# Ranking Desired Tuples by Database Exploration

Xuedi Qin† Chengliang Chai† Yuyu Luo† Tianyu Zhao†
Nan Tang‡ Guoliang Li† Jianhua Feng† Xiang Yu† Mourad Ouzzani‡
†Department of Computer Science, Tsinghua University ‡Qatar Computing Research Institute, HBKU
{qxd17@mails., chaicl15@mails., luoyy18@mails., zhaoty17@mails., liguoliang@, fengjh@,
x-yu17@mails.}tsinghua.edu.cn, {ntang, mouzzani}@hbku.edu.qa

***Abstract*—Database exploration – the problem of finding and ranking desired tuples – is important for data discovery and analysis. Precisely specifying SQL queries is not always practical, such as "*finding and ranking off-road cars based on a combination of Price, Make, Model, Age, Mileage, etc*" – not only due to the query complexity (*e.g.,* with may have many *if-then-else*, *and*, *or* and *not* logic), but also because the user typically does not have the knowledge of all data instances (and their variants).**

**We propose DEXPLORER, a system for interactive database exploration. From the user perspective, we propose a simple and user-friendly interface, which allows to: (1) confirm whether a tuple is desired or not, and (2) decide whether a tuple is more preferred than another. Behind the scenes, we jointly use multiple ML models to learn from the above two types of user feedback. Moreover, in order to effectively involve *human-in-the-loop*, we need to select a set of tuples for each user interaction so as to solicit feedback. Therefore, we devise *question selection* algorithms, which consider not only the estimated benefit of each tuple, but also the possible partial orders between any two suggested tuples. Experiments on real-world datasets show that DEXPLORER outperforms existing approaches in effectiveness.**

## I. INTRODUCTION

We study the problem of *interactive database exploration*, for the scenarios that a user needs to find desired and ranked tuples, but the query intent is hard to precisely specify. This is common in practice, for both non-experts who cannot write SQL queries and scripts, and experts who are not familiar with the data. Adding to the complexity is that the query intent may be complicated, *e.g.,* finding tuples that satisfy a combination of many *if-then-else*, *and*, *or* and *not* conditions, and are ranked by a weighted function over multiple attributes.

**Existing Works.** We categorize existing works as follows.

*Keyword search* [17], [23] retrieves *some* desired tuples by providing keywords. However, keyword queries are typically under-specified and thus ambiguous. Even with iterative refinement, it is almost impossible to formulate a keyword query that can be unambiguously translated to a (complicated) query.

*Query-by-example* either infers an SQL query $Q$ [28], [9] or learns a ML model $M$ [19] over a database $D$ using user provided examples $E$. Based on $E$, they try to infer $Q$ or $M$ such that $Q(D)$ or $M(D)$ is good *w.r.t.* $E$, where the quantification of "goodness" varies in different applications.

*Preference between tuples* [26], [32] ranks tuples by inferring from user provided partial orders for tuple pairs, where [32] is to find the top-1 result, and [26] is to rank all tuples.

**Limitations of Existing Works.** Let *DE* denote the general problem that discovers and ranks tuples, *DE-Decision* a special case of DE where users only want desired tuples (without ranking), and *DE-Ranking* another special case where users want the ranking over all tuples. Table I compares different methods for data exploration.

(I) [*Supported Problems*.] Almost all existing approaches focus on *either* DE-Decision *or* DE-Ranking. Besides DEXPLORER, only SQLSynthesizer [33] can discover an SQL query with ranking, but it can only support simple (hierarchical) ranking function such as ORDER BY attribute *year* first and then ORDER by attribute *kilometer*. No existing approach can support a combination of selection conditions and a more natural (but maybe more complicated) ranking function such as $(-0.018 \times price + 0.982 \times powerPS)$.

(II) [*Supported Special Cases*.] Even for approaches that support either DE-Decision or DE-Ranking, some existing methods can only handle some special cases, as described in the column "Supported Special Cases" in Table I.

(III) [*Interactive vs. Static*.] Some approaches are interactive with the users while others support static input, as shown in the column "Interactive" in Table I. Note that some back-end inference algorithms can easily support user interactions (such as ML models), while others might be hard (such as keyword-based or SQL query inference).

The main difference between this work and those existing works is how to *holistically* solve the general DE problem, instead of treating DE-Decision and DE-Ranking separately, by minimizing the interactions with users.

**Our Methodology.** We propose DEXPLORER for with the following main features. [***Interactivity***.] DEXPLORER is designed as an interactive system, because providing a set of representative, unbiased, and sufficient samples in one shot is hard. [***Usability***.] It offers an easy-to-use interface for any user, which gives a list of tuples and allows two simple operations "click" and "drag" for the user to provide *true/false* tuple labels and partial orders between tuples. [***Capability***.] In order to infer (possibly) complicated query intent, we propose to jointly train several ML models instead of guessing SQL queries, along the same line of [19], because tightly specifying conditions in SQL queries in database exploration might be hard. More specifically, we use a random forests [22] to infer desired tuples, and a hybrid ranking model combining LambdaMART [31] and Ranking SVM to rank tuples. [***Effectiveness***.] We devise novel question selection algorithms that jointly estimate the benefit of soliciting feedback from both

| Project | Input | DE-Decision | DE-Ranking | DE | Supported Special Cases | Interactive |
|---|---|---|---|---|---|---|
| DISCOVER [18] | (1) | ✓ | | | only find a small number of interesting tuples | |
| SQLSynthesizer [33] | (2) | | | ✓ | only simple ranking function | |
| SQUID [9] | (2) | ✓ | | | does not support the "or" predicate | |
| AIDE [6] | (2) | ✓ | | | | ✓ |
| Huang et al. [19] | (2) | ✓ | | | does not support the "or" predicate | ✓ |
| Chaudhuri et al. [2] | (3) | | ✓ | | only rank categorical attributes | |
| Qian et al. [26] | (3) | | ✓ | | | ✓ |
| Xie et al. [32] | (3) | | ✓ | | only find top-1 result | ✓ |
| DEXPLORER | (1,2,3) | ✓ | ✓ | ✓ | | ✓ |

TABLE I
COMPARISON WITH STATE-OF-THE-ART (USER INPUT 1: KEYWORD, 2: EXAMPLES, 3: PARTIAL ORDERS)

tuple labeling and partial order labeling.

**Contributions.** Our contributions are summarized below.
(1) We formally define the problem of interactive database exploration. (Section II)

(2) We presents DEXPLORER that iteratively trains ML models and on-the-fly predicts result for interactive data exploration with the users. (Section III)

(3) We present a hybrid decision and ranking model for question selection to minimize human cost. (Section IV)

(4) We describe methods to handle two special cases: the user wants only desired tuples, or only ranked tuples. (Section V)

(5) We conduct extensive experiments to show the effectiveness of DEXPLORER. (Section VI)

## II. PROBLEM STATEMENT

Informally speaking, given a relational table $T$, the user wants to find a subset $R \subseteq T$, and tuples in $R$ are ranked.

In practice, a user's query intent might be formulated as either an SQL query, machine learning (ML) models, or some specific logic. Generally speaking, we want to infer/learn a method/model $f()$ as the proxy of a user's query intent, where $f(T)$ returns the ranked list $R'$ that is close to $R$.

***Example 1:*** Suppose a user wants a new manual petrol car that is produced after year 2010, not provided by commercial sellers, and its brand can be either BMW with price $\leq 10000$, or Volkswagen with price $\leq 8000$. Moreover, assume that she wants all cars to be ranked, *e.g.,* using a weighted sum function: $-0.018 \times price + 0.982 \times powerPS$. That is, the query intent could be expressed as $Q_1$ below: □

```
SELECT * FROM Car
WHERE seller != "commercial" AND year ≥ 2010
AND gearbox="manually" AND fuelType="petrol"
AND ((brand = "bmw" AND price ≤ 10000) OR
(brand = "volkswagen" AND price ≤ 8000))
ORDER BY -0.018 * price + 0.982 * powerPS DESC;
```

$Q_1$

Example 1 shows a user's query intent may be: (1) ***hard to specify:*** there might have complicated predicates in the WHERE clause and it is almost impossible to manually specify the weighted sum function with correct parameters in the ORDER BY clause, and (2) ***hard to infer:*** no existing work (see Table I) can effectively infer such a query, not to say more complicated cases.

