) of top 81 conference publications. We designed 5 queries ranging the number of joins (3 to 8 relations), the number of selection predicates (2 to 4), and the result cardinality (15 to 468 tuples).

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

Case study data. We retrieved several public lists (sources listed in our technical report [21]) 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).

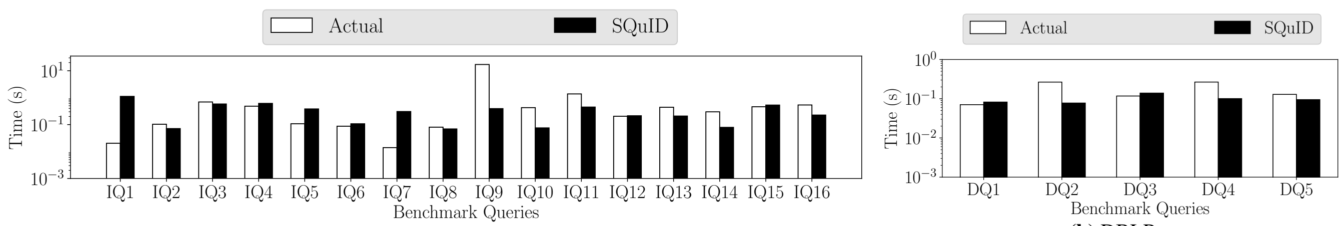
Metrics. We report query discovery time as a metric of efficiency. We measure effectiveness using precision, recall, and f-score. If Q is the intended query, and Q' is the query inferred by SQUID, precision is computed as $\frac{Q'(\mathcal{D}) \cap Q(\mathcal{D})}{Q'(\mathcal{D})}$ and recall as $\frac{Q'(\mathcal{D}) \cap Q(\mathcal{D})}{Q(\mathcal{D})}$; 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.

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 can't meaningfully compare SQUID with those approaches. Removing the open-world requirement, SQUID is most similar to the QRE system TALOS [53] with respect to expressiveness and capabilities (Figure 3). We compare the two systems for query reverse engineering tasks in Section 7.5. We also compare SQUID against PU-learning methods [19] in Section 7.6.



(a) IMDb **(b)** DBLP

Figure 10: SQuID achieves high accuracy with few examples (typically ~ 5) in most benchmark queries.



(a) IMDb **(b)** DBLP

Figure 11: SQuID rarely produces queries that are slower than the original with respect to query runtime.

7.2 Scalability

In our first set of experiments, we examine the scalability of SQuID against increasing number of examples and varied dataset sizes. Figure 9(a) displays the abduction time for the IMDb and DBLP datasets 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 in practice.

Figure 9(b) 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 original 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.[15] 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.

7.3 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 10 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.

[15]Details of the data generation process are in our technical report [21].

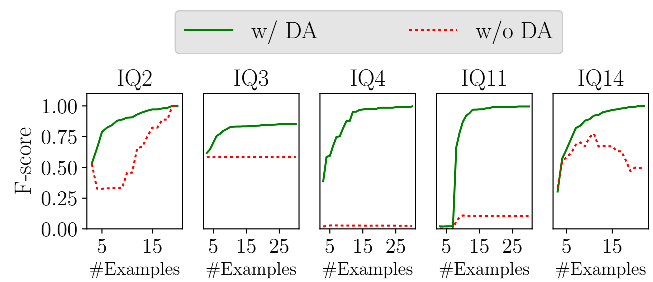


Figure 12: Effect of disambiguation on IMDb

`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 was 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.

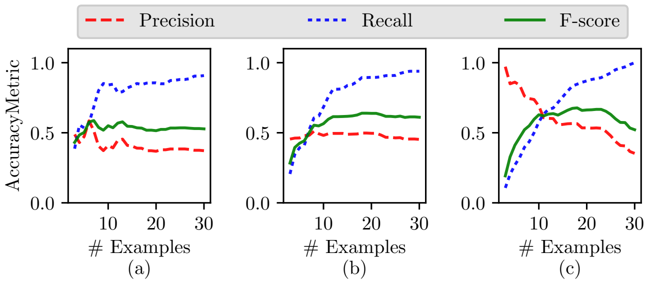


Figure 13: Precision, recall, and f-score for (a) Funny actors (b) 2000s Sci-Fi movies (c) Prolific DB researchers

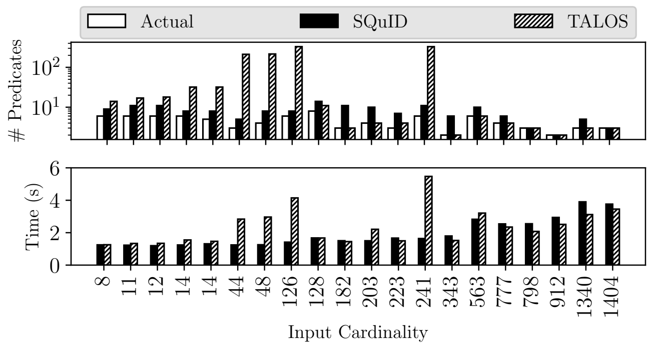


Figure 14: Both systems achieve perfect f-score on Adult (not shown). SQUID produces significantly smaller queries, often by orders of magnitude, and is often much faster.

