Predicting Fact Contributions from Query Logs with Machine Learning

Dana Arad
Tel Aviv University
danaarad1@mail.tau.ac.il

Daniel Deutch
Tel Aviv University
danielde@post.tau.ac.il

Nave Frost
eBay Research
nafrost@ebay.com

ABSTRACT

A recent line of work has proposed to quantify the contribution of database tuples to query answers using Shapley values, a game theoretic function that has been extensively used as means of attribution in other areas, notably Machine Learning.

In this paper we analyze and evaluate LearnShapley, a solution that employs Machine Learning to rank input facts based on their estimated (Shapley-based) contribution to query answers. LearnShapley is trained on a corpus of SPJU queries, their output and the Shapley values of each input tuple with respect to each output tuple. At inference time, LearnShapley is given a new SPJU query over the same database schema, an output tuple of interest, and its lineage (i.e. the set of all facts that have contributed in some way to the generation of the tuple). Our experiments evaluate to what extent LearnShapley is able to leverage similarity measures applied to the query in hand and the queries stored in the repository, to compute a ranking of the facts in the lineage based on their contribution. Overall, our experiments indicate that a log of past queries, output tuples and their Shapley values includes a reasonably relevant signal for predicting the ranking of facts contributions for a new SPJU query over the same database. Both DBShap and our code are publicly available, and may serve for further investigation of Machine Learning approaches for explainability in databases.

1 INTRODUCTION

Explaining query answers has been extensively studied in recent years, following many different approaches [1, 14, 21, 29, 39, 40]. A notable such approach is to quantify the contribution of each input database fact to each tuple in the query result. Here again, many different contribution measures have been proposed [29, 31, 33, 40]. In particular, [30] and subsequent works [15, 37], have advocated for the use of Shapley values [42], a game-theoretic function for distributing wealth in a cooperative game. Shapley values have strong theoretical justifications, have been extensively used in Game Theory and have been adopted as a contribution measure in a variety of areas, notably explaining the predictions of Machine Learning classifiers.

Example 1.1. Our running example focuses on a database of movies, an instance of which is shown in Figure 1. Consider the query q_{inf} in Figure 2a, looking for movies released in 2007 and produced by American production companies. For a given tuple in the query result, many different database facts contribute in some way to its derivation. For q_{inf} and the output tuple `Alice`, one such derivation involves a_1 (the tuple representing `Alice`), r_1 (the role of `Alice` in `Superman`), m_1 (`Superman` was produced in 2007 by `Universal`) and c_1 (`Universal` is an American company).

`movies`

	name	year	company
m_1	Superman	2007	Universal
m_2	Spiderman	2007	Universal
m_3	Aquaman	2007	Warner
m_4	Batman	2008	Warner
m_5	Wonder Woman	2008	Disney

`roles`

	actor	movie
r_1	Alice	Superman
r_2	Alice	Spiderman
r_3	Alice	Aquaman
r_4	Alice	Batman
r_5	Bob	Aquaman
r_6	Bob	Batman
r_7	David	Aquaman
r_8	David	Batman
r_9	Carol	Wonder Woman

`companies`

	company	country
c_1	Universal	USA
c_2	Warner	USA
c_3	Disney	USA

`actors`

	name	age
a_1	Alice	45
a_2	Bob	23
a_3	Carol	65
a_4	David	30

Figure 1: DB Instance

Other derivations exist as well, involving information on other movies, production companies and roles.

The set of all facts appearing in at least one derivation is called the tuple's *lineage*. The lineage may contain many database facts, and may be hard to interpret. In particular, some facts contribute more than others to the derivation of the output tuple. In our example, `Universal` is more influential than `Warner` since Alice participated in two movies produced by `Universal` and only in one produced by `Warner`. Measuring the contribution is far more complex in general, and the measure of Shapley values has been shown to be a good fit for quantifying the contribution in query answering [30]. For this example, the Shapley values of all facts in the `companies` table with respect to each of the query's results are presented as heat-maps in Figure 2d.

Shapley values can help us gain insights on the importance of facts with respect to tuples in the query result, yet computing Shapley values for query answering is intractable in general (specifically, it is $FP^{\#P}$-hard [30] in data complexity). Existing solutions [15] rely on detailed query provenance, captured by boolean expressions which are further processed into particular forms. Storing such detailed form of provenance for every query and every output tuple typically entails a non-negligble overhead, despite significant advancements in provenance tracking such as [1, 2, 41]. The solution in [15] is confined to Select-Project-Join-Union (SPJU) queries, and while the complexity of computing Shapley values has been studied for other fragments of SQL such as negation and aggregation, there are, to our knowledge, no available implementation for SQL fragments beyond SPJU. See Section 6 for a detailed overview of previous solutions for Shapley values computation.

In this experiments and analysis paper we ask whether a Machine Learning approach, specifically LearnShapley that was first

```
SELECT DISTINCT  actors.name
FROM movies, actors, companies, roles
WHERE movies.title = roles.movie AND
actors.name = roles.actor AND
movies.company = companies.name AND
companies.country = "USA" AND
 movies.year = 2007
```

(a) Inference query q_{inf}

```
SELECT DISTINCT  movies.title
FROM movies, actors, companies, roles
WHERE movies.title = roles.movie AND
actors.name = roles.actor AND
movies.company = companies.name AND
companies.country = "USA" AND
 movies.year = 2007  AND
 actors.name = "Alice"
```

(b) Log query q_1

```
SELECT DISTINCT  actors.name
FROM movies, actors, companies, roles
WHERE movies.title = roles.movie AND
actors.name = roles.actor AND
movies.company = companies.name AND
companies.country = "USA" AND
 actors.age > 30
```

(c) Log query q_2

	Alice	Bob	David
c_1	0.158	0	0
c_2	0.075	0.25	0.25
c_3	0	0	0

(d) q_{inf} Shapley values

	Superman	Spiderman	Aquaman
c_1	0.25	0	0.25
c_2	0	0.25	0
c_3	0	0	0

(e) q_1 Shapley values

	Alice	Carol
c_1	0.115	0
c_2	0.115	0
c_3	0	0.25

(f) q_2 Shapley values

Figure 2: Inference query along with two queries from the historic query log, and the Shapley values of production companies facts with respect to all output tuples. Differences between the queries are color-highlighted.

introduced in a demonstration paper [5] (including no experiments or analysis), can serve as an alternative to the algorithmic approach, in the context of Shapley values for SPJU Query Answering. The goal of LearnShapley is, given a SPJU query and the lineage of its output fact, to *rank* facts appearing in the lineage based on their (hidden) contribution. Thereby, through an investment in training time (see below), LearnShapley circumvents the computational overhead at inference time incurred by the previously proposed exact computation algorithm, both in the sense that it relies on a simpler input (lineage vs. provenance) and in the sense that inference itself is significantly faster. The main question that we ask here, is to what extent does the predicted ranking match the gold ranking which is based on the real Shapley values. The reason for our focus on SPJU is that, as explained above, there is no existing tractable implementation for other SQL fragments. This means that for fragments beyond SPJU there is no training data available for learning, and also no baselines to test the results against.

Dataset Used in Training, Experiments and Analysis. Training and experiments are carried over a repository of SPJU queries (Select, Project, Join, Union), answers and their real Shapley values (called DBShap), which consists of two databases, IMDB and Academic, a total of 293 SPJU queries having a total of one million results and 18 million contributing facts along with their Shapley value information. We used splits of this dataset for training and for testing LearnShapley. Note that the generation of DBShap involves running the exact algorithm of [15] over the detailed provenance of queries, yet this is done offline and not at deployment. In contrast, at deployment time LearnShapley has access only to the query lineage.

Pre-training Objectives. Many Machine Learning approaches follow the pre-training / fine-tuning paradigm, in which the model is trained on intermediate tasks in order to learn fundamental concepts that will later improve the performance on the end task. In order to obtain signals from historic queries, one should identify query characteristics that are similar between a new unseen query and the historic queries. To this end, we define several pre-training objectives, based on three key notions of query similarity: syntactic similarity, the similarity of the queries' results. For the latter, we define a novel metric for rank-based similarity that captures how similar are the contributions of facts to the queries' output tuples.

Example 1.2. Assume an historic query log that consists of queries q_1 and q_2, depicted in Figure 2b and 2c. q_1 outputs titles of movies released in 2007, produced by American production companies, in which Alice played a role. q_2 returns the names of actors over 30 years old, that played a role in a movie produced by a American production company. Syntactic similarities and differences to the query of interest (q_{inf}) are presented in Figure 2a.

The results of q_1 are the movies `Superman`, `Spiderman` and `Aquaman`, and the results of q_2 are the actors `Alice` and `Carol`. Figure 2e and 2f depict the Shapley values of facts from the `companies` table with respect to each of the queries results.

The historic query log along with the historic Shapley values can aid in providing insights on the influential facts for new queries. For instance, based on the Shapley values of q_1, we observe that the production company associated with c_1 contributed significantly to two of the movies that Alice participated in; the fact c_2 also contributed to one of the query results (`Spiderman`), whereas c_3 had no contribution to any result. In addition, based on the Shapley values of q_2, we can deduce that the production companies associated with facts c_1 and c_2 are influential for `Alice`, while c_3 is influential for `Carol`. Going back to the q_{inf}, based on the insights learned from q_2, we may predict that `Universal` (c_1) and `Warner` (c_2) are dominant production companies with respect to `Alice`. Furthermore, based on the insights obtained from q_1, one may learn that c_1 has a higher contribution than c_2.

Experiments. We then conduct an extensive experimental study, examining the performance of LearnShapley as well as different ablations thereof with respect to the DBShap benchmark. We compare the rankings that LearnShapley outputs with the gold ranking that uses the actual Shapley values, using standard measures of comparing ranked lists (nCDG and precision at k). We show that LearnShapley consistently achieves good results with respect to both metrics. Training a model incurs the (offline) overhead of several days for the databases in DBShap. Training is of course not needed by an algorithmic solution such as in [15]. On the other hand, LearnShapley has the benefit of relying at deployment only on the query lineage (set of contributing tuples) rather than detailed boolean provenance. Furthermore, the inference time of LearnShapley to predict ranking of contribution based on the lineage typically incurs only split seconds and is significantly faster than exact computation (even if we assume that the algorithm of [15] is given access to the detailed boolean provenance). We conclude that overall, once the model is deployed, it constitutes a fast solution for real-time ranking of facts contribution in query answering.