**User Operations.** We allow two *user operations*:

(1) *true/false* labeling: the user may label a given tuple as *true*(desired) or *false*(not desired); and

(2) a partial order: given two tuples $t_i$ and $t_j$, the user might tell which tuple is more preferable.

**User Questions.** A *question* $\mathbb{I}$ is a list of $k$ tuples, which solicits two types of labels from a user:
(1) *decision:* the user will split $\mathbb{I}$ into three disjoint sets of tuples: $\mathbb{D}^+$ with *true* labels, $\mathbb{D}^-$ with *false* labels, and $\mathbb{D}^?$ meaning unknown. Let $\mathbb{D}$ be the set of all tuple labels, *i.e.,* $\mathbb{D} = \mathbb{D}^+ \cup \mathbb{D}^- \cup \mathbb{D}^?$.
(2) *ranking:* rank (partial) pairs of tuples in $\mathbb{D}^+$, which gets a set $\mathbb{R}$ of partial orders. Implicitly, any tuple $t_i \in \mathbb{D}^+$ is more preferred than any tuple $t_j \in \mathbb{D}^-$, and no need to rank two tuples in $\mathbb{D}^-$, denoted by $t_x \equiv t_y$ for any $t_x, t_y \in \mathbb{D}^-$.

***Example 2:*** The table in Figure 1 shows the labeling examples based on $Q_1$ in Example 1. (1) The user can *annotate* the tuples she desires or not, *e.g.,* $\mathbb{D}^+ = \{t_1, t_2, t_3, t_4\}$ and $\mathbb{D}^- = \{t_5, t_6\}$. (2) The user can specify that which cars are more preferred for tuples in $\mathbb{D}^+$, either through a total order such as $(t_1 \succ t_2 \succ t_3 \succ t_4)$ or a set of partially ordered lists such as $(t_1 \succ t_2 \succ t_3)$ and $(t_1 \succ t_4)$. □

**Problem Statement.** Given a table $T$, a parameter $k$ (*i.e.,* the number of tuples to show in each question), and a budget $n$ for the number of questions, the problem of *interactive database exploration* (**IDE**) is to iteratively ask $n$ questions $\{\mathbb{I}_1, \ldots, \mathbb{I}_n\}$ ($|\mathbb{I}_i| = k$ for $i \in [1, n]$) to the user, and infer a ranked set $R$ of tuples based on the user feedback.

**Two Research Challenges.** In order to solve IDE, there are two main research challenges: (1) *Answer Inference*: *how to infer $R$ based on the user feedback for $\mathbb{I}_1, ..., \mathbb{I}_n$?* (Section III) (2) *Question Selection*: *how to select a question $\mathbb{I}_i$ in the $i$-th iteration?* (Section IV)

**Two Special Cases.** (1) *IDE-Decision*: the user has no preference between desired tuples; and (2) *IDE-Ranking*: the user only wants to rank all tuples.

**Supporting Multiple Tables.** Interactively exploring multiple tables is supported by the following two steps. (1) *Attribute enrichment:* it is to enrich the entity table to be explored by adding more attributes, through "joining" other relational tables. This problem has been studied by [30] for data visualization, and [21] for database browsing. (2) *Attribute selection.* When the number of enriched attributes from other tables is large, we use the method [4] to rank attributes and select a reasonable number (10 by default) of attributes.

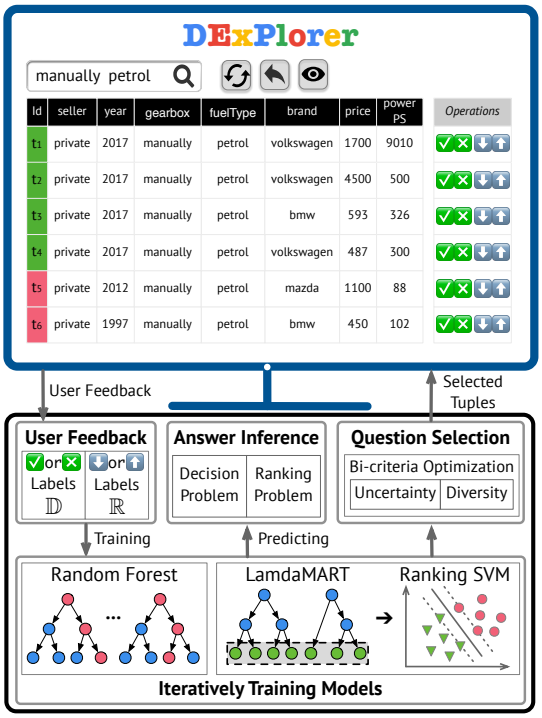



Fig. 1. An overview of DExPlorer

## III. Overview of DExPlorer

### A. System Overview (Figure 1)

**Front-end.** It interacts with the user in multiple iterations until user budget is used up or the answer cannot be improved. At each iteration, it provides a question $I$ with $k$ tuples, on which two operations are permissible: (1) "*click*" to annotate a tuple to be either *true* or *false*; and (2) "*drag*" to annotate that one is ranked higher than another.

The answers annotated by the user are then transformed to a set of *true/false* labels of tuples in $\mathbb{I}$, and a set $\mathbb{R}$ of partial orders between pairs of tuples in $\mathbb{I}$.

**Back-end.** In the $i$-th iteration, the user will provide a set $\mathbb{D}_i$ of *true/false* labels and a set $\mathbb{R}_i$ of partial orders, the back-end of DExPlorer needs to address two problems: answer inference and question selection.

*Answer Inference.* Given the user feedback from all $i$ iterations, *i.e.,* $\{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_i\}$ and $\{\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_i\}$, it is to infer the (ranked) result $R$.

*Question Selection.* It is to select a set $\mathbb{I}_i$ with $k$ tuples for the user to annotate in the $i$-th iteration.

The major challenge is that question selection for IDE is a bi-criteria problem that needs to estimate the *true/false* labeling and the partial order labeling between tuples.

*Termination.* The process will terminate, either if the user budget is used up, or the back-end inference will converge.

To see more about the system demonstration, please refer to our recent demo paper [27]. And in the following, we will focus on answer inference. The details for question selection will be discussed in Section IV.

### B. Answer Inference

Note that if the function $f()$ can be expressed by a simple SQL query, then using previous methods (*e.g.,* SQLSynthesizer [33]) is enough. In practice, nevertheless, it might be hard to capture a user's query intent by simple SQL queries. Consequently, ML-based methods (*e.g.,* using decision trees in [19]) have been used for the IDE-decision problem, and different mathematical models [26], [32] have been studied to model the IDE-ranking problem. As discussed earlier in Table I, existing methods fall short of modeling many real-world cases. Hence, we advocate to jointly use several more advanced ML models for interactive data exploration.

**A Naïve Solution.** A straightforward solution is to learn a function $f(T)$ (for example, a learning-to-rank [15] model such as Ranking SVM [20]) that can rank and score tuples in $T$ (*e.g.,* $\langle (t_1, 0.99), (t_2, 0.98), \ldots, (t_n, 0.04) \rangle$) and select a threshold (*e.g.,* $\theta = 0.8$) to separate desired and undesired tuples – all the tuples above the threshold are returned as desired tuples and ranked based on their corresponding scores.