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 11(a) and 11(b) we graph the average runtime of the abduced queries and the actual benchmark queries. 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.

Effect of entity disambiguation. Finally, we found that entity disambiguation never hurts abduction accuracy, and may significantly improve it. Figure 12 displays the impact of disambiguation for five IMDb benchmark queries, where disambiguation significantly improves the f-score.

7.4 Qualitative Case Studies

In this section, we 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 our technical report [21]), and used these names as examples of the query intent "funny actors." Figure 13(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.

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 13(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 13(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.[16] 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 don't 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.

7.5 Query Reverse Engineering

We present an experimental comparison of SQUID with TALOS [53], a state-of-the-art Query Reverse Engineering (QRE) system.[17] QRE systems operate in a closed-world setting, assuming that the provided examples comprise the entire query output. In contrast, SQUID assumes an open-world setting, and only needs a few examples. In the closed-world setting, SQUID is handicapped against a dedicated QRE system, as it does not take advantage of the closed-world constraint in its inference.

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 [53]. 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.).[18] We adopt the notion of *instance equivalent query* (IEQ) from the QRE literature [53] 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 $\geq$ 0.98 f-score. SQUID failed only for `IQ10`, which is a query that falls outside the supported query family, as discussed in Section 7.3. For the DBLP and Adult datasets, SQUID successfully reverse-engineered all benchmark queries.

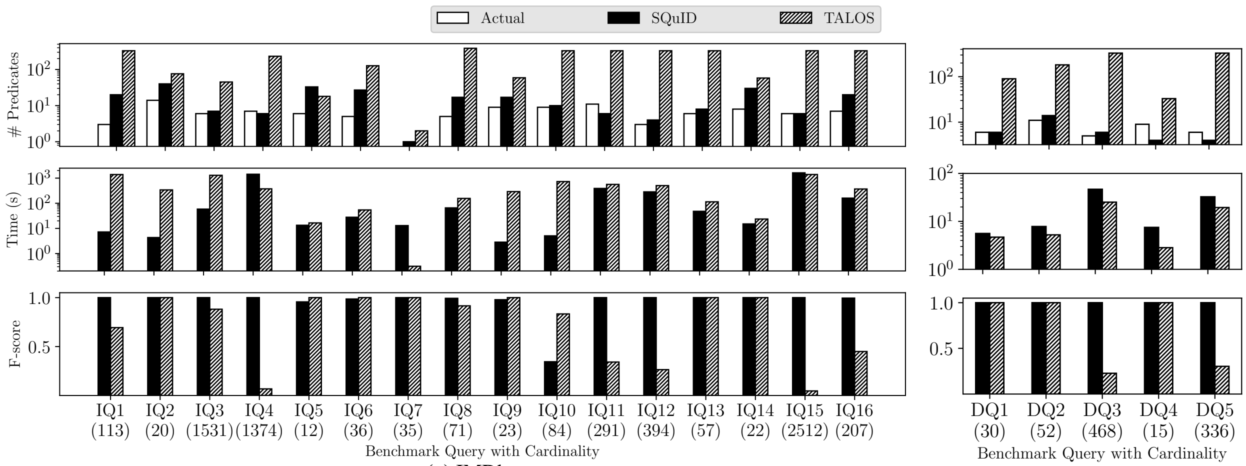
Comparison with TALOS. We compare SQUID to TALOS on three metrics: number of predicates (including join and selection predicates), query discovery time, and f-score.

Adult. Both SQUID and TALOS achieved perfect f-score on the 20 benchmark queries. Figure 14 compares the systems in terms of the

[16] To counter this bias, our case study experiments use popularity masks (derived from public lists) to filter the examples and the abduced query outputs [21].

[17] Other related methods either focus on more restricted query classes [31, 61] or do not scale to data sizes large enough for this evaluation [62, 54] (overview in Figure 3).

[18] Details on the system parameters are in our technical report [21].