Limitations of LearnShapley. LearnShapley is a system for *in-domain learning*, namely the queries used in training and the queries used in testing/deployment should be on the same database schema. The system aims at picking up signals from the contribution of facts to past queries, generalizing to predict their contribution to new queries. We analyze how well LearnShapley performs this generalization. We further investigate: what about facts that were not seen at training time? Can LearnShapley reasonably predict their relative order of contribution? Our results indicate that some signal exists and is utilized in this respect, yet there is a significant room for improvement, calling for further research in this respect. Such research could proceed in two directions: improving the prediction quality for unseen facts for the given schema, and generalization to a new schema. In terms of expected input, the model expects the lineage as input at inference time. This constitutes a significant step forward compared to previous work [15] requiring full detailed provenance, yet it would be desirable to further generalize LearnShapley so that even the lineage is not needed. The challenge here is that LearnShapley is currently trained on positive samples, i.e. facts with non-zero Shapley value. Thus, the system is not able to accurately differentiate between contributing and non-contributing facts, which is essential for predicting the ranking of any arbitrary set of facts from the database.

The structure of this paper is as follows. We start by recalling the necessary preliminaries in Section 2. In Section 3, we recall the problem of learning to rank facts contribution in query answering, and the components of LearnShapley. In Section 4 we provide statistics and examples on the training and test data, and in Section 5 we detail the design and results of our experimental study. We overview related work in Section 6 and conclude in Section 7.

2 PRELIMINARIES

We next recall the necessary preliminaries on relational databases, lineage and provenance, Shapley values computation [15], query similarities [6]and [5], and present extended versions for examples from [5].

2.1 Relational Databases, Lineage and Provenance

We follow the convention in [30] and subsequent works on Shapley values, and use the term *facts* to refer to tuples from the input database (whose contributions we wish to assess), and the term *tuples* to refer to tuples in the query answer (with respect to which the contribution is made). Then, a relation $R = \{t_1, \ldots, t_k\}$ is a finite set of facts, and a database $\mathcal{D} = \biguplus_{i \in I} R_i$ is a disjoint union of a finite set of relations. A query q is a function that takes as input a database $\mathcal{D}$ and a outputs a set of tuples, i.e., $q(\mathcal{D}) = \{t'_1, \ldots, t'_r\}$.

To explain query results, we associate database facts with *unique annotations*, serving as their identifiers. Two commonly used forms of explanations are then lineage and provenance, which vary in their granularity. Lineage only shows the facts that were used in the evaluation of the query and contributed to the inclusion of t in $q(\mathcal{D})$, while provenance also reveals the relationships between those contributing facts. Formally, for a database $\mathcal{D}$, query q, and output tuple t, the provenance of (q, t) on $\mathcal{D}$, denoted $\mathsf{Prov}(\mathcal{D}, q, t)$, is a Boolean function that represents the dependence of tuple t on the input facts of $\mathcal{D}$. Intuitively, $\mathsf{Prov}(\mathcal{D}, q, t)$ captures the conditions under which t appears in the query result. The lineage of (q, t) on $\mathcal{D}$, denoted $\mathsf{Lineage}(\mathcal{D}, q, t)$, is the set of variables in $\mathsf{Prov}(\mathcal{D}, q, t)$. There are existing methods for capturing both the provenance and lineage of (q, t) on $\mathcal{D}$ during the evaluation of $q(\mathcal{D})$ [2, 16, 41]. Since lineage does not require capturing the dependencies between different facts, its capture tends to be more efficient. For example, in [17] the authors show that provenance may be significantly compressed when equating some variables, which is lower-bounded by the lineage size.

Example 2.1. Recall the database $\mathcal{D}$ depicted in Figure 1 and note now the annotations that are attached to facts (the annotations are r_i for $i = 1, \ldots, 9$, m_i for $i = 1, \ldots, 5$, c_i for $i = 1, \ldots, 4$ and a_i for $i = 1, \ldots 4$). The query $q_{\inf}$ depicted in Figure 2a. For output tuple Alice the provenance is:

$$\begin{aligned}\mathsf{Prov}(\mathcal{D}, q_{\inf}, \text{Alice}) =& (a_1 \wedge m_1 \wedge c_1 \wedge r_1) \vee \\ & (a_1 \wedge m_2 \wedge c_1 \wedge r_2) \vee \\ & (a_1 \wedge m_3 \wedge c_2 \wedge r_3). \\ \mathsf{Lineage}(\mathcal{D}, q_{\inf}, \text{Alice}) =& \{a_1, m_1, m_2, m_3, c_1, c_2, r_1, r_2, r_3\}.\end{aligned}$$

2.2 Shapley values in query answering

The Shapley value [42], a game-theoretic measure of wealth distribution with strong theoretical foundations [38], has recently been suggested as a method for evaluating the contribution of input facts to query answers [30]. Let q be a query, $t \in q(\mathcal{D})$ be an output tuple, and $f \in \mathcal{D}$ be a fact. Denote $q_t(\mathcal{D})$ as a Boolean function returning 1 if and only if $t \in q(\mathcal{D})$. The Shapley value of f in $\mathcal{D}$ for query (q, t), denoted $\mathsf{Shapley}(\mathcal{D}, q, t, f)$, is defined as:

$$\mathsf{Shapley}(\mathcal{D}, q, t, f) \overset{\text{def}}{=} \sum_{E \subseteq \mathcal{D} \setminus \{f\}} \frac{|E|!(|\mathcal{D}| - |E| - 1)!}{|\mathcal{D}|!} \Big(q_t(E \cup \{f\}) - q_t(E) \Big).$$

Intuitively, the value $\mathsf{Shapley}(\mathcal{D}, q, t, f)$ represents the contribution of f to the query's output t.

Example 2.2. Recall that Example 1.1 explained intuitively why

$$\mathsf{Shapley}(\mathcal{D}, q_{\inf}, \text{Alice}, c_1) > \mathsf{Shapley}(\mathcal{D}, q_{\inf}, \text{Alice}, c_2).$$

Next, we will exemplify it more formally. Recall the provenance of `Alice` from Example 2.1. For simplicity we will ignore facts in $\mathcal{D}$ that are not part of $\mathsf{Lineage}(\mathcal{D}, q_{\inf}, \text{Alice})$, as they have no contribution to the result. To calculate $\mathsf{Shapley}(\mathcal{D}, q_{\inf}, \text{Alice}, c_2)$, we need to consider all $E \subseteq \mathcal{D} \setminus \{c_2\}$ such that $q_t(E \cup \{c_2\}) - q_t(E) \neq 0$. This requires that $\{a_1, m_3, r_3\} \subseteq E$ and neither the clauses $(a_1 \wedge m_1 \wedge c_1 \wedge r_1)$ and $(a_1 \wedge m_2 \wedge c_1 \wedge r_2)$ are satisfied. In total, there is one such set of size 3 ($\{a_1, m_3, r_3\}$), 5 sets of size 4 (e.g., $\{a_1, m_3, c_1, r_3\}$), 10 sets of size 5 (e.g., $\{a_1, m_1, m_3, c_1, r_3\}$), 8 sets of size 6 (e.g., $\{a_1, m_1, m_2, m_3, c_1, r_3\}$), and a single set of size 7 ($\{a_1, m_1, m_2, m_3, r_1, r_2, r_3\}$). Thus,

$$\mathsf{Shapley}(\mathcal{D}, q_{\inf}, \text{Alice}, c_2) = \frac{3! \cdot 5!}{9!} + 5 \cdot \frac{4! \cdot 4!}{9!} + 10 \cdot \frac{5! \cdot 3!}{9!} + 8 \cdot \frac{6! \cdot 2!}{9!} + \frac{7! \cdot 1!}{9!} = \frac{19}{252} \approx 0.075.$$

Similarly, left for the reader, it can be shown that

$$\mathrm{Shapley}(\mathcal{D}, q_{\inf}, \mathrm{Alice}, c_1) = 2 \cdot \frac{3! \cdot 5!}{9!} + 10 \cdot \frac{4! \cdot 4!}{9!} + 19 \cdot \frac{5! \cdot 3!}{9!} + 15 \cdot \frac{6! \cdot 2!}{9!} + 3 \cdot \frac{7! \cdot 1!}{9!} = \frac{10}{63} \approx 0.158.$$

It is important to note that calculating the Shapley value for most queries is computationally intractable [30]. However, a recent study [15] has demonstrated an effective approach for computing Shapley values that relies on query provenance. This solution shows that, if provenance can be transformed into a specific circuit form, Shapley values can be computed in polynomial time. However, it is important to note that the approach in [15] may introduce a significant overhead over the native query computation. Capturing query provenance, which is required by this method, may introduce a significant overhead, and the obtained boolean formula must also be processed in an additional time-consuming step. In this work, we use the solution in [15] in an offline manner over a query log to generate training data for our model. During inference, we provide predictions on the ordering of influential facts without the need to capture provenance at all.

2.3 Query Similarity Metrics

To predict the contribution of facts to the results of a given target query, LearnShapley relies on training data that includes Shapley values with respect to the answers of queries from a historic query log. To learn from these past queries, we need first to assess the similarity of these queries to the target query. Many similarity metrics have been proposed for computing query similarity, the quality of which heavily depends on the exact task, database schema and other factors. We utilize two existing notions of query similarity - syntax-based and witness based.

Syntax-Based Similarity: We follow [24] which defined the similarity of two queries using set similarity. A query q is a represented as a set of operations, namely $operations(q) = \{op_1, op_2, ..., op_k\}$ where op_i is either a projection, a selection or a join:

- A projection operation $\Pi_{R.c}$ is defined by the relation R and column c that the data is projected onto.
- A selection operation $\sigma_{R.c,\varphi}$ is defined by a relation R, a column c, and a Boolean condition φ that is applied on the relation column.
- A equi-join operation $\bowtie_{R_1.c_1=R_2.c_2}$ is defined by two pairs of relations and columns, R_1, c_1 and R_2, c_2 that are compared.

Two operations are equal if they are of the same type and have the same features.