***Example 3:*** Suppose a user wants to buy a BMW car, and ranks them by a weighted sum function: -0.018 $*$ price $+$ 0.982 $*$ powerPS, the ground truth $Q$ can be expressed as:

```
SELECT * FROM Car WHERE brand = "bmw"
ORDER BY -0.018 * price + 0.982 * powerPS DESC;
```
$Q_2$

Let's use Ranking SVM to rank tuples, we may get weights $w_1, w_2$ for attribute price and powerPS, and $w_3$ for one-hot attribute of the value bmw; that is, the score of a tuple $t$ is $w_1 \times t[\text{price}] + w_2 \times t[\text{powerPS}] + w_3 \times t[\text{brand=bmw}]$. However, if there is an Audi car with a high $w_1 \times t[\text{price}] + w_2 \times t[\text{powerPS}]$ value, it may be ranked high although it is not desired. □

Example 3 shows that although Ranking SVM (or other learning-to-rank models) can be used, it is not ideal to handle the combined cases of both decision and ranking. Similarly, a decision-only model (*e.g.,* a binary classifier) is also not ideal.

Next, we will discuss how DExPlorer does the decision answer inference, the ranking answer inference, and then the hybrid approach of combining them for the IDE problem.

**Decision Inference.** Essentially, it is a binary classification problem – deciding whether a tuple is desired or not. There are several choices: decision tree (DT), random forests (RF) [22], or support vector machines (SVM) [3]. [6] uses DT for IDE-Decision. However, as observed by [19], DT is not ideal to capture complex predicates.

DExPlorer employs RF for three reasons.

(1) RF, a collection of many DTs, is more robust to capture complicated cases (*e.g.,* with *if-then-else*, *and*, *or* and *not*) [29] and better prevents overfitting, compared with DTs.

(2) In some cases, RF, with shallow decision trees and a few branches, is still interpretable. That is, for simple cases, using RF does not sacrifice too much interpretability.

(3) As will be seen shortly in question selection (Section IV), RF, which outputs a vector with 0's and 1's (each DT in an RF will output either 0 or 1 for one input), is a better choice

than SVM that gives a single value for computing diversity between tuples, an important feature for question selection.

**Ranking Inference.** We consider to support linear weighted ranking functions, because in many applications, especially in databases with numeric values, a weighted sum among multiple attributes is widely used to model user preferences, which is also observed by [20], [26].

The ranking problem in DExPLORER is to learn a function $g(t)$. We employ Ranking SVM [20], [26] as the basic model for two reasons. (1) The scoring function used to rank tuples can be roughly captured by a linear function, *i.e.,* $g(t) = \vec{w} \cdot \vec{t}$, where $\vec{w}$ is a weight vector quantifying the preference and importance of attributes, and $\vec{t}$ is the feature vector of tuple $t$. Ranking SVM is a natural choice for this case. (2) Compared with complicated model like RankNet [1], Ranking SVM can avoid overfitting by using a small number of training data.

However, one drawback of using a lightweight model, *e.g.,* Ranking SVM, is that it needs sophisticated feature engineering. One way to combat this is to use another model to capture more *distinguishing features*, which is inspired by a classical solution for Click-Through Rate (CTR) Prediction.

In many commercial IR systems [14], [16], the solution for CTR follows the GBDT + LR [16] framework, where GBDT is Gradient Boosting Decision Tree and LR is Linear Regression. More specifically, the CTR problem is a binary classification problem to predict whether an advertisement will be clicked by a user. In the model, the sub-model GBDT can capture distinguishing features combination and LR is used to make prediction based on these features.

**LambdaMART + Ranking SVM.** Inspired by the GBDT + LR model, we propose a hybrid ranking model: LambdaMART + Ranking SVM, where LambdaMART [31] is a tree structure model based on the MART (Multiple Additive Regression Tree) [10] that transforms the input features called *feature transformation* (see Figure 1). The "Lambda" in LambdaMART denotes a special Lambda value which is the negative gradient in the MART algorithm. The output of LambdaMART is then fed to Ranking SVM to infer tuple ranking.

### C. Iteratively Training and Predicting

Given $\mathbb{D}_i$ for decision labels and $\mathbb{R}_i$ for partial order labels in the $i$-th iteration, next we describe how to use them for decision and ranking training and prediction.

**Training.** (1) *Decision inference using RF:* it incrementally trains RF by feeding new labels in $\mathbb{D}_i$, obtaining $RF_i$. (2) *Ranking inference using LambdaMART + Ranking SVM:* Given a set of features of ordered tuple pairs $\mathbb{R}_i$ as training examples: (2.a) We first use $\mathbb{R}_i$ to incrementally train the LambdaMART model, obtaining $LM_i$; (2.b) For each tuple $t$ (whose one-hot encoding feature vector is denoted as $\vec{t}'$) appearing in $\mathbb{R}_i$, $LM_i(t)$ will output a $m$-dimension transformed feature vector $(x_1, \ldots, x_m)$, where $m$ is the number of trees in LambdaMART (see Figure 1), and $x_j$ ($j \in [1, m]$) is the leaf node index of $\vec{t}'$ that ends up falling in the $j$-th tree. (2.c) Let $\vec{t}''$ be the one-hot encoding of the transformed feature $(x_1, \ldots, x_m)$. (2.d) Let $\vec{t}$ be the concatenation ($\oplus$) of $\vec{t}'$ and $\vec{t}''$. (2.e) For each ordered tuple pair $(t_y, t_z)$ in $\mathbb{R}_i$, we use the enriched features $(\vec{t}_y, \vec{t}_z)$ to incrementally train Ranking SVM, obtaining $RS_i$.

***Example 4:*** Consider the 2 trees in Figure 1 generated by LambdaMART. The 1st tree has 4 leaves and the 2nd tree has 3 leaves. Assume that $\vec{t}' = (0.8, 0.7)$ ends up falling in the 2nd and 3rd leaf node of tree 1 and tree 2, respectively. Thus the transformed feature for $\vec{t}'$ is $(1, 2)$, with the corresponding one-hot encoding for the 1st and 2nd features to be $(0, 1, 0, 0)$ and $(0, 0, 1)$, respectively. Hence, $\vec{t}'' = (0, 1, 0, 0, 0, 0, 1)$; $\vec{t} = \vec{t}' \oplus \vec{t}'' = (0.8, 0.7, 0, 1, 0, 0, 0, 0, 1)$. □

**Prediction.** (1) *Decision prediction:* it uses $RF_i$ to predict each unlabeled tuple, predicting $R_i$ as a set of desired tuples. (2) *Ranking prediction:* For each $t \in R_i$: (2.a) It uses trained $LM_i$ to get the enriched features $\vec{t}$; (2.b) It computes for $t$ a score using the learned weight function $\vec{w}$ in $RS_i$ as $\vec{w} \cdot \vec{t}$; and (2.c) It ranks tuples in $R_i$ based on their scores.

## IV. QUESTION SELECTION

In each user iteration, we need to select a list of $k$ tuples from the table $T$ as one question $\mathbb{I}$. There are three challenges for the problem of IDE question selection.

**Challenge 1.** [*Reducing Uncertainty for Decision and Ranking*.] Note that we have different models for decision and ranking answer inference. Also, the training examples for decision model are tuples, while the training examples for the ranking model are tuple pairs. Therefore, the first challenge is how to select tuples such that the uncertainty of both the decision and ranking models is reduced.

**Challenge 2.** [*Increasing Diversity of Tuple List*.] Existing works either provide sample tuples for decision question [28], [9], [19] or partial orders [2], [26], [32] for ranking questions. Different from them, DExPLORER provides a list of tuples as one question for one user iteration. Hence, besides that the tuples in the list should reduce uncertainty of both models, the tuples should also be diverse and representative.

**Challenge 3.** [*Interactive Question Selection*.] As will be shown shortly, the IDE question selection problem is NP-Hard, and existing algorithms cannot provide the question $\mathbb{I}$ in interactive time. Thus how to design an efficient yet effective algorithm for IDE question selection is important.

### A. Uncertainty and Diversity

Uncertainty is the most commonly used criteria for tuple selection (*i.e.,* **Challenge 1**), which measures the confidence of the current model on evaluating a question.