(a) IMDb **(b)** DBLP

Figure 15: 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.

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 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 15(a) compares the two systems on the 16 benchmark queries of the IMDb dataset. SQUID produced better queries in almost all cases: 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 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 without a 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 cases 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 15(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.

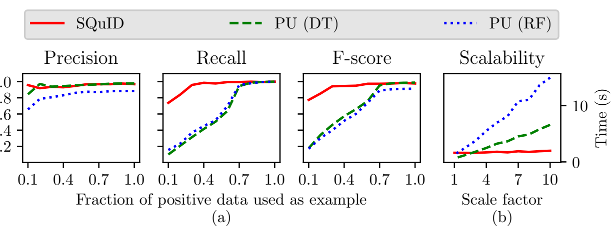


Figure 16: (a) PU-learning needs a large fraction ($>$ 70%) of the query results (positive data) as examples to achieve accuracy comparable to SQUID. (b) The total required time for training and prediction in PU-learning increases linearly with the data size. In contrast, abduction time for SQUID increases logarithmically.

7.6 Comparison with learning methods

Query intent discovery can be viewed 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 setting by learning a classifier from positive examples and unlabeled data in a semi-supervised setting. We compare SQUID against an established PU-learning method [19] on 20 benchmark queries of the Adult dataset. The setting of this experiment conforms with the technique's requirements [19]: the dataset comprises of a single relation and the examples are chosen uniformly at random from the positive data.

Figure 16(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 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 16(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 because 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. Section 8 provides further discussion on the connections between SQUID and machine learning approaches.

8. RELATED WORK

Query-by-Example (QBE) was an early effort to assist users without SQL expertise in formulating SQL queries [64]. Existing QBE systems [49, 46] 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 [14] 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 non-experts.

Set expansion is a problem corresponding to QBE in Knowledge Graphs [63, 55, 57]. SPARQLByE [15], built on top of a SPARQL QRE system [4], allows querying RDF datasets by annotated (positive/negative) example tuples. In semantic knowledge graphs, systems address the entity set expansion problem using maximal-aspect-based entity model, semantic-feature-based graph query, and entity co-occurrence information [36, 28, 26, 41]. 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 derived semantic properties without exploding the graph size. For example, to represent "appearing in more than K comedies", the knowledge graph would require one property for each possible value of K.

Interactive approaches rely on relevance feedback on system-generated tuples to improve query inference and result delivery [1, 11, 16, 23, 35]. Such systems typically expect a large number of interactions, and are often not suitable for non-experts who may not be sufficiently familiar with the data to provide effective feedback.

Query Reverse Engineering (QRE) [56, 6] is a special case of QBE that 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 [53], 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 [31] 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 [61, 31], aggregation without joins [51], or top-k queries [45]. REGAL+[52] handles SPJA queries but only considers the schema of the example tuples to derive the joins and ignores other semantics. In contrast, SQUID considers joining relations without attributes in the example schema (Example 1.1).

A few QRE methods target expressive SPJ queries [62, 54], 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 [54] also expects user hints towards precise discovery of the constants of the query predicates.

Machine learning methods can model QBE settings as classification problems, and relational machine learning targets relational settings in particular [24]. However, while the examples serve as positive labels, QBE settings do not provide explicit negative examples. Semi-supervised statistical relational learning techniques [58] can learn from unlabeled and labeled data, but require *unbiased* sample of negative examples. There is no obvious way to obtain such a sample in our problem setting without significant user effort.

Our problem setting is better handled by one-class classification [38, 32], more specifically, Positive and Unlabeled (PU) learning [59, 37, 9, 19, 8, 42], which learns from positive examples and unlabeled data in a semi-supervised setting [13]. 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 [9] (e.g., all unlabeled entities are negative [44], examples are selected completely at random [19, 7], positive and negative entities are naturally separable [59, 37, 50], similar entities are likely from the same class [33]). These assumptions create a poor fit for our problem setting where the example set is very small, it may exhibit user biases, response should be real-time, and intents may involve deep semantic similarity.

Other approaches that assist users in query formulation involve query recommendation based on collaborative filtering [18], query autocompletion [34], and query suggestion [20, 17, 29]. Another approach to facilitating data exploration is keyword-based search [3, 27, 60]. User-provided examples and interactions appear in other problem settings, such as learning schema mappings [48, 47, 12]. The query likelihood model in IR [39] resembles our technique, but does not exploit the similarity of the input entities.