Finally, $\mathsf{sim}_s(q, q')$ is the set similarity of q and q' using Jaccard similarity:

$$\mathsf{sim}_s(q, q') = \frac{|operations(q) \cap operations(q')|}{|operations(q) \cup operations(q')|}.$$

Example 2.3. We wish to calculate the syntax-based similarity of the inference query $q_{\inf}$ depicted in Figure 2a and q_1 depicted in Figure 2b. The queries differentiate in their projection operations, $\Pi_{actors.name}$ and $\Pi_{movie.title}$, and q_1 contains an additional selection operation $\sigma_{actors.name=\text{"}Alice\text{"}}$, as highlighted in Figure 2. Using operation set representation:

$$operations(q_{\inf}) = \{\Pi_{actors.name}, \bowtie_{movies.title=roles.movie}, \bowtie_{actors.name=roles.actor}, \bowtie_{movies.company=companies.name}, \sigma_{companies.country=\text{"}USA\text{"}}, \sigma_{movies.year=2007}\}$$

$$operations(q_1) = \{\Pi_{movie.title}, \bowtie_{movies.title=roles.movie}, \bowtie_{actors.name=roles.actor}, \bowtie_{movies.company=companies.name}, \sigma_{companies.country=\text{"}USA\text{"}}, \sigma_{movies.year=2007}, \sigma_{actors.name=\text{"}Alice\text{"}}\}$$

The similarity of the queries is the Jaccard similarity of the operation sets:

$$\mathsf{sim}_s(q_{\inf}, q_1) = \frac{|operations(q_{\inf}) \cap operations(q_1)|}{|operations(q_{\inf}) \cup operations(q_1)|} = \frac{5}{8}$$

Witness-Based Similarity: Another similarity metric can be defined using the notion of witnesses [6], where $witnesses(q) = q(\mathcal{D})$. The similarity between two queries q and q' is computed as the set similarity of the witnesses sets, using Jaccard similarity

$$\mathsf{sim}_w(q, q') = \frac{|witnesses(q) \cap witnesses(q')|}{|witnesses(q) \cup witnesses(q')|} = \frac{|q(\mathcal{D}) \cap q'(\mathcal{D})|}{|q(\mathcal{D}) \cup q'(\mathcal{D})|}.$$

Example 2.4. Now, we wish to calculate the witness-based similarity of queries $q_{\inf}$ and q_1. However, since the queries have different projection operations, they do not share any witnesses and $\mathsf{sim}_w(q_{\inf}, q_1) = 0$. Instead, we calculate the witness-based similarity metric on $q_{\inf}$ and q_2 depicted in Figure 2c, which equals $\frac{1}{4}$.

$$\mathsf{sim}_s(q_{\inf}, q_2) = \frac{|witnesses(q_{\inf}) \cap witnesses(q_2)|}{|witnesses(q_{\inf}) \cup witnesses(q_2)|} = \frac{1}{4}$$

2.4 Language Models

Modern day language models follow the pre-training/fine-tuning paradigm, and pre-trained language models have achieved notable success on various natural language tasks. Recent lines of work have also demonstrated how these pre-trained models can be applied to other fields, and specifically to database-related tasks [13, 22, 47]. During the pre-training stage, a model is trained on auxiliary tasks that help the model learn meaningful representations of the input. In this work, we employ BERT [18] and utilize the contextualize representations learned during pre-training, which after further fine-tuning enables the model to comprehend semantic and syntactic aspects of queries and database structure.

The input for BERT consists of two sentences, s_1 and s_2 separated by a special token `[SEP]`. Another special token, `[CLS]`, is added before the first sentence. BERT is jointly pre-trained on two objectives: (1) masked language modeling, where the model has to reconstruct masked tokens from the input, and (2) next sentence prediction, where the model predicts whether s_2 is entailed from s_1. The output of the final hidden state corresponding to the `[CLS]` token is used as the representation of the full input sequence, and can be passed to subsequent classification and regression tasks, as in next-sentence prediction. Different pre-training objectives can affect the performance of the model on downstream tasks, and thus should be constructed to assist the model in learning meaningful signals [13, 26]. In this work we construct three pre-training objectives based on different notions of query similarity, that help the model learn meaningful representations, as demonstrated in Section 5.5.

3 LEARNSHAPLEY

This section recaps (and extends, where needed) the description of LearnShapley from [5], serving as further preliminaries for

```
SELECT DISTINCT  actors.age
FROM movies, actors, companies, roles
WHERE movies.title = roles.movie AND
actors.name = roles.actor AND
movies.company = companies.name AND
companies.country = "USA" AND
 movies.year = 2007
```

(a) query q_3

	45	23	30
c_1	0.158	0	0
c_2	0.075	0.25	0.25
c_3	0	0	0

(b) q_3 Shapley values

Figure 3: q_3, a variation of q_{inf} with a different projection clause. Although the results of q_3 and q_{inf} contain different output tuples, their computation and contributing facts are identical. These signals are captured by the rank-based similarity.

our novel contributions which are the experiments and analysis that follow.

3.1 Problem Definition

Given a database $\mathcal{D}$, a query log Q is a set of quartets (q, t, f, v) where q is a query, t is an output tuple of q (i.e. $t \in q(\mathcal{D})$), $f \in \mathcal{D}$ is a fact, and $v = \text{Shapley}(\mathcal{D}, q, t, f)$. Our goal is to train a model on Q so that given a new query $q \notin Q$, an output tuple $t \in q(\mathcal{D})$, and the *lineage* $\text{Lineage}(q, \mathcal{D}, t)$, output a ranked list of the facts in $\text{Lineage}(q, \mathcal{D}, t)$ whose ranking is similar as possible to their ranking according to the (hidden) Shapley values with respect to t.

For example, recall Example 2.1: for the query q_{inf} and output tuple `Alice`. The Shapley values of some facts in $\text{Lineage}(\mathcal{D}, q_{\text{inf}}, \text{Alice})$ are shown in Example 2.2. Our goal is predicting a ranking of all such facts, which will be close (as defined in Section 5.2) to their ranking according to their actual Shapley values.

The solution focuses on in-domain learning, meaning that the train, dev and test queries are over the same database. This corresponds to a common scenario in which a DBA has a log of past queries and needs to handle new queries over the same database. We also focus on a setting where the lineage (but not the detailed provenance) is given. Other settings are left for future work and briefly discussed in Section 7.

3.2 Query Similarity

Learning to predict query similarity can help our model identify latent aspects in the query, and better associate new queries with the signals from the query log. While syntax-based and witness-based similarities capture meaningful aspects of query similarity, they do not capture signals regarding the reasoning, i.e. contributing facts, that led to the appearance of tuples in the query's results. For example, equivalent queries will have perfect witness-based but may have a low syntax-based score. Moreover, queries that are almost identical, except for the projection operation, will have a witness-based score of 0, but may still provide signals since their computation is almost identical. We define a novel query similarity metric, rank-based similarity, in order to capture this different and complementing notion of similarity.

Rank-Based Similarity: We present a novel method of evaluating query similarity that is based on the ranking of facts' Shapley values with respect to output tuples. This is rooted in the observation that two output tuples sets may be very different, yet still informative for each other. The Shapley values, and thus the ranking, are available at training time, and the idea is to use this information to have the model learn which queries yield similar ranking of facts. Then, at inference time (where the ranking is of course not available), the learned similarity of queries with this respect can help the model focus on inferring a ranking based on the most relevant queries.

Example 3.1. Figure 3 shows q_3, which outputs the ages of actors that played in movies released in 2007 and produced by American production companies. q_3 is similar to q_{inf} (shown in Figure 2a) in a way that witness-based and syntax-based similarity fail to capture: Since q_3 and q_{inf} have different projection clauses, their witness-based similarity is 0. Their syntax-based similarity is high, $\frac{5}{7}$, as only one operation is changed. However, this is also the case if we examine q_{inf} and a variation of q_{inf} with a $movie.year != 2007$, for which the computational reasoning is completely different. q_3 and q_{inf} share the same computation process up until the projection clause, meaning they share information on the provenance of each output tuple, and the ranking of the facts that contributed to the appearance of these tuples in the result. When comparing q_{inf} and q_3, the facts that contributed to the appearance of `45` in q_3 are identical to that of `Alice` in q_{inf}, and again for `23` and `30` with respect to `Bob` and `David`.

Since queries can be similar in their computation process, but have different projection clause, the first task is aligning the output tuples sets based on similarity of their contributing facts rankings:

Given two queries q and q', we define the complete bipartite graph, $G = (q(\mathcal{D}) \cup q'(\mathcal{D}), E)$, where $q(\mathcal{D}) = \{t_1, \ldots, t_n\}$ $q'(\mathcal{D}) = \{t'_1, \ldots, t'_m\}$ are the outputs of queries q and q' over $\mathcal{D}$. Let us denote by $rank_{t_i}(\mathcal{D}, q)$ the ranking of all facts $f \in \mathcal{D}$, ranked by their Shapley values with respect to the output tuple t_i of query q. We define a weight function $w : E \to [0, 1]$ as

$$w(t_i, t'_j) = 1 - K_d(rank_{t_i}(\mathcal{D}, q), rank_{t'_j}(\mathcal{D}, q')).$$

Where K_d is the normalized Kendall tau distance. We proceed to find a maximum weighted matching of (G, w), denoted as $\mathcal{M}$, which can be computed efficiently using the Hungarian algorithm [23]. The similarity of q and q' is then defined as

$$\text{sim}_r(q, q') = \frac{\sum_{e \in \mathcal{M}} w(e)}{|q(\mathcal{D})| + |q'(\mathcal{D})| - |\mathcal{M}|}.$$

We find an alignment of the queries' results, such that each matched pair has the most similar facts ranking. The closer the similarity is to one, the greater the size of the matching and the quality of each individual match altogether.

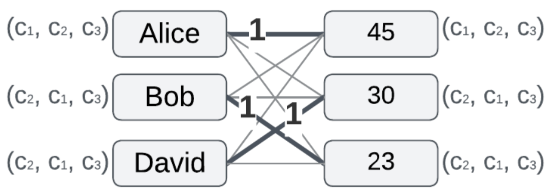


Figure 5: A maximum weighted matching on $G = (q_{\text{inf}}(\mathcal{D}) \cup q_3(\mathcal{D}), E)$. Facts ranking presented besides the nodes.