**Uncertainty for Decision Questions.** We define the uncertainty of a tuple $t$ as the entropy of the predicted results of all decision trees in the random forest. Suppose we have $n$ trees, and $m$ is the number of trees which predicted the tuple $t$ as positive, then the uncertainty of $t$ is defined as: $u(t) = -(e \log e + (1 - e) \log(1 - e))$, where $e = \frac{m}{n}$. The larger the $u(t)$, the larger the uncertainty.

***Example** 5:* Suppose we have 5 trees in RF. For a tuple $t$, assume that there are 3 trees predicting $t$ as positive, then the uncertainty of $t$ is $u(t) = -(\frac{3}{5}\log\frac{3}{5} + \frac{2}{5}\log\frac{2}{5}) = 0.97$. □

**Uncertainty for Ranking Questions.** Given a pair of tuples $t_i$ and $t_j$, the Ranking SVM model uses a hyperplane to make decision about whether $t_i \succ t_j$. It also learns a parameter vector $\vec{\mathsf{w}}$, such that $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i$ denotes the ranking score of tuple $t_i$. The higher the score is, the higher the ranking $t_i$ should be. If the pair is above the hyperplane, *i.e.,* if $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j > 0$, then $t_i \succ t_j$, and vice versa. Thus, those pairs close to the hyperplane are the uncertain ones, *i.e.,* when $|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j|$ is close to 0.

***Example** 6:* Suppose we have 3 tuples $t_1$, $t_2$ and $t_3$, a parameter vector $\vec{\mathsf{w}} = (0.5, 0.5)$, and $k = 2$. We represent the three tuples by their one-hot vectors, *i.e.,* $\vec{\mathsf{t}}_1 = (0.8, 0.7), \vec{\mathsf{t}}_2 = (0.8, 0.6), \vec{\mathsf{t}}_3 = (0.6, 0.6)$. Then, we compute $|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j|$ for each pair, *i.e.,* $|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_1 - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_2| = 0.05, |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_2 - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_3| = 0.1, |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_1 - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_3| = 0.15$. Since the score of $|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_1 - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_2| = 0.05$ is the closest to 0 (*i.e.,* the closest to the hyperplane), we choose tuples $t_1$ and $t_2$ for preference labeling. □

Besides selecting tuples that the decision and ranking models are uncertain about, we should also care about the diversity of the selected tuples (*i.e.,* **Challenge 2**). Consider two tuples with high uncertainty but with highly similar content, it is a waste of human efforts to label both of them because only labeling one of them can produce a model that is very likely to predict the other correctly. Naturally, we should also consider the diversity of selected tuples.

**Diversity.** A simple way to measure the diversity is to compute the string similarity of each tuple pair using some predefined function. However, such method does not consider the semantic information (dynamically) incorporated by user feedback. Thus, we can use the feature vectors produced by RF and LambdaMART to measure the similarity of two tuples. Let $\vec{\mathsf{v}}'(t)$ be the prediction vector of all trees in RF for tuple $t$, where the $i$-th element in $\vec{\mathsf{v}}'(t)$ denotes the prediction label (1 or 0) of the $i$-th tree in the random forest. Let $\vec{\mathsf{v}}''(t)$ be the one-hot encoding transformed feature vector of tuple $t$ output by the LambdaMART sub-model (*i.e.,* $\vec{\mathsf{v}}''(t) = \vec{\mathsf{t}}''$). We concatenate $\vec{\mathsf{v}}'(t)$ and $\vec{\mathsf{v}}''(t)$ as $\vec{\mathsf{v}}(t)$, which is used to compute the diversity. More specifically, we define the similarity $s(t_i, t_j)$ of tuple $t_i$ and $t_j$ using the Cosine Similarity: $s(t_i, t_j) = cos(\vec{\mathsf{v}}(t_i), \vec{\mathsf{v}}(t_j))$. The smaller the similarity, the larger the diversity between two tuples.

***Example** 7:* Consider two tuples $t_1$ and $t_2$. The prediction vectors of RF for $t_1$ and $t_2$ are $\vec{\mathsf{v}}'(t_1) = (1, 0, 1), \vec{\mathsf{v}}'(t_2) = (1, 1, 1)$. The one-hot encoding transformed features for $t_1$ and $t_2$ are $\vec{\mathsf{v}}''(t_1) = (0, 1, 0, 1), \vec{\mathsf{v}}''(t_2) = (1, 0, 0, 1)$. Then $\vec{\mathsf{v}}(t_1) = (1, 0, 1, 0, 1, 0, 1), \vec{\mathsf{v}}(t_2) = (1, 1, 1, 1, 0, 0, 1)$, and the similarity of $t_1$ and $t_2$ is $s(t_1, t_2) = cos(\vec{\mathsf{v}}(t_1), \vec{\mathsf{v}}(t_2)) = 0.67$. □

Considering both the uncertainty and diversity, we define the IDE question selection problem as follows.

*Definition 1 (IDE Question Selection):* Given a partially trained RF model and hybrid ranking model, a table $T$, and a number $k$, the problem is to select a set of $k$ tuples $S^*$ from $T$ such that the following equation is minimized:

$$S^* = \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{t \in S} (1 - u(t)) + \alpha \sum_{t_i, t_j \in S} |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j| + \beta \sum_{t_i, t_j \in S} s(t_i, t_j) \tag{1}$$

where $u(t)$ is the normalized uncertainty of tuple $t$, $\alpha$ is a parameter to trade-off decision and ranking questions, $\vec{\mathsf{w}}$ is the weight vector output by the hybrid ranking model, $\beta$ is a parameter that provides a trade-off between the uncertainty and diversity, and $s(t_i, t_j)$ is the similarity of $t_i$ and $t_j$.

***Theorem** 1:* The IDE question selection is NP-hard.

We can prove that the IDE question selection problem is NP-hard by a reduction from the MAXSUMDISPERSION problem that is known to be NP-hard [12].

### *B. Question Selection Algorithms*

*1) **AQS: An Approximate Algorithm**:* Inspired by [11], we present a 2-approximation algorithm, denoted as AQS, for IDE question selection problem. The algorithm first initializes an empty result set and calculates a new similarity $s'(t_i, t_j) = \frac{1-u(t_i)}{k-1} + \frac{1-u(t_j)}{k-1} + \alpha|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j| + \beta s(t_i, t_j)$ for each tuple pair $(t_i, t_j)$. It then selects the tuple pair that has not been added to the result set and with the least $s'$ scores, and adds the pair to the result set. It iteratively runs $\lfloor \frac{k}{2} \rfloor$ times. It adds another tuple to the result set if $k$ is odd and current result has $k - 1$ tuples. The selected $k$ tuples are then returned.

The time complexity of AQS is $O(\max(|T|^2 L, |T|^2 k))$, where $L$ is the length of tuples. However, the time to compute a question can still be long when $|T|$ is large, thus the above algorithm is not efficient enough to satisfy the online question selection requirement.

*2) **IQS: An Efficient Algorithm**:* Considering both uncertainty and diversity is hard due to its high computational complexity (NP-hard). We thus propose to first solve the IDE question selection by only considering decision and ranking uncertainty (*i.e.,* $\beta = 0$), and then incorporate the diversity into the solution.

**Question Selection without Diversity.** We set $\beta$ = 0 in Eq. (1) to ignore diversity:

$$S^* = \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{t \in S} (1 - u(t)) + \alpha \sum_{t_i, t_j \in S} |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j| \tag{2}$$

To better illustrate the optimization problem, we first denote $S = [t_1, t_2, ..., t_k]$, where $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i \leq \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j$ iff $i \leq j$, then we expand the second term of Eq. 2 to Eq. 3. Thus we have:

$$\begin{aligned} S^* &= \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{i=1}^{k} (1 - u(t_i)) + \alpha \sum_{i=1}^{k} \big((i-1)\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - (k-i)\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i\big) \\ &= \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{i=1}^{k} (1 - u(t_i)) + \alpha \sum_{i=1}^{k} (2i - k - 1)\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i \end{aligned} \tag{3}$$

We first introduce some notations. We sort $T$ by $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}$ in ascending order and obtain a sorted list $T = [t_1, t_2, ..., t_{|T|}]$. We use $T_m$ to denote the prefix of list $T$ with length $m$, *i.e.,* $T_m = [t_1, t_2, ..., t_m]$. We define $S(m, n) = \underset{S \subseteq T_m, |S|=n}{\arg\min} \sum_{i=1}^{|S|} (1 -$

$u(t_i)) + \alpha \sum_{i=1}^{|S|}(2i-k-1)\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i$ and $F(m,n)$ denotes the corresponding optimal value. We can see that $S(|T|,k)$ is the optimal solution of Eq. 2 and $F(|T|,k)$ is the corresponding optimal value. Then we have:

$$F(m,n) = \min(F(m-1,n), F(m-1,n-1)+\phi(m,n)) \quad (4)$$

where $\phi(m,n) = (1-u(t_m))+\alpha(2n-k-1)\vec{\mathsf{w}}\vec{\mathsf{t}}_m$. We devise a dynamic programming algorithm to find the optimal solution.