9. SUMMARY AND FUTURE DIRECTIONS

In this paper, 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 in query reverse engineering (a special case of query intent discovery) and with PU-learning. Our empirical results highlight the flexibility of our method, as it is extremely effective in a broad range of scenarios. Notably, even though SQUID targets query intent discovery with a small set of a examples, it outperforms the state-of-the-art in query reverse engineering in most cases, and is superior to learning techniques.

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 increase sample diversity and improve abduction, techniques for adjusting the depth of association discovery, on-the-fly αDB construction, and efficient αDB maintenance for dynamic datasets.

Acknowledgements: This material is based upon work supported by the NSF under grants CCF-1763423 and IIS-1453543. We thank Quoc Trung Tran for sharing the source code of TALOS.

10. REFERENCES

[1] A. Abouzied, D. Angluin, C. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *PODS*, pages 49–60, 2013.

[2] S. Agarwal, A. Sureka, N. Mittal, R. Katyal, and D. Correa. DBLP records and entries for key computer science conferences, 2016.

[3] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: a system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[4] M. Arenas, G. I. Diaz, and E. V. Kostylev. Reverse engineering SPARQL queries. In *WWW*, pages 239–249, 2016.

[5] O. Arieli, M. Denecker, B. V. Nuffelen, and M. Bruynooghe. Coherent integration of databases by abductive logic programming. *J. Artif. Intell. Res.*, 21:245–286, 2004.

[6] P. Barceló and M. Romero. The Complexity of Reverse Engineering Problems for Conjunctive Queries. In *ICDT*, volume 68, pages 7:1–7:17, 2017.

[7] J. Bekker and J. Davis. Positive and unlabeled relational classification through label frequency estimation. In *ILP*, pages 16–30, 2017.

[8] J. Bekker and J. Davis. Estimating the class prior in positive and unlabeled data through decision tree induction. In *AAAI*, pages 2712–2719, 2018.

[9] J. Bekker and J. Davis. Learning from positive and unlabeled data: A survey. *CoRR*, abs/1811.04820, 2018.

[10] L. E. Bertossi and B. Salimi. Causes for query answers from databases: Datalog abduction, view-updates, and integrity constraints. *Int. J. Approx. Reasoning*, 90:226–252, 2017.

[11] A. Bonifati, R. Ciucanu, and S. Staworko. Learning join queries from user examples. *TODS*, 40(4):24:1–24:38, 2016.

[12] A. Bonifati, U. Comignani, E. Coquery, and R. Thion. Interactive mapping specification with exemplar tuples. In *SIGMOD*, pages 667–682, 2017.

[13] O. Chapelle, B. Schlkopf, and A. Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010.

[14] D. Deutch and A. Gilad. Qplain: Query by explanation. In *ICDE*, pages 1358–1361, 2016.

[15] G. I. Diaz, M. Arenas, and M. Benedikt. SPARQLByE: Querying RDF data by example. *PVLDB*, 9(13):1533–1536, 2016.

[16] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Aide: An active learning-based approach for interactive data exploration. *TKDE*, 28(11):2842–2856, 2016.

[17] M. Drosou and E. Pitoura. Ymaldb: Exploring relational databases via result-driven recommendations. *VLDBJ*, 22(6):849–874, 2013.

[18] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *TKDE*, 26(7):1778–1790, 2014.

[19] C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *SIGKDD*, pages 213–220. ACM, 2008.

[20] J. Fan, G. Li, and L. Zhou. Interactive sql query suggestion: Making databases user-friendly. In *ICDE*, pages 351–362, 2011.

[21] A. Fariha and A. Meliou. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *CoRR*, abs/1906.10322, 2019.

[22] A. Fariha, S. M. Sarwar, and A. Meliou. SQuID: Semantic similarity-aware query intent discovery. In *SIGMOD*, pages 1745–1748, 2018.

[23] X. Ge, Y. Xue, Z. Luo, M. A. Sharaf, and P. K. Chrysanthis. Request: A scalable framework for interactive construction of exploratory queries. In *Big Data*, pages 646–655, 2016.

[24] L. Getoor and B. Taskar. Introduction to statistical relational learning, 2007.

[25] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[26] J. Han, K. Zheng, A. Sun, S. Shang, and J. R. Wen. Discovering neighborhood pattern queries by sample answers in knowledge base. In *ICDE*, pages 1014–1025, 2016.

[27] V. Hristidis and Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[28] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri. Querying knowledge graphs by example entity tuples. *TKDE*, 27(10):2797–2811, 2015.

[29] L. Jiang and A. Nandi. SnapToQuery: Providing interactive feedback during exploratory query specification. *PVLDB*, 8(11):1250–1261, 2015.

[30] A. C. Kakas. Abduction. In *Encyclopedia of Machine Learning and Data Mining*, pages 1–8. Springer, 2017.