Example 3.2. We will demonstrate the computation of the rank-based similarity metric for queries q_{inf} and q_3. First, we construct the complete bipartite graph where the nodes in each side consists of the output tuples of each query q_{inf} and q_3, as can be seen in Figure 5. For brevity we will use the facts ranking of `companies` table. In practice, we use all facts in the union of the lineages of all output tuples from both queries. We proceed

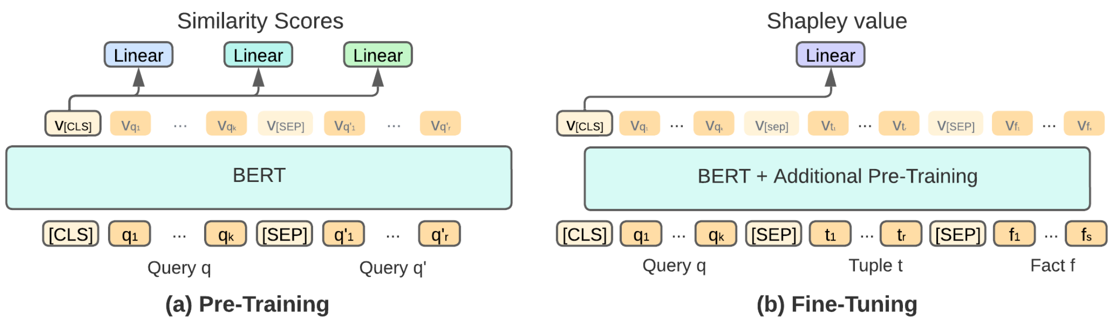


Figure 4: Overview of LearnShapley's system architecture. During pre-training (a), LearnShapley takes two queries as input, and outputs the syntax, witness and rank-based similarity scores. Then, LearnShapley is fine-tuned (b) for Shapley value prediction, which will be used to rank the contributing facts with respect to a specific query and output tuple.

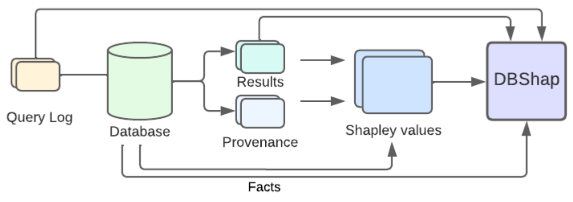


Figure 6: DBShap data collection pipeline. Queries are executed using a provenance-aware database to obtain the output tuples and detailed provenance. Shapley values of contributing facts are computed using the provenance w.r.t. each query and output tuple.

to find a maximum weighted matching $\mathcal{M}$, the edges of $\mathcal{M}$ are marked in bold, and the weight of each edge is denoted (note that there exist other maximum weighted matchings). We then have:

$$\mathrm{sim}_r(q_{\mathrm{inf}}, q_3) = \frac{\sum_{e \in \mathcal{M}} w(e)}{|q_{\mathrm{inf}}(\mathcal{D})| + |q_3(\mathcal{D})| - |\mathcal{M}|} = \frac{3}{3} = 1$$

Note that computing this metric requires exact Shapley values and may be costly. In our system, it is used mainly in the training process. Thus, the computation is done offline and does not affect the inference time.

3.3 Model Architecture

Since the queries, tuples and facts are textual, LearnShapley utilizes BERT (Bidirectional Encoder Representations from Transformers) [18], a self-supervised technique that learns contextualized representations of sequences of text. BERT is a pre-trained model, meaning that the model is trained on an intermediate task with the goal of learning meaningful representations of textual inputs, and can later be fine-tuned, refining its performance towards many different target tasks. We employ BERT in order to learn contextualized representations of queries and facts. Thus, we extend BERT's training methodology, and propose additional pre-training objectives, to refine the models language capabilities to the setting of queries and tuples. Our pre-training objectives aim to help the model learn different aspects of query structure and similarity. Then, we fine-tune the model with the task of predicting the exact Shapley value of a fact with respect to a query and output tuple. We use the predicted Shapley value to rank the facts according to their contribution.

Pre-Training: We define our pre-training objective as follows: we create three signals based on the similarities defined in Section 3.2: rank-based similarity denoted as sim_r, witness-based similarity denoted as sim_w and syntax-based similarity denoted as sim_s. Given two queries q and q', and their tokenized representation $q = q_1, \ldots, q_k$ and $q' = q'_1, \ldots, q'_r$, the input for the model is the sequence of tokens

$$\texttt{[CLS]}, q_1, \ldots, q_k, \texttt{[SEP]}, q'_1, \ldots, q'_r.$$

Where `[CLS]` and `[SEP]` tokens are the special tokens described by [18]. BERT returns a sequence of contextualized vectors

$$v_{\texttt{[CLS]}}, v_{q_1}, \ldots, v_{q_k}, v_{\texttt{[SEP]}}, v_{q'_1}, \ldots, v_{q'_r}.$$

We add three separate linear layers on top of the $v_{\texttt{[CLS]}}$, each corresponding to a single similarity metric. We use regression loss and train the model to predict $\mathrm{sim}_r(q, q')$, $\mathrm{sim}_w(q, q')$ and $\mathrm{sim}_s(q, q')$, respectively.

We define the pre-training loss as an aggregation of the losses of each individual objective (α, β, γ are hyper-parameters, we found equal weights to work best):

$$\ell_{\text{pre-training}} = \alpha \cdot \ell_{\mathrm{sim}_s} + \beta \cdot \ell_{\mathrm{sim}_r} + \gamma \cdot \ell_{\mathrm{sim}_w}$$

Fine-Tuning: In order to rank the influencing facts, we further fine tune the model to predict the Shapley value of a fact f with respect to a query q and an output tuple t. Given the tokenized representation of each element, $q = q_1, \ldots, q_k$, $t = t_1, \ldots, t_r$, and $f = f_1, \ldots, f_s$, we construct the input for the model:

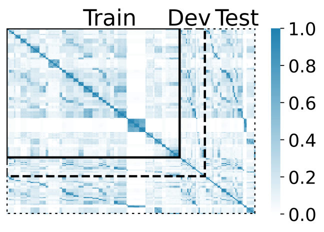
$$\texttt{[CLS]}, q_1, \ldots, q_k, \texttt{[SEP]}, t_1, \ldots, t_r, \texttt{[SEP]}, f_1, \ldots, f_s$$

We then add a linear layer on top of the contextual representation of the `[CLS]` token, and further train the model using regression loss. In this stage, we use the Shapley values scaled by a magnitude of 1000 in order to avoid numeric issues.

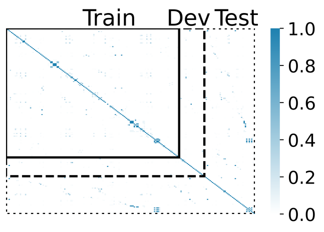
4 TRAINING AND EVALUATION DATA

We train and evaluate LearnShapley using a dataset of SPJU queries over two databases, IMDB and Academic. For each triplet of query, output tuple and fact, DBShap also includes the exact Shapley value of the fact with respect to the query and output tuple, computed using [15]. The dataset, called DBShap, includes queries, query results and contributing facts (in our prior work the solution was demonstrated over queries from DBShap but not tested over it, and its description was correspondingly only partial). The data collection pipeline is detailed in Figure 6. For each pair of queries, DBShap also contains query similarity scores

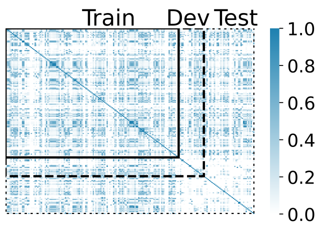
IMDB



(a) Syntax

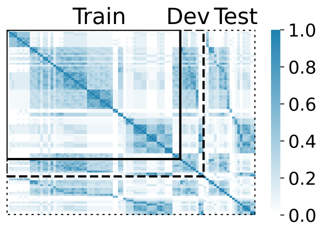


(b) Witness

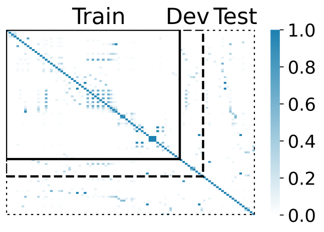


(c) Rank

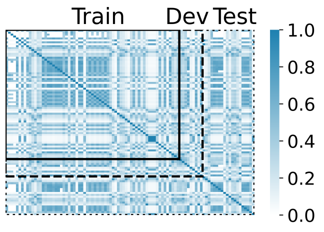
Academic



(d) Syntax



(e) Witness



(f) Rank

Figure 7: Heatmaps of DBShap queries similarities

computed according to the syntax-based, witness-based and rank-based similarities defined in Section 3.2. All queries in DBShap are in SPJU (Select, Project, Join, Union), since the solution in [15] is restricted to this query class.

Overall, DBShap includes 293 unique queries, yielding a total of 1M output tuples, and 18M contributing facts. Following common practice, we randomly split the queries into train (70%), dev (10%) and test (20%). Each split includes the queries along with their results and contributing facts. DBShap is publicly available in [3].