*Dynamic-Programming for Uncertainty Only.* It first initializes $F[0...|T|][0...k]$ to all zero, where $F[m][n] = F(m,n)$ and initializes $S[0...|T|][0...k]$ to all $\emptyset$, where $S[m][n] = S(m,n)$. Then we can calculate $F$ by Eq. 4 and update $S$ correspondingly. Finally, we can derive $F[|T|][k]$ and $S[|T|][k]$.

**Question Selection Considering both Uncertainty and Diversity.** When considering both uncertainty and diversity, we optimally choose tuples with high uncertainty by the above dynamic programming algorithm, but when adding a tuple $t$ to the result set $S$, we check whether $t$ has a high similarity with existing selected tuples. If yes, we just drop it; else, we include it to the result.

Algorithm 1 (IQS) shows how to do question selection with both uncertainty and diversity. Based on the dynamic programming algorithm, we add a constraint to maintain the selected tuples diversified (*i.e.,* the similarity between these selected $n$ tuples is less than a threshold $\delta$) as far as possible.

The algorithm first sorts all tuples in $T$ by $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}$ (line 1), and gets the sorted list $T = [t_1, t_2, ..., t_{|T|}]$. Then it initializes all $F[0...|T|][0...k]$ to zeros (line 2), and $S[0...|T|][0...k]$ to $\emptyset$ (line 3). First, the same as the above uncertainty-only algorithm, if $F[m-1][n] \leq F[m-1][n-1] + \phi(m,n)$, we will drop $t_m$ (lines 7-8) because discarding $t_m$ will get a better solution. Otherwise, the algorithm checks whether the tuple $t_m$ has a high similarity with existing tuples which have been added into the current result (lines 10) using the function $hasHighSimilarity(t,S,\delta)$. If so, $t_m$ is also dropped to keep high diversity (lines 11-12). Otherwise, $t_m$ is added to the diversified solution of $S[m][n]$ (lines 14-15). More specifically, $hasHighSimilarity(t,S,\delta)$ checks whether there exists a tuple $t' \in S$ such that $s(t,t') > \delta$. If so, it returns $true$, and $false$ otherwise. Finally, $S[|T|][k]$ is returned as the selected question with $k$ tuples (line 16).

*Complexity.* The time complexity of the *hasHighSimilarity* function is $O(kL)$, where $L$ is the length of tuples. Note that the attributes number in $T$ is small, thus the function is effective. We first sort $T$ (line 1), and the time complexity for sorting is $O(|T|\log|T|)$. The time complexity for the lines 4 - 15 is $O(k^2L|T|)$. Thus the time complexity for Algorithm 1 is $O(\max(|T|\log|T|, k^2L|T|))$.

***Example** 8:* Figure 2 shows an example of computing $S$ when considering only uncertainty (Figure 2(c)) and both uncertainty and diversity (Figure 2(d)). Suppose we have 10 tuples, and want to select 5 tuples for labeling. Figure 2(a) shows the variables for calculating decision and ranking uncertainty of $t_1, ..., t_{10}$, and Figure 2(b) shows the similarity of

**Input**: $T$, $k$, a weight vector $\vec{\mathsf{w}}$, a diversity threshold $\delta$
**Output**: an approximate optimal set $S$

```
1  sort T by ascending order of w·t, T ← [t_1, t_2, ..., t_|T|];
2  initialize F[0...|T|][0...k] ← 0;
3  initialize S[0...|T|][0...k] ← ∅;
4  for m ← 1 to |T| do
5      for n ← 1 to min(m − 1, k) do
6          if F[m − 1][n] ≤ F[m − 1][n − 1] + φ(m, n) then
7              F[m][n] ← F[m − 1][n];
8              S[m][n] ← S[m − 1][n];
9          else
10             if hasHighSimilarity(t_m, S[m − 1][n − 1], δ)
               then
11                 F[m][n] ← F[m − 1][n];
12                 S[m][n] ← S[m − 1][n];
13             else
14                 F[m][n] ← F[m − 1][n − 1] + φ(m, n);
15                 S[m][n] ← S[m − 1][n − 1] ∪ {t_m};
16 return S[|T|][k];
```

**Algorithm 1**: IDE QUESTION SELECTION (IQS)

any tuple pairs. For ease of description, let $\alpha = \beta = 1$ and $\delta = 0.9$. Figure 2(c) shows the matrix $S$ where each cell is a solution without considering diversity during the dynamic programming process and Figure 2(d) shows the matrix $S'$ when diversity is considered .

The cells colored in blue in the matrix $S'$ (Figure 2(d)) are different from those cells in $S$ (Figure 2(c)) because we consider diversity. The cells $S'[m][n]$ marked by green in Figure 2(d) are copied from $S'[m-1][n]$ by dropping $t_m$ because $t_m$ has high similarities with some tuples in $S'[m-1, n-1]$ even when $F[m-1][n-1] + \phi(m,n) \leq F[m-1][n]$. Take the cell $S[6][2]$ and the green cell $S'[6][2]$ as examples. First, when we only consider the uncertainty to compute the $S[6][2]$, we have $F[6][2] = \min(F[5][2], F[5][1] + \phi(6,2))$, $F[5][2] = -2.67$, and $F[5][1]+\phi(6,2) = -3.84$. We then observe that $F[5][1]+\phi(6,2) < F(5,2)$, thus adding $t_6$ to $S[5][1]$ is the optimal solution for selecting 2 tuples in $\{t_1, t_2, ..., t_6\}$ ($S[6][2] = \{t_5, t_6\}$). However, now we consider the diversity to compute $S'[6][2]$. Before adding $t_6$ to $S'[5][1]$, *i.e.,* $\{t_5\}$, we first check whether $t_6$ has high similarity with $\{t_5\}$. Because $s(t_5, t_6) = 0.97 > \delta = 0.9$ in Figure 2(b), we cannot add $t_6$ to $S'[5][1]$, and thus $S'[6][2] = S'[5][2] = \{t_4, t_5\}$. □

## V. SPECIAL CASES

In this section, we discuss two special cases: IDE-Decision (only *true/false* labels) and IDE-Ranking (only partial orders).