[31] D. V. Kalashnikov, L. V. S. Lakshmanan, and D. Srivastava. FastQRE: Fast query reverse engineering. In *SIGMOD*, pages 337–350, 2018.

[32] S. S. Khan and M. G. Madden. A survey of recent trends in one class classification. In *AICS*, pages 188–197. Springer, 2009.

[33] T. Khot, S. Natarajan, and J. W. Shavlik. Relational one-class classification: A non-parametric approach. In *AAAI*, pages 2453–2459, 2014.

[34] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. SnipSuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.

[35] H. Li, C. Chan, and D. Maier. Query from examples: An iterative, data-driven approach to query construction. *PVLDB*, 8(13):2158–2169, 2015.

[36] L. Lim, H. Wang, and M. Wang. Semantic queries by example. In *EDBT*, pages 347–358, 2013.

[37] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu. Building text classifiers using positive and unlabeled examples. In *ICDM*, pages 179–188, 2003.

[38] L. M. Manevitz and M. Yousef. One-class svms for document classification. *JMLR*, 2(Dec):139–154, 2001.

[39] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[40] T. Menzies. Applications of abduction: knowledge-level modelling. *Int. J. Hum.-Comput. Stud.*, 45(3):305–335, 1996.

[41] S. Metzger, R. Schenkel, and M. Sydow. QBEES: query-by-example entity search in semantic knowledge graphs based on maximal aspects, diversity-awareness and relaxation. *J. Intell. Inf. Syst.*, 49(3):333–366, 2017.

[42] F. Mordelet and J.-P. Vert. A bagging svm to learn from positive and unlabeled examples. *Pattern Recognition Letters*, 37:201–209, 2014.

[43] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: A new way of searching. *VLDBJ*, 25(6):741–765, 2016.

[44] A. Neelakantan, B. Roth, and A. McCallum. Compositional vector space models for knowledge base completion. In *ACL*, pages 156–166, 2015.

[45] K. Panev, N. Weisenauer, and S. Michel. Reverse engineering top-k join queries. In *BTW*, pages 61–80, 2017.

[46] F. Psallidas, B. Ding, K. Chakrabarti, and S. Chaudhuri. S4: Top-k spreadsheet-style search for query discovery. In *SIGMOD*, pages 2001–2016, 2015.

[47] L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In *SIGMOD*, pages 73–84, 2012.

[48] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *ICDT*, pages 89–103, 2010.

[49] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *SIGMOD*, pages 493–504, 2014.

[50] A. Srinivasan. The aleph manual.

[51] W. C. Tan, M. Zhang, H. Elmeleegy, and D. Srivastava. Reverse engineering aggregation queries. *PVLDB*, 10(11):1394–1405, 2017.

[52] W. C. Tan, M. Zhang, H. Elmeleegy, and D. Srivastava. REGAL+: reverse engineering SPJA queries. *PVLDB*, 11(12):1982–1985, 2018.

[53] Q. T. Tran, C. Chan, and S. Parthasarathy. Query reverse engineering. *VLDBJ*, 23(5):721–746, 2014.

[54] C. Wang, A. Cheung, and R. Bodik. Synthesizing highly expressive sql queries from input-output examples. In *PLDI*, pages 452–466, 2017.

[55] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM*, pages 342–350, 2007.

[56] Y. Y. Weiss and S. Cohen. Reverse engineering spj-queries from examples. In *PODS*, pages 151–166, 2017.

[57] Word grabbag. `http://wordgrabbag.com`, 2018.

[58] R. Xiang and J. Neville. Pseudolikelihood EM for within-network relational learning. In *ICDM*, pages 1103–1108, 2008.

[59] H. Yu, J. Han, and K. C. Chang. PEBL: positive example based learning for web page classification using SVM. In *SIGKDD*, pages 239–248, 2002.

[60] Z. Zeng, M. Lee, and T. W. Ling. Answering keyword queries involving aggregates and GROUPBY on relational databases. In *EDBT*, pages 161–172, 2016.

[61] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *SIGMOD*, pages 809–820, 2013.

[62] S. Zhang and Y. Sun. Automatically synthesizing sql queries from input-output examples. In *ASE*, pages 224–234, 2013.

[63] X. Zhang, Y. Chen, J. Chen, X. Du, K. Wang, and J. Wen. Entity set expansion via knowledge graphs. In *SIGIR*, pages 1101–1104, 2017.

[64] M. M. Zloof. Query-by-example: The invocation and definition of tables and forms. In *PVLDB*, pages 1–24, 1975.