Table 1: DBShap statistics

		train	dev	test	Total
IMDB	# queries	137	20	40	197
	# results	923,688	7,085	39,018	969,791
	# facts	16,588,223	95,992	1,136,501	17,820,716
	avg results per query	6,742.25	354.25	975.45	4,922.80
	max results for a query	306,735	2,556	15,078	306,735
	avg contributing facts per result	17.96	13.55	29.13	18.37
	max contributing facts for a result	261	224	233	261
Academic	# queries	67	9	20	96
	# results	27,080	792	2,174	30,046
	# facts	195,850	8,779	26,508	231,137
	avg results per query	404.18	88	108.7	312.98
	max results for a query	10,640	625	594	10,640
	avg contributing facts per result	7.23	11.08	12.19	7.69
	max contributing facts for a result	262	95	165	262

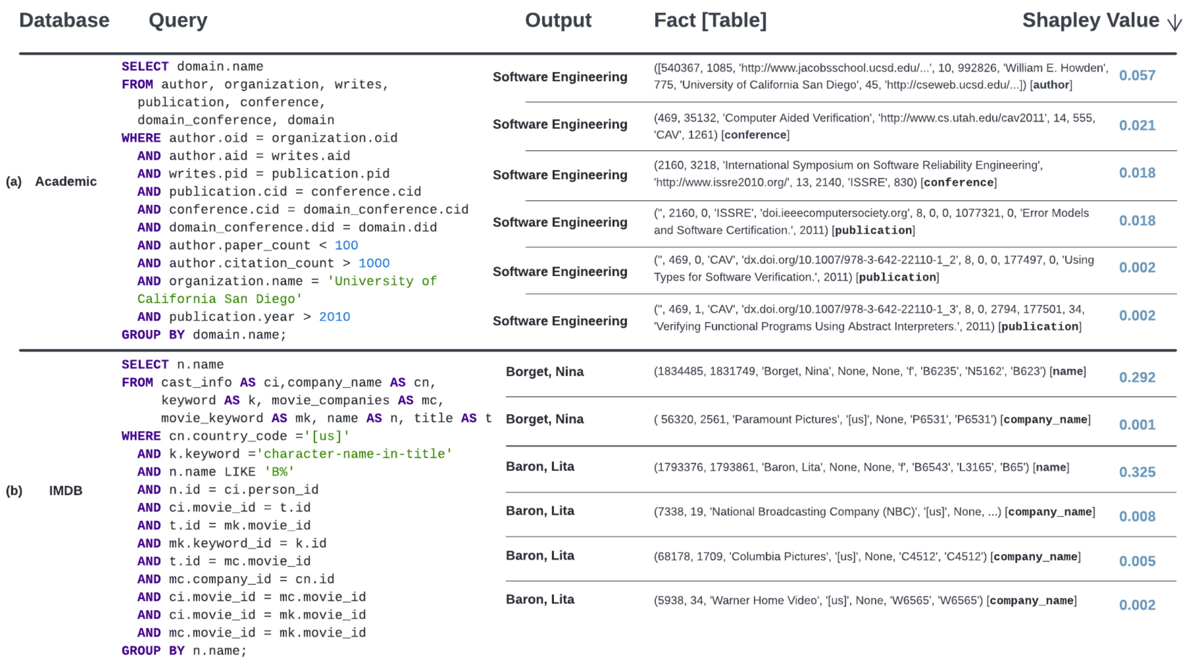
Table 2: Average query similarities in splits of DBShap

		Average Query Similarity			
		train-train	train-dev	train-test	All Queries
IMDB	Syntax-Based Similarity	0.144	0.140	0.146	0.150
	Witness-Based Similarity	0.012	0.002	0.006	0.015
	Rank-Based Similarity	0.179	0.185	0.171	0.145
Academic	Syntax-Based Similarity	0.204	0.241	0.171	0.208
	Witness-Based Similarity	0.030	0.010	0.010	0.033
	Rank-Based Similarity	0.321	0.312	0.308	0.323

DBShap Statistics: On average, the IMDB queries have around 5k results per query, and around 18 contributing facts per result. The Academic queries have an average of around 312 results per query and around 8 contributing facts per result. Both databases include some queries and output tuples with over 200 contributing facts. Queries, results and facts statistics are summarized in Table 1.

Table 2 summarizes statistics on the query similarity scores in DBShap. The table presents the average query similarity between different data splits, i.e. the average similarity of train queries with respect to train, dev and test queries, respectively. Figure 7 shows a heatmap representation of these similarity scores. The heatmaps demonstrate the notion of orthogonality between the different metrics, and show that each captures different characteristics of query similarities as different areas of the grid are activated.

Examples from DBShap: Figure 8 shows examples of queries, output tuples and a small subset of facts and corresponding Shapley values from DBShap. Query (a) over the Academic database outputs the domain names for which there are conferences that have publications published after 2010 by authors from University of California San Diego, with less than 100 publications and a total of more than 1000 citations. We focus on one of the query results, "Software Engineering". Examples of contributing facts include a tuple for a particularly prolific author, and of relevant conferences in which papers were published (CAV and ISSRE). Query (b), over IMDB, outputs the names of people whose name start with the letter "B" and participated in the cast of movies produced by American production companies. Consider the output tuple representing the actress Lita Baron. Baron participated in the movie "Dale Robertson", which has only one release, by NBC. By comparison, the 1949 adventure film "Bomba on Panther Island" featuring Baron has many releases over the years, only one of which was by the Warner Home Video production company. The Shapley values of NBC and Warner Home Video with respect to the query and output tuple (shown in Figure 8) reflect this: NBC has a greater contribution compared to Warner Home Video company for this output tuple.



(a) Academic

```sql
SELECT domain.name
FROM author, organization, writes,
  publication, conference,
  domain_conference, domain
WHERE author.oid = organization.oid
  AND author.aid = writes.aid
  AND writes.pid = publication.pid
  AND publication.cid = conference.cid
  AND conference.cid = domain_conference.cid
  AND domain_conference.did = domain.did
  AND author.paper_count < 100
  AND author.citation_count > 1000
  AND organization.name = 'University of
  California San Diego'
  AND publication.year > 2010
GROUP BY domain.name;
```

Output	Fact [Table]	Shapley Value ↓
Software Engineering	([540367, 1085, 'http://www.jacobsschool.ucsd.edu/...', 10, 992826, 'William E. Howden', 775, 'University of California San Diego', 45, 'http://cseweb.ucsd.edu/...]) [**author**]	0.057
Software Engineering	(469, 35132, 'Computer Aided Verification', 'http://www.cs.utah.edu/cav2011', 14, 555, 'CAV', 1261) [**conference**]	0.021
Software Engineering	(2160, 3218, 'International Symposium on Software Reliability Engineering', 'http://www.issre2010.org/', 13, 2140, 'ISSRE', 830) [**conference**]	0.018
Software Engineering	(", 2160, 0, 'ISSRE', 'doi.ieeecomputersociety.org', 8, 0, 0, 1077321, 0, 'Error Models and Software Certification.', 2011) [**publication**]	0.018
Software Engineering	(", 469, 0, 'CAV', 'dx.doi.org/10.1007/978-3-642-22110-1_2', 8, 0, 0, 177497, 0, 'Using Types for Software Verification.', 2011) [**publication**]	0.002
Software Engineering	(", 469, 1, 'CAV', 'dx.doi.org/10.1007/978-3-642-22110-1_3', 8, 0, 2794, 177501, 34, 'Verifying Functional Programs Using Abstract Interpreters.', 2011) [**publication**]	0.002

(b) IMDB

```sql
SELECT n.name
FROM cast_info AS ci,company_name AS cn,
     keyword AS k, movie_companies AS mc,
     movie_keyword AS mk, name AS n, title AS t
WHERE cn.country_code ='[us]'
  AND k.keyword ='character-name-in-title'
  AND n.name LIKE 'B%'
  AND n.id = ci.person_id
  AND ci.movie_id = t.id
  AND t.id = mk.movie_id
  AND mk.keyword_id = k.id
  AND t.id = mc.movie_id
  AND mc.company_id = cn.id
  AND ci.movie_id = mc.movie_id
  AND ci.movie_id = mk.movie_id
  AND mc.movie_id = mk.movie_id
GROUP BY n.name;
```

Output	Fact [Table]	Shapley Value ↓
Borget, Nina	(1834485, 1831749, 'Borget, Nina', None, None, 'f', 'B6235', 'N5162', 'B623') [**name**]	0.292
Borget, Nina	(56320, 2561, 'Paramount Pictures', '[us]', None, 'P6531', 'P6531') [**company_name**]	0.001
Baron, Lita	(1793376, 1793861, 'Baron, Lita', None, None, 'f', 'B6543', 'L3165', 'B65') [**name**]	0.325
Baron, Lita	(7338, 19, 'National Broadcasting Company (NBC)', '[us]', None, ...) [**company_name**]	0.008
Baron, Lita	(68178, 1709, 'Columbia Pictures', '[us]', None, 'C4512', 'C4512') [**company_name**]	0.005
Baron, Lita	(5938, 34, 'Warner Home Video', '[us]', None, 'W6565', 'W6565') [**company_name**]	0.002

Figure 8: Selected data samples from DBShap with a subset of influential facts

Table 3: LearnShapley results compared to Nearest Query baselines and LearnShapley ablations. Highest scores are marked in bold, second highest marked with underline.

		IMDB				Academic			
	Model	NDCG@10	P@1	P@3	P@5	NDCG@10	P@1	P@3	P@5
LearnShapley	BERT-large + Pre-Training + Fine-Tuning	**0.952**	**0.938**	**0.932**	**0.947**	**0.972**	**0.963**	**0.900**	**0.878**
	BERT-base + Pre-Training + Fine-Tuning	<u>0.948</u>	<u>0.925</u>	<u>0.930</u>	<u>0.945</u>	<u>0.968</u>	<u>0.958</u>	<u>0.900</u>	<u>0.873</u>
Ablations	BERT base + Fine-Tuning	0.944	0.888	0.924	0.943	0.952	0.953	0.868	0.872
	Transformer Encoder	0.928	0.846	0.884	0.922	0.927	0.898	0.861	0.871
Nearest Queries	Syntax-Based Similarity (n=3)	0.679	0.447	0.467	0.421	0.291	0.049	0.073	0.068
	Witness-Based Similarity (n=3)	0.679	0.444	0.462	0.399	0.291	0.043	0.078	0.70
	Rank-Based Similarity (n=3)	0.677	0.422	0.439	0.396	0.290	0.045	0.087	0.076

5 EXPERIMENTS

We have implemented LearnShapley in Python 3.7 using PyTorch [35] and the Transformers library [46]. All experiment were preformed on a Linux Ubuntu 18.04.5 LTS machine with 512G RAM, AMD EPYC 7302 16-Core Processor and two GeForce RTX 309 GPUs. The source code of our implementation is available in [4].