*Definition 2 (IDE-Decision Question Selection):* Given a partially trained RF model, a table $T$, and a number $k$, the problem is to select a set of $k$ tuples $S^*$ from $T$ so that the following equation is minimized:

$$S^* = \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{t \in S}(1-u(t)) + \beta \sum_{t_i, t_j \in S} s'(t_i, t_j) \quad (5)$$

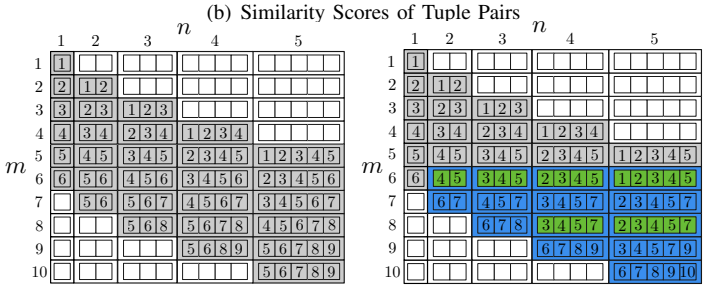
where $u(t)$ is the normalized uncertainty of tuple $t$, $\beta$ is a parameter that provides a trade-off between uncertainty and diversity, $s'(t_i, t_j)$ is the similarity of $t_i$ and $t_j$ defined by

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $1-u(t_i)$ | 1.0 | 0.12 | 0.28 | 0.53 | 0.12 | 0.12 | 0.53 | 0 | 0.12 | 0.53 |
| $\vec{\mathsf{w}}\vec{\mathsf{t}}_i$ | 0.002 | 0.16 | 0.26 | 0.49 | 0.68 | 0.68 | 0.70 | 0.79 | 0.80 | 0.94 |

(a) Variables for Computing Decision and Ranking Uncertainty

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $t_2$ | 0.28 | | | | | | | | |
| $t_3$ | 0.09 | 0.45 | | | | | | | |
| $t_4$ | 0.02 | 0.77 | 0.04 | | | | | | |
| $t_5$ | 0.63 | 0.03 | 0.20 | 0.68 | | | | | |
| $t_6$ | 0.84 | 0.78 | 0.50 | 0.81 | 0.97 | | | | |
| $t_7$ | 0.13 | 0.18 | 0.68 | 0.79 | 0.29 | 0.61 | | | |
| $t_8$ | 0.42 | 0.21 | 0.81 | 0.92 | 0.01 | 0.68 | 0.63 | | |
| $t_9$ | 0.07 | 0.27 | 0.80 | 0.65 | 0.13 | 0.42 | 0.07 | 0.87 | |
| $t_{10}$ | 0.26 | 0.52 | 0.95 | 0.25 | 0.79 | 0.77 | 0.77 | 0.01 | 0.79 |

(b) Similarity Scores of Tuple Pairs




(c) Matrix $S$ (no diversity) (d) Matrix $S'$ (with diversity)

Fig. 2. Example of Computing $S$

the Cosine Similarity of $\vec{\mathsf{v}}'(t_i)$ and $\vec{\mathsf{v}}'(t_j)$, *i.e.,* $s'(t_i, t_j) = cos(\vec{\mathsf{v}}'(t_i), \vec{\mathsf{v}}'(t_j))$.

**Solution for IDE-Decision:** We still solve the IDE-Decision problem using algorithm 1 but transform Eq. 1 to Eq. 5 by setting $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}} = 0$ and replacing $s(t_i, t_j)$ with $s'(t_i, t_j)$.

*Definition 3 (IDE-Ranking Question Selection):* Given a partially trained hybrid ranking model, a table $T$, and a number $k$, the problem is to select a set of $k$ tuples $S^*$ from $T$ so that the following equation is minimized:

$$S^* = \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{t_i, t_j \in S} |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j| + \beta \sum_{t_i, t_j \in S} s''(t_i, t_j) \quad (6)$$

where $\vec{\mathsf{w}}$ is the weight vector output by the hybrid ranking model, $\beta$ is a parameter that provides a trade-off between the uncertainty and diversity, $s''(t_i, t_j)$ is the similarity of $t_i$ and $t_j$ defined by the Cosine Similarity of $\vec{\mathsf{v}}''(t_i)$ and $\vec{\mathsf{v}}''(t_j)$, *i.e.,* $s''(t_i, t_j) = cos(\vec{\mathsf{v}}''(t_i), \vec{\mathsf{v}}''(t_j))$.

**Solution for IDE-Ranking:** We also solve the IDE-Ranking problem using Algorithm 1 but transform Eq. 1 to Eq. 6 by setting $u(t) = 1$ and replacing $s(t_i, t_j)$ with $s''(t_i, t_j)$.

## VI. Experiments

**Datasets.** We use two datasets: `Car` and `Pub` (for publications). The `Car` dataset is from the Kaggle (https://www.kaggle.com/orgesleka/used-cars-database) with one relational table, and the `Pub` dataset is crawled from the ACM Digital Library (https://www.acm.org/publications/digital-library) that contains six relational tables. Table II gives more statistics of these two datasets.

**The Ground Truth SQL Queries.** We use SQL queries as the proxy for user's query intent, because providing other types of ground truth is subjective and thus not feasible. Generally speaking, an SQL query for an IDE problem can be considered as a combination of an IDE-Decision and an IDE-Ranking problem, which is specified by the **SELECT**, **WHERE**, followed by the **ORDER BY** clause.

| Dataset | Table | #-Tuples | #-Col | #-Cat | #-Num |
|---|---|---|---|---|---|
| **Car** | **Car** | 248419 | 14 | 10 | 4 |
| **Pub** | **Author** | 6881 | 3 | 1 | 2 |
| | **Institution** | 1453 | 4 | 2 | 2 |
| | **Paper** | 1947 | 3 | 1 | 2 |
| | **Conference** | 16 | 5 | 3 | 2 |
| | **Paper_Author** | 8615 | 2 | 0 | 2 |
| | **Paper_Keyword** | 2322 | 2 | 1 | 1 |

TABLE II
STATISTICS OF DATASETS (#-COL: #-COLUMN; #-CAT: #-CATEGORICAL; #-NUM: #-NUMERIC)

The **decision** cases of the tested SQL queries are shown in Table IV: $D_1$–$D_8$ are for dataset `Car`, and $D_9, D_{10}$ are for dataset `Pub`. These cases are designed to be representative and cover many different cases. More specifically, we designed 5 types of queries: AND (only *and* conjunction is included in the selective condition), OR (only *or* disjunction), NO (only *not* exclusion), MIXED (a mix of *and, or, not*), and JOIN (multiple tables are involved). In addition, Table IV also shows the size of the result of the query $D_i$ in the column $|D|$, and its selectivity in the column "%" (*e.g.,* $|D_1|/|\texttt{Car}| = 5.77\%$).

The **ranking** cases of the tested SQL queries are shown in Table V: $R_1$–$R_4$ are for dataset `Car`, and $R_5$ is for dataset `Pub`. They are categorized into 3 types: Hierarchical Sorting (sorted by multiple attributes hierarchically), Weighted Sum Sorting (sorted by a weighted sum score), and Hierarchical Weighted Sorting (sorted by combining the above two types).

The 10 tested SQL queries, $Q_1$–$Q_{10}$, for the **IDE** problem are shown in Table III, which combine the cases in Table IV and Table V. Here, the $D$ and $R$ denote the cases in Table IV and Table V, respectively. For example, $Q_1$ is obtained by combining $D_1$ for decision and $R_1$ for ranking.

Note that, in each iteration, the selected tuples are labeled by a simulated user using the ground truth.

**Environment.** All experiments are conducted on a MacBook Pro with 16 GB 1600 MHz RAM and 2.5 GHz Intel Core i7 CPU, running OS X Version 10.14.5.

**Parameters.** We include tuples ($k = 10$) in one question for each interaction with the user. We will ask 20 iterations for all datasets and show the performance during this period. To bootstrap, we allow the user to input one keyword, *e.g.,* the user can input "bmw" for $Q_1$. Moreover, we need to utilize $\alpha$ and $\beta$ to balance the decision and ranking, as well as uncertainty and diversity. The method is as follows. We observe that the first item in Equation 1 (*i.e.,* $\sum_{t \in S}(1-u(t))$) has $k$ elements, and stands for uncertainty in the decision problem; the second item (*i.e.,* $\sum_{t_i, t_j \in S} |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j|$) and the third item (*i.e.,* $\sum_{t_i, t_j \in S} s(t_i, t_j)$) both have $\binom{k}{2}$ elements, and stand for uncertainty in ranking problem and diversity, respectively. Therefore, to make decision and ranking, and uncertainty and diversity equivalently important, we should make sure $k = \binom{k}{2}\alpha$ and $k + \binom{k}{2}\alpha = \binom{k}{2}\beta$ (*i.e.,* $\alpha = \frac{2}{k-1}, \beta = \frac{4}{k-1}$).

| $Q$ | Dataset | $D$ | $R$ | $Q$ | Dataset | $D$ | $R$ |
|---|---|---|---|---|---|---|---|
| $Q_1$ | Car | $D_1$ | $R_1$ | $Q_6$ | Car | $D_6$ | $R_1$ |
| $Q_2$ | Car | $D_2$ | $R_2$ | $Q_7$ | Car | $D_7$ | $R_3$ |
| $Q_3$ | Car | $D_3$ | $R_3$ | $Q_8$ | Car | $D_8$ | $R_4$ |
| $Q_4$ | Car | $D_4$ | $R_4$ | $Q_9$ | Pub | $D_9$ | $R_5$ |
| $Q_5$ | Car | $D_5$ | $R_2$ | $Q_{10}$ | Pub | $D_{10}$ | $R_5$ |

TABLE III
TESTED IDE PROBLEMS

Therefore, we set $\alpha = 0.22$ and $\beta = 0.44$ for $k = 10$.

### A. IDE Problem

*1) **Effectiveness**:* **Metrics.** The Kendall tau distance [5] is a widely used metric for quantifying two ranking lists [7], [13], [8], with the basic idea of counting the number of pairwise disagreements between two ranking lists. However, in our case, we do not treat the result as a single ranked list. Instead, we consider two parts: $Q(T)$ as the ground truth desired and ranked tuples, and $Q'(T) = T \setminus Q(T)$ as the undesired tuples. Given any two tuples $t_i, t_j \in T$, the ground truth partial order between them is either $t_i \succ t_j$ if (1) both $t_i$ and $t_j$ are in $Q(T)$ but $t_i$ is ranked higher than $t_j$, or (2) $t_i \in Q(T)$ but $t_j \in Q'(T)$; or $t_i \equiv t_j$ when $t_i, t_j \in Q'(T)$.

Let $R$ be the desired and ranked tuples predicted by an algorithm, and $R' = T \setminus R$. Given any tuple pair $(t_i, t_j)$, based on $R$ and $R'$, it will predict a label, which is either $t_i \succ t_j$ or $t_i \equiv t_j$, similar to the above process.

Let $N = \binom{|T|}{2}$ be the total number of tuple pairs. Let $KD$ be the "disagreement" of all tuple pairs, *i.e.,* the total number of tuple pairs whose predicted labels are different from ground truth labels. We use the normalized Kendall tau distance as the evaluation metric, denoted by $accuracy = \frac{N-KD}{N}$.

***Example** 9:* Suppose $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. The ground truth is $Q(T) = \{t_1, t_2, t_3\}$ that is ranked as $t_1 \succ t_2 \succ t_3$, and $Q'(T) = \{t_4, t_5, t_6\}$ (*i.e.,* $t_4 \equiv t_5 \equiv t_6$ for their ranking). The predicted result is $R = \{t_1, t_2, t_4\}$ that is ranked as $t_1 \succ t_4 \succ t_2$, and $R' = \{t_3, t_5, t_6\}$. The total number of tuple pairs is $N = \binom{6}{2} = 15$. There are $KD = 6$ mistakenly predicted tuple pairs: $\{t_4 \succ t_2, t_4 \succ t_6, t_3 \equiv t_5, t_3 \equiv t_6, t_4 \succ t_5, t_4 \succ t_6\}$. Therefore, $accuracy = \frac{15-6}{15} = 0.6$. □

Besides using *accuracy* to capture the global ranking quality, the users might also be interested in the top-$l$ results. Thus we also used $precision@l = \frac{|R[1:l] \cap Q(T)[1:l]|}{l}$, where $R[1:l]$ and $Q(T)[1:l]$ denote the set of top-$l$ tuples in $R$ and $Q(T)$, respectively.

**Comparisons.** We test (1) DEXPLORER uses the model in Section III-B for answer inference and IQS for question selection and (2) DEXPLORER-AQS uses the same answer inference model with DEXPLORER but different question selection algorithm AQS. We also compared with (3) SQL-Synthesizer [33] (denoted by SSY), which infers SQL by accepting ranked positive tuples, and (4) a learning-to-rank based solution as described in Section III-B using Ranking SVM, denoted by LTR. More specifically, LTR uses the ranking labels to train the Ranking SVM model, obtain a score for each tuple $t$ and rank based on the scores. Then we learn a threshold $\theta$ to separate the desired and undesired tuples, which results in the maximum information gain. (5) ERROR-10%: since the user may not always provide correct feedback, we also test the effectiveness of DEXPLORER when the answers of 10% questions are wrong.

**Exp-1: Accuracy of IDE:** Figure 3 shows the *accuracy* of the IDE problems $Q_1 - Q_{10}$ for DEXPLORER, DEXPLORER-AQS, LTR, SSY and ERROR-10%. Figure 3 shows that:

(1) With the increasing of #-Questions, the accuracy of DEXPLORER, DEXPLORER-AQS and SSY all increase, as expected. They finally achieve *accuracy* of 0.987, 0.968, 0.499 respectively on average of $Q_1 - Q_{10}$ after asking 20 questions. DEXPLORER behaves similarly to DEXPLORER-AQS, and even better in some cases (*i.e.,* $Q_7, Q_8$). However, as shown in Section VI-A2, DEXPLORER-AQS takes longer time (*i.e.,* more than 1 hour for $Q_1 - Q_8$), while DEXPLORER can return results in interactive time because our question selection algorithm, IQS, is efficient.

(2) SSY supports decision using a decision tree and supports ranking by one or more attributes hierarchically. More specifically, it only supports the ranking type like $R_1$ in Table V, so SSY can only capture the **ORDER BY** clause of $Q_1, Q_6$. Therefore, for $Q_1$ and $Q_6$, SSY learns the target SQL query, ranks the desired tuples correctly, and achieves an *accuracy* of 1.0. However, for $Q_3, Q_5, Q_9, Q_{10}$, their corresponding IDE-Decision problems are inferred by SSY correctly, but the IDE-Ranking problems are not learned. Thus the *accuracy* of SSY is mainly decided by the size of the query results. The number of query result of $Q_3, Q_9, Q_{10}$ are small, thus many tuple pairs which are not in the query result were correctly predicted, and the *accuracy* is high for $Q_3, Q_9, Q_{10}$. For $Q_5$, there are many tuples in the query result, but SSY does not capture the ranking information of tuples in the query result, thus the tuple pairs in the query result are all wrongly predicted, and thus the *accuracy* is low. For $Q_2, Q_4, Q_7, Q_8$, their corresponding IDE-Decision problems are so complex that the decision tree fails to capture the SQL. Thus their accuracy is not good. Based on the above analysis, we can conclude that SSY can only capture simple SQL queries for both decision and ranking, and the accuracy highly depends on the size of the query results, while DEXPLORER has a stable performance for all cases.

(3) LTR has a low accuracy even with the number of questions increasing: the final average *accuracy* achieved by LTR is 0.16, which is much less than that of DEXPLORER (0.987). Even if they both build models to learn the ranking, DEXPLORER performs much better than LTR because our system builds customized IDE-Decision model to discover desired tuples while LTR simply learns a threshold to do that. Therefore, we conclude that the IDE problem cannot be solved perfectly by a single model, i.e., the decision and ranking based models should work together to achieve a good performance.