5.1 Compared Methods

The exact algorithm for Shapley value computation from [15] is used *in an offline manner* to compute DBShap, from which distinct subsets are used as training and test data for LearnShapley. This in particular means that we compare the ranking produced by LearnShapley to the gold ranking yielded by [15]. Note that in contrast to the alternative of using [15] for deployment, LearnShapley does not require detailed provenance for inference but rather only lineage. We also compare the performance of our system to another possible approach that learns from a query log but relies only on the lineage at inference time, namely that of *Nearest Queries*. Nearest Queries is based on the aggregation of Shapley values of contributing facts of the n nearest queries to the query in question, where the similarity metric and n, number of neighbors to consider are configurable. We implemented the Nearest Queries models in Python 3.7. We present variations of this model using syntax-based, witness based and rank-based similarities, using $n = 3$, which led to the best results. Note that Nearest Queries with rank-based similarity is only feasible in a controlled experiment, since computing rank based similarity requires exact Shapley values of all contributing facts.

5.2 Evaluation Metrics

We evaluate the predicted rank over the set of test queries over each database in DBShap. Given a database $\mathcal{D}$, query q, an output tuple t, and a set of facts Lineage$(\mathcal{D}, q, t)$, we wish to rank the fact according to their contribution. To this end, we first obtain the predicted Shapley value of every fact $f \in$ Lineage$(\mathcal{D}, q, t)$. We then rank the facts according to the predicted Shapley values in order to produce the predicted rank. We evaluate the predicted rank with respect to the gold rank using two metrics:

- **Normalized Discounted Cumulative Gain (NDCG)** is a measure of ranking quality [45], calculated as the ratio between the Discounted Cumulative Gain (DCG) and the Ideal Discounted Cumulative Gain (IDCG). The DCG is a sum of the scores, or gains, of ranked elements based on its position in the result. The IDCG is the highest DCG out of all possible rankings, used to normalize the DCG score between 0 and 1. We measure NDCG for the top 10 contributing facts, denoted as NDCG@10.
- **Precision at k (p@k)** measures how many facts from the top k are in the model's prediction. If there are ties, we take the first k facts plus all subsequent facts that hold the same rank as the k'th element. We measure p@k for k=1,3,5, denoted as p@1, p@3 and p@5.

5.3 Main Experimental Results

Table 3 shows the NDCG@10, p@1, p@3 and p@5 scores of LearnShapley over both databases in DBShap. We trained two variations of LearnShapley, LearnShapley-base and LearnShapley-large which utilizes BERT-base and BERT-large, respectively. After the pre-training stage, we chose the pre-training checkpoint with the lowest MSE score on the dev set. After fine-tuning, we chose the checkpoint with the highest NDCG@10 score over the dev set. Both versions of LearnShapley significantly outperforms all nearest queries models for both IMDB and Academic, and achieves scores close to 1 in all metrics. In terms of NDCG@10, LearnShapley-base demonstrates 0.7 (Academic) and 0.3 (IMDB) points improvement compared to baseline models. In comparison, previous work which uses detailed boolean provenance at deployment to rank database facts which was tested on a slightly different set of queries over IMDB, was able to achieve near-perfect NDCG and p@5 scores [15]. While our results are slightly lower, they are achieved when given only the lineage at deployment time rather than the boolean provenance. On the Academic database, LearnShapley-base achieves 0.8-0.9 points improvement on p@1, p@3 and p@5, respectively. On IMDB, we observe slightly smaller improvements: 0.4-0.5 points improvements in p@1, p@3 and p@5 scores, since the performance of baseline methods is much higher compared to Academic. We conjecture that this is caused by different characteristics of the query log used for training: Nearest Queries models rely more heavily on the query log having very similar queries to the query of interest at inference time, while LearnShapley's architecture allows it to generalize better. These results demonstrate that LearnShapley better captures the importance and contribution of facts and better generalizes to unseen query-output tuple pairs. LearnShapley-large achieves even better NDCG@10 scores with additional 0.004 points in both Academic and IMDB, and 0.003-0.007 additional p@k points compared to LearnShapley-base.

5.4 Analysis

Performance as a Function of Lineage Size: Figure 9a shows the relationship between the size of the lineage of queries and output tuples from the Academic database and the performance of LearnShapley-base in terms of NDCG@10. Each data point represents a query and output tuple, where output tuples of the same query have the same color. The dotted black line is the linear trendline. LearnShapley shows an improvement in NDCG@10 scores as the number of facts in the lineage decreases. Nevertheless, LearnShapley achieves high NDCG@10 scores even for queries and output tuples with the largest size lineage in the Academic test set in DBShap, with 165 contributing facts.

Performance as a Function of Query Similarity: The pre-training objective aims to help the model identify similar queries and learn signals from these queries. Thus, we analyze the similarity scores of nearest queries vs. the NDCG@10 score for query and output tuple pair. Each data point in Figure 10 represents a query and output tuple pair, (q, t). The x axis is the similarity score of q and (top) the single nearest query or (bottom) the average of 5 nearest queries with respect to syntax-based, witness-based and rank-based similarities. The y axis is the NDCG@10 score of the ranking of contributing facts with respect to (q, t) obtained by running LearnShapley-base. When taking the average of 5 nearest queries we see a positive correlation with respect to all similarity metrics, which is not the case when comparing to the single most similar query. This is an indication of the model's ability to generalize and aggregate signals from several queries in the training data in order to make predictions at test time.

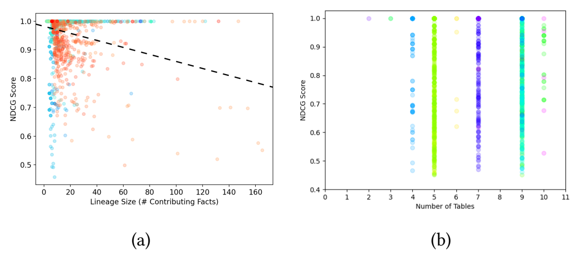
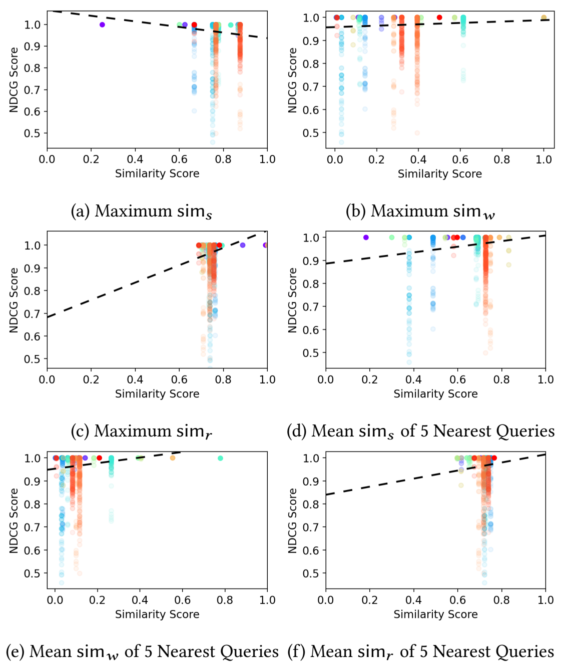


Figure 9: Performance (NDCG@10) of LearnShapley on query-output tuple pairs from the Academic database, as a function of (a) the number of contributing fact in the lineage and (b) the number of tables joined by the query. Each point represents a query and output tuple, where output tuples of the same query have the same color. Darker colors represent multiple points in the same location.

Performance as a Function of the Query Complexity: Figure 9b shows the model's performance for varying number of tables joined in the query, which is commonly used as a measure of the query complexity. Interestingly, we have observed no significant correlation between the query complexity and the model performance. Also, we observe that the model achieves very high scores even for some very complex queries joining 9 and 10 tables.

5.5 Ablation Study

Language Models: A key component of LearnShapley is the use of a language model in order to learn complex signals of queries and databases semantics. A natural question is whether the use of BERT is necessary, or maybe smaller models will still be able to learn signals and apply them to new data. Table 3 shows the performance of a transformer encoder with 3 layers and 8 attention heads compared to the performance of LearnShapley. The transformer encoder was randomly initialized and then trained for 50 epochs using the same data used during the fine-tuning stage of LearnShapley. As in LearnShapley, we chose the checkpoint with the highest NDCG@10 score on the dev set. On Academic, the transformer model shows a drop of 0.2 in NDCG@10 compared to LearnShapley-base, and 0.01, 0.032 and 0.003 points drop in p@1, p@3 and p@5. On IMDB, we observe



(a) Maximum sim_s (b) Maximum sim_w

(c) Maximum sim_r (d) Mean sim_s of 5 Nearest Queries

(e) Mean sim_w of 5 Nearest Queries (f) Mean sim_r of 5 Nearest Queries

Figure 10: NDCG@10 of LearnShapley on query and output tuple pairs (q, t) with respect to maximum similarity score to q in DBShap (top) and the average similarity score of 5 most similar queries to q in DBShap (bottom), using syntax-based ((a), (d)), witness-based ((b), (e))) and rank-based ((c), (f)) similarity metrics. Each point represents a query and output tuple, where output tuples of the same query have the same color. Darker colors represent multiple points in the same location.

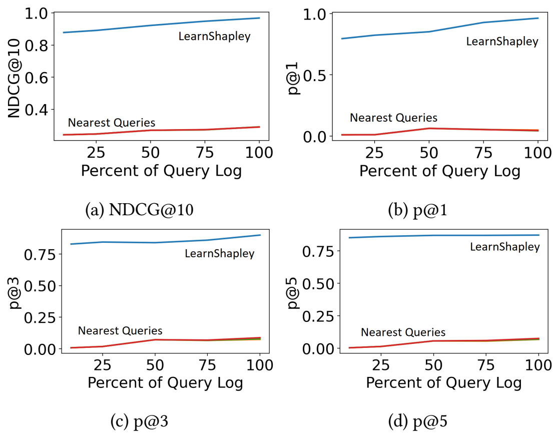
a slightly smaller drop of 0.02 points in NDCG@10, and 0.079, 0.046 and 0.023 points drop in p@1, p@3 and p@5, respectively.

Pre-Training: Another natural question is whether or not the additional pre-training stage indeed helps LearnShapley to learn relevant signals. We test this property by directly fine-tuning BERT on the same data as in the fine-tuning stage of LearnShapley, and comparing the performance of both models. We fine-tune BERT-base for 10 epochs and choose the checkpoint with the highest NDCG@10 score on the dev set. Results are shown in Table 3, under Ablations. We can see that for both databases LearnShapley-base obtains higher scores across all metrics. Pre-training improves the model performance by 0.016 (Academic) and 0.004 (IMDB) points in terms of NDCG@10, and adds 0.001-0.037 points of p@k scores to the base version of the model.

Similarity Metrics: The similarity metrics used during pre-training represent different characteristics of query similarity. We train variations of LearnShapley-base on the Academic dataset, pre-trained on different subsets of similarity metrics, in order to study the effect of each metric combination on the model's performance. Each model was pre-trained for 20 epochs, and fine-tuned for 10 epochs on the same data as LearnShapley. As in LearnShapley, after pre-training we chose the checkpoint with the lowest MSE score on the dev set. For fine-tuning, Ablation results are summarized in Table 4. While all metrics provide the model with helpful information, some metrics contribute more than others. As expected, the highest performance using all evaluation metrics is obtained when LearnShapley is pre-trained on all similarity metrics, i.e. rank-based, syntax-based and witness-based. When pre-training on two of these similarity metrics we observe a slight drop in performance - an average of 0.05 drop in NDCG@10 and 0.01 drop in p@1. Removing the witness-based similarity metric has the most effect in terms of NDCG@10, resulting in a 0.07 drop. This is demonstrated again in the case of single similarity metrics - a model trained on witness similarity alone achieve the highest scores out of all single-metric models when measuring both NDCG@10 and p@k. This is somewhat surprising, since witness similarity provides the most sparse signal to the model out of all metrics, meaning most of the witness similarity scores are 0. This demonstrates the precision/recall trade-off of witness similarity. In the few cases where witness similarity is high, it is a strong indication that the semantics of the queries are similar and the queries likely share computational information.

Table 4: Performance of LearnShapley-base with different combinations of similarity metrics used for pre-training.

	NDCG@10	p@1	p@3	p@5
LearnShapley-base	**0.968**	**0.958**	**0.900**	**0.873**
witness & rank (w/o syntax)	0.960	0.945	0.889	0.861
syntax & rank (w/o witness)	0.959	0.957	0.881	0.867
witness & syntax (w/o rank)	0.963	0.939	0.900	0.854
syntax (w/o witness & rank)	0.934	0.873	0.865	0.861
witness (w/o syntax & rank)	0.939	0.870	0.871	0.868
rank (w/o witness & syntax)	0.932	0.856	0.896	0.860



(a) NDCG@10 (b) p@1

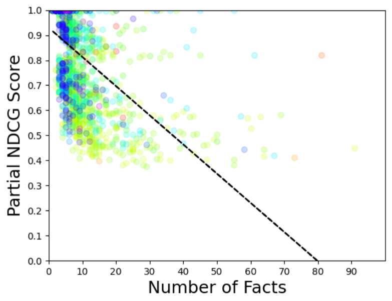
(c) p@3 (d) p@5

Figure 11: Performance (NDCG@10, p@1, p@3 and p@5) as a function of the training set size on the Academic dataset. Training set size controls the amount of over-all data seen during training, and particularly the percent of new facts, previously unseen by the model, at test time. LearnShapley is in blue (top), and Nearest Queries models are in red, yellow and green (overlapping, bottom).

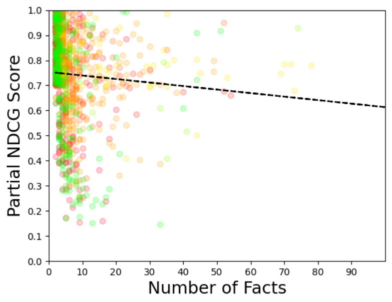
Tradeoffs. Introducing the fine-tuning phase incurred additional 25% time at training time (different fine-tuning objectives were similar in terms of their overhead on their training time); as shown above, it yields in return a non-negligible gains in terms of quality.

5.6 Varying Query Log Sizes

The performance of LearnShapley heavily relies on the number of queries in the log and their density in terms of query similarity. We study the effect of the query log size by training variations of LearnShapley and Nearest Queries models on 10%, 25%, 50% and 75% of the original query log in DBShap. Each subset was randomly chosen, and contains all smaller subsets (i.e. the 25% experiment includes all queries used in the 10% experiment). Other than the difference in training data, the training of LearnShapley and Nearest Queries models is the same as in 5.3. Figure 11 summarizes the results. LearnShapley trained on only 10% and 25% percent of the query log still outperforms baseline methods that were trained on 100% of the queries. As expected, training on smaller subsets of the query log results in lower performance across all metrics and models examined, however LearnShapley is proving to be more robust and shows a smaller drop in performance compared to the naive Nearest Queries models.



(a) Facts Seen During Training



(b) Facts Unseen During Training

Figure 12: NDCG of partial rankings on the Academic test set: for each query and output tuple, we measure separately the NDCG of facts seen during training, and new facts, unseen during training.

5.7 Ranking Unseen Facts

The input to LearnShapley includes only queries, their output and lineage rather than the entire database. This in particular means that at training time, LearnShapley only "sees" facts participating in the lineage of query results in the training set. Our experiments demonstrates the performance of LearnShapley in such scenarios. In the test set of DBShap, 37.75% of the facts that appear in the lineage of queries are new to the model in the sense that they did not appear in the lineage of queries in the training set. Furthermore, the experiments described in Section 5.6 demonstrates the performance of LearnShapley with increasing percentages of new, unseen facts. There, 75,50,25% log sizes correspond to 40.1%, 45% and 69% (respectively) of the facts in the test set were new to the model.

We analyze the performance of LearnShapley on seen and unseen facts from the Academic test set. Figure 12 demonstrates the partial NDCG scores of the rankings generated by LearnShapley, limited to the subset of either seen or unseen facts. As expected, LearnShapley achieves higher partial NDCG score on the subsets of facts that are seen by the model during training (10% improvement on average). While the overall scores of unseen facts are lower, Figure 12b shows that in many cases, LearnShapley is still able to preform well. Note that the NDCG scores in Figure 12b are evaluated on subsets of fact, thus the reported partial NDCG score are not comparable to the NDCG score of the complete set of facts, namely those shown in Figure 9. Table 5 shows an example of a test query, whose lineage contains new facts. LearnShapley is able to generalize and produces a correct ranking of the previously unseen fact, marked in bold. Note, in contrast, that the Nearest Queries approach predicts the score for each fact to be its average score for other (nearest) queries in the training set. This score is naturally 0 for every unseen fact. The baseline approach thus utilizes no signal on unseen facts, simply placing them below all seen facts and in arbitrary order.

Table 5: Prediction of LearnShapley for a query (top) and output tuple (first tuple in the table) with facts in the lineage unseen by the model during training (ranked 5, in bold). LearnShapley successfully predicts the rank of the previous unseen fact.

```
SELECT author.name
FROM author, organization, writes, publication, conference
WHERE author.oid = organization.oid AND
author.aid = writes.aid AND writes.pid = publication.pid AND
publication.cid = conference.cid AND
organization.name = 'University of Michigan' AND
publication.year >2010
GROUP BY author.name
```

Predicted Rank	True Rank	Tuple Value
1	1	['Michael G. Kallitsis', ...]
4	2	['University of Michigan', ...]
3	3	['Network Decomposition...]
2	4	['IEEE SmartGridComm', ...]
5	**5**	**['IEEE GLOBECOM', ...]**
6	6	['A decentralized algorithm for optimal...', ...]

5.8 Inference Times

We compare the inference time of LearnShapley to Nearest Queries models with syntax-based and witness-based similarities. Table 6 summarizes the maximum and average inference times for query and output tuple pairs from DBShap. The time complexity of k-NN algorithm is $O(nd)$ where d is the time complexity of the distance function. In our implementation, the distance between a pair of queries q_1 and q_2 is equal to one minus their similarity score: $d(q_1, q_2) = 1 - sim(q_1, q_2)$. Thus, the time complexity and subsequent inference time of Nearest Queries model depends on the used similarity metric and the size of the query log. We see this reflected in the inference times in Table 6: witness-based similarity requires only set operations on existing data from DBShap, i.e. output tuple ids, while syntax-based similarity requires additional pre-processing in the form of decomposing the query

Table 6: Mean and Maximum inference times in seconds for query and output tuple pairs from DBShap. Inference time evaluated on Nearest Queries models with syntax-based similarity and witness-based similarities, and both sizes of LearnShapley.

	Academic		IMDB	
	Mean	Max	Mean	Max
Exact computation	2.491	469.885	3.750	1404.49
Nearest Queries (Syntax)	0.094	5.516	0.227	31.167
Nearest Queries (Witness)	0.013	0.043	0.090	0.335
LearnShapley-base	0.0786	1.070	0.169	1.393
LearnShapley-large	0.173	2.333	0.363	2.954

into the different query operations. At inference time, LearnShapley requires forward passes through the model whose execution time only depends on the model size. Thus, the inference time of LearnShapley-base is slightly longer than witness-based Nearest Queries, but 5× less than that of syntax-based Nearest Queries on average. While LearnShapley-large has the longest inference times on average, syntax-based Nearest Queries has the longest times in the worst case, with 30× and 10× longer inference times compared to LearnShapley-base and LearnShapley-large. We also compare the execution time of the state-of-the-art baseline for exact computation [15] to the execution time of LearnShapley at deployment. Observe that LearnShapley is much faster at deployment than exact Shapley value computation. In addition, the Shapley computation algorithm in [15] works on detailed why-provenance whereas LearnShapley only needs the lineage at deployment. Note that the results are aggregated (average/max) over pairs of a query and an output tuple; the overall effect of the speed-up exacerbates (in absolute terms of the overall execution time) if we repeat the process in order to explain all output tuples of a given query.

The effect of ablations. Most of the different ablations had no effect on the inference time since they did not affect the architecture. One ablation that did make a difference is the transformer encoder architecture, where we used a randomly initialized transformer encoder that is smaller in size, compared to BERT. This variation achieved lower quality scores across all of our metrics, and thus, for brevity, we do not include its execution time.

6 RELATED WORK

Query answering explanations and Shapley values: There are two main types of models that exist for explaining database query results. The first are models that provide information about the provenance of output tuples, such as the facts involved in their calculation, and potentially the relationships between them, at different levels of detail (e.g. [8, 9, 11, 14, 20]). A major challenge in this area has been the complexity of provenance expressions, which makes it difficult to efficiently compute and store. As a result, the fields of provenance factorization and summarization have been studied as ways to compactly represent provenance (e.g. [7, 10, 25, 34, 36]). The second type are models that quantify the contributions of facts (e.g. [19, 29, 32, 33, 40]). This latter approach is the one that we follow in this paper.

Specifically, the works of [15, 30, 37] have studied in depth the use of Shapley values to quantify facts contribution in query answering. In [30, 37] the authors have laid the foundations for studying the problem, but had not focused on practically efficient algorithms for computing Shapley values; in fact, they have shown that exact computation is intractable for a large class of queries (namely, non-hierarchical ones). In [15], the authors introduce an exact method for computing Shapley values and a faster, but less reliable, inexact method for ranking facts based on Shapley values. Both methods utilize *boolean provenance* for relevant tuples. The exact method uses knowledge compilation [12], and is only applicable in cases where the boolean provenance can be compiled into a deterministic and decomposable circuit of a reasonable size. The inexact method, called CNF Proxy, starts from a non-factorized DNF representation of the provenance and applies Tseytin transformation [44] to obtain a CNF. Such non-factorized representations may not be readily available, and may be much larger than the lineage due to repetition of variables across clauses. In our work, we used the exact computation method proposed in [15] to calculate DBShap offline, which we then use as training data. However, during inference, our model relies solely on lineage, not the full provenance. Its fast inference time renders it suitable for real-time analysis.

Language models: The use of pre-trained language models has gained significant attention in the field of natural language processing [18, 27]. These models have also been applied to the field of databases, with research proving their utility in tasks such as table understanding [13, 22, 47], column understanding [43], and text-to-SQL [28, 48].

7 CONCLUSION

In this work, we preformed extensive experiments and analysis of LearnShapley, a novel machine learning approach designed to rank facts based on their contribution to query results. To this end we introduced DBShap, a dataset for the learning task of ranking fact contribution in query answering.

Our analysis reveals that LearnShapley consistently achieves high performance, with NDCG@10, p@1, p@3, and p@5 scores exceeding 0.9 for all metrics. These scores reflect the high fidelity of the predicted rankings compared to the true ranking obtained via the actual Shapley values of the tuples with respect to the new query. We further analyze the performance of LearnShapley and show that it generalizes and achieves high NDCG@10 scores for even the queries with the largest lineage size in DBShap. Future work in this direction can focus on improving LearnShapley's ability to generalize to unseen facts and tables, and develop new methods can be applied to previously unseen databases.

Additionally, we preformed an extensive ablation study on the effect of each component of LearnShapley. We find that the proposed pre-training objectives along with the task-specific fine-tuning, all contribute to the model's high performance.

In this work, we focus on a setting where the lineage (but not the detailed provenance) is given. Future work can utilize DBShap to design models for different tasks, such as predicting the lineage of a given query and result, or obtaining the ranking of contributing facts without assuming access to the lineage.

Acknowledgments. This research has been partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 804302).

REFERENCES

[1] Bahareh Arab, Dieter Gawlick, Vasudha Krishnaswamy, Venkatesh Radhakrishnan, and Boris Glavic. 2018. Using Reenactment to Retroactively Capture Provenance for Transactions. *IEEE Transactions on Knowledge and Data Engineering* 30, 3 (2018), 599–612. https://doi.org/10.1109/TKDE.2017.2769056

[2] Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. 2018. GProM-a swiss army knife for your provenance needs. *A Quarterly bulletin of the Computer Society of the IEEE Technical Committee on Data Engineering* 41, 1 (2018).

[3] Dana Arad, Daniel Deutch, and Nave Frost. 2022. DBShap: A dataset of queries, query results, fact contributions and Shapley values over the IMDB and Academic databases. https://www.cs.tau.ac.il/~danielde/dbshap_page/dbshap.html.

[4] Dana Arad, Daniel Deutch, and Nave Frost. 2022. LearnShapley: A system for ranking facts contributions based on query log. https://github.com/danaarad/LearnShapley.

[5] Dana Arad, Daniel Deutch, and Nave Frost. 2022. LearnShapley: Learning to Predict Rankings of Facts Contribution Based on Query Logs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management.* 4788–4792.

[6] Natalia Arzamasova, Klemens Böhm, Bertrand Goldman, Christian Saaler, and Martin Schäler. 2019. On the usefulness of SQL-query-similarity measures to find user interests. *IEEE Transactions on Knowledge and Data Engineering* 32, 10 (2019), 1982–1999.

[7] Nurzhan Bakibayev, Dan Olteanu, and Jakub Závodnỳ. 2012. FDB: A query engine for factorised relational databases. *arXiv preprint arXiv:1203.2672* (2012).

[8] Peter Buneman, James Cheney, Wang-Chiew Tan, and Stijn Vansummeren. 2008. Curated databases. In *Proceedings of PODS.* 1–12. https://homepages.inf.ed.ac.uk/opb/papers/inv.pdf

[9] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *ICDT*. Springer, 316–330. https://repository.upenn.edu/cgi/viewcontent.cgi?article=1209&context=cis_papers

[10] Adriane P Chapman, Hosagrahar V Jagadish, and Prakash Ramanan. 2008. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* 993–1006.

[11] Yingwei Cui, Jennifer Widom, and Janet L Wiener. 2000. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)* 25, 2 (2000), 179–227. http://ilpubs.stanford.edu:8090/252/1/1997-3.pdf

[12] Adnan Darwiche and Pierre Marquis. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17 (2002), 229–264. https://arxiv.org/abs/1106.1819

[13] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record* 51, 1 (2022), 33–40.

[14] Daniel Deutch, Nave Frost, and Amir Gilad. 2020. Explaining natural language query results. *The VLDB Journal* 29, 1 (2020), 485–508.

[15] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. 2022. Computing the Shapley Value of Facts in Query Answering. In *Proceedings of the 2022 International Conference on Management of Data.* 1570–1583.

[16] Daniel Deutch, Amir Gilad, and Yuval Moskovitch. 2015. Selective provenance for datalog programs using top-k queries. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1394–1405.

[17] Daniel Deutch, Yuval Moskovitch, and Noam Rinetzky. 2019. Hypothetical Reasoning via Provenance Abstraction. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.* ACM, 537–554. https://doi.org/10.1145/3299869.3300084

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[19] Dennis Dosso, Susan B Davidson, and Gianmaria Silvello. 2022. Credit distribution in relational scientific databases. *Information Systems* 109 (2022), 102060.

[20] Todd J Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of PODS.* 31–40. https://repository.upenn.edu/cgi/viewcontent.cgi?article=1022&context=db_research

[21] Todd J Green and Val Tannen. 2017. The semiring framework for database provenance. In *Proceedings of PODS.* 93–99. https://dl.acm.org/doi/10.1145/3034786.3056125

[22] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349* (2020).

[23] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[24] Gokhan Kul, Duc Thanh Anh Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2018. Similarity metrics for SQL query clustering. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2408–2420.

[25] Seokki Lee, Bertram Ludäscher, and Boris Glavic. 2020. Approximate summaries for why and why-not provenance (extended version). *arXiv preprint arXiv:2002.00084* (2020).

[26] Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. 2020. Pre-training via paraphrasing. *Advances in Neural Information Processing Systems* 33 (2020), 18470–18481.

[27] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[28] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627* (2020).

[29] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2020. The Shapley value of tuples in query answering. In *ICDT*, Vol. 155. Schloss Dagstuhl, 20:1–20:19. https://arxiv.org/abs/1904.08679

[30] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2020. The Shapley value of tuples in query answering. In *ICDT*, Vol. 155. Schloss Dagstuhl, 20:1–20:19. https://arxiv.org/abs/1904.08679

[31] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. The complexity of causality and responsibility for query answers and non-answers. *PVLDB* 4, 1 (2010), 34–45. https://www.vldb.org/pvldb/vol4/p34-meliou.pdf

[32] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F Moore, and Dan Suciu. 2010. The complexity of causality and responsibility for query answers and non-answers. *Proceedings of the VLDB Endowment* (2010).

[33] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and explanations in databases. *Proceedings of the VLDB Endowment (PVLDB)* 7, 13 (2014), 1715–1716. http://www.vldb.org/pvldb/vol7/p1715-meliou.pdf

[34] Dan Olteanu and Jakub Závodnỳ. 2012. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory.* 285–298.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[36] Christopher Ré and Dan Suciu. 2008. Approximate lineage for probabilistic databases. *Proceedings of the VLDB Endowment* 1, 1 (2008), 797–808.

[37] Alon Reshef, Benny Kimelfeld, and Ester Livshits. 2020. The impact of negation on the complexity of the Shapley value in conjunctive queries. In *Proceedings of PODS.* 285–297. https://arxiv.org/abs/1912.12610

[38] Alvin E Roth. 1988. *The Shapley value: essays in honor of Lloyd S. Shapley.* Cambridge University Press.

[39] Sudeepa Roy, Laurel J. Orr, and Dan Suciu. 2015. Explaining query answers with explanation-ready databases. *Proceedings of the VLDB Endowment (PVLDB)* 9, 4 (2015), 348–359. http://www.vldb.org/pvldb/vol9/p348-roy.pdf

[40] Babak Salimi, Leopoldo E. Bertossi, Dan Suciu, and Guy Van den Broeck. 2016. Quantifying causal effects on query answering in databases. In *TaPP*. USENIX Association. http://web.cs.ucla.edu/~guyvdb/papers/SalimiTaPP16.pdf

[41] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and probability management in postgresql. *Proceedings of the VLDB Endowment (PVLDB)* 11, 12 (2018), 2034–2037. https://hal.inria.fr/hal-01851538/file/p976-senellart.pdf

[42] Lloyd S Shapley. 1953. A value for n-person games. *Contributions to the Theory of Games* 2, 28 (1953), 307–317. http://www.library.fa.ru/files/Roth2.pdf#page=39

[43] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data.* 1493–1503.

[44] Grigori S Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of reasoning.* Springer, 466–483. https://link.springer.com/chapter/10.1007/978-3-642-81955-1_28

[45] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. 2013. A theoretical analysis of NDCG ranking measures. In *Proceedings of COLT*, Vol. 8. 6. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.490&rep=rep1&type=pdf

[46] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[47] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314* (2020).

[48] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. GraPPa: grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845* (2020).