(4) The more complex the query is, the more questions we need to answer to reach a certain *accuracy*. For example, $Q_6 - Q_8$ are more complex than $Q_1 - Q_5$, so the *accuracy* of $Q_6 - Q_8$ increases slower than $Q_1 - Q_5$ as shown in Figure 3. More specifically, for $Q_3$ (Figure 3(c)) and $Q_8$ (Figure 3(h)), when

| $D$ | Dataset | Type | SQL | $\|D\|$ | % |
|---|---|---|---|---|---|
| $D_1$ | Car | AND | **SELECT** * **FROM** Car **WHERE** brand="bmw" **AND** price≤10000 **AND** gearbox="manually" | 14325 | 5.77 |
| $D_2$ | Car | AND | **SELECT** * **FROM** Car **WHERE** seller="private" **AND** offerType="offer" **AND** price≤100000 **AND** abtest="control" **AND** vehicleType="limousine" **AND** year≥2000 **AND** gearbox="manually" **AND** powerPS≥150 **AND** kilometer≤150000 **AND** fuelType="petrol" **AND** brand="volkswagen" **AND** notRepairedDamage="no" | 545 | 0.22 |
| $D_3$ | Car | OR | **SELECT** * **FROM** Car **WHERE** brand="bmw" **OR** brand="audi" | 53288 | 21.45 |
| $D_4$ | Car | OR | **SELECT** * **FROM** Car **WHERE** price≤100 **OR** abtest="test" **OR** vehicleType="suv" **OR** year= 2018 **OR** powerPS≥1500 **OR** model="verso" **OR** kilometer≤5000 **OR** brand="volvo" | 136984 | 55.14 |
| $D_5$ | Car | NO | **SELECT** * **FROM** Car **WHERE** brand≠"bmw" | 219101 | 88.2 |
| $D_6$ | Car | MIXED | **SELECT** * **FROM** Car **WHERE** (brand="bmw" **AND** price≤10000) **OR** brand="volkswagen" | 71315 | 28.71 |
| $D_7$ | Car | MIXED | **SELECT** * **FROM** Car **WHERE** fuelType="petrol" **AND** year≥2010 **AND** gearbox="manually" **AND** ((brand="bmw" **AND** price≤10000) **OR** (brand="volkswagen" **AND** price≤8000)) | 596 | 0.24 |
| $D_8$ | Car | MIXED | **SELECT** * **FROM** Car **WHERE** fuelType!="petrol" **AND** year≥2010 **AND** gearbox="manually" **AND** ((brand="bmw" **AND** price≤10000) **OR** (brand="volkswagen" **AND** price≤8000)) | 3541 | 1.43 |
| $D_9$ | Pub | JOIN | **SELECT** title, short, year, country, rank **FROM** Paper, Paper_Author, Author, Institution, Conference **WHERE** Paper.id=Paper_Author.paper_id **AND** Paper_Author.author_id=Author.id **AND** Author.institution_id=Institution.id **AND** Paper.conference_id=Conference.id **AND** short="SIGMOD" **AND** year=2018 **AND** country="USA" **AND** rank≤10 | 157 | 1.82 |
| $D_{10}$ | Pub | JOIN | **SELECT** title, short, year, keyword, rank **FROM** Paper, Paper_Author, Author, Institution, Conference, Paper_Keyword **WHERE** Paper.id=Paper_Author.paper_id **AND** Paper_Author.author_id=Author.id **AND** Author.institution_id=Institution.id **AND** Paper.conference_id=Conference.id **AND** Paper.id=Paper_Keyword.paper_id **AND** (keyword="data visualization" **OR** (short="SIGMOD" **AND** year=2018)) | 3978 | 25.28 |

TABLE IV
TESTED IDE-DECISION PROBLEMS

| $R$ | Dataset | Type | SQL |
|---|---|---|---|
| $R_1$ | Car | Hierarchical Sorting | **ORDER BY** year, -kilometer, -price, powerPS **DESC** |
| $R_2$ | Car | Weighted Sum Sorting | **ORDER BY** -0.018∗price+0.06∗year+0.982∗powerPS-0.001∗kilometer **DESC** |
| $R_3$ | Car | Hierarchical Weighted Sorting | **ORDER BY** notRepairedDamage, vehicleType, fuelType, brand, -0.018 ∗ price + 0.982 ∗ powerPS **DESC** |
| $R_4$ | Car | Hierarchical Weighted Sorting | **ORDER BY** year, -kilometer, -0.018 ∗ price + 0.982 ∗ powerPS **DESC** |
| $R_5$ | Pub | Weighted Sum Sorting | **ORDER BY** -0.8 ∗ rank + 0.2 ∗ year **DESC** |

TABLE V
TESTED IDE-RANKING PROBLEMS

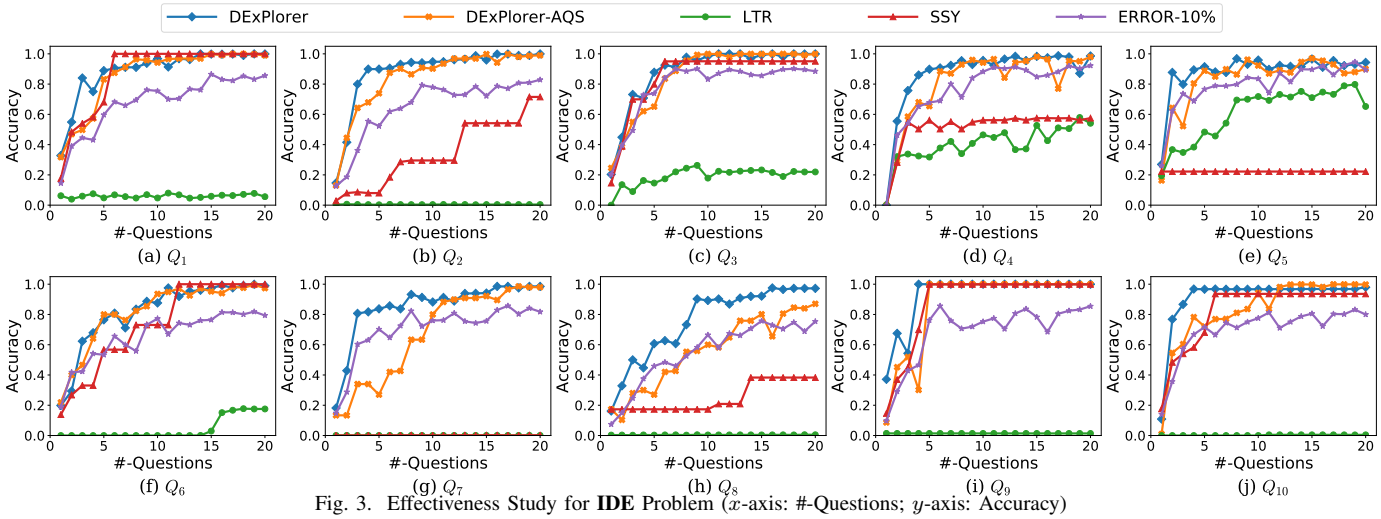



Fig. 3. Effectiveness Study for **IDE** Problem ($x$-axis: #-Questions; $y$-axis: Accuracy)

| Dataset | | Car | | | | | | | | Pub | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Qid | | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ | $Q_9$ | $Q_{10}$ |
| DExPlorer | #-Clicking | 120 | 80 | 120 | 120 | 90 | 140 | 140 | 120 | 40 | 40 |
| | #-Dragging | 32 | 6 | 43 | 56 | 63 | 49 | 17 | 24 | 7 | 9 |
| | Accuracy | 0.97 | 0.94 | 0.99 | 0.96 | 0.94 | 0.96 | 0.94 | 0.93 | 0.99 | 0.97 |
| DExPlorer-SEP | #-Clicking | 120 | 80 | 120 | 120 | 90 | 140 | 140 | 120 | 40 | 40 |
| | #-Dragging | 35 | 14 | 45 | 62 | 70 | 56 | 23 | 28 | 12 | 12 |
| | Accuracy | 0.88 | 0.84 | 0.80 | 0.74 | 0.71 | 0.79 | 0.69 | 0.71 | 0.88 | 0.83 |

TABLE VI
TREATING IDE-DECISION AND IDE-RANKING HOLISTICALLY VS. SEPARATELY USING OUR PROPOSED MODELS

we ask 5 questions using DExPlorer, the *accuracy* of $Q_3$ is 0.88 while the *accuracy* of $Q_8$ is only 0.61.

(5) Wrong answers reduce the effectiveness, as expected. But even though 10% of the questions are answered incorrectly, DExPlorer still performs better than LTR and SSY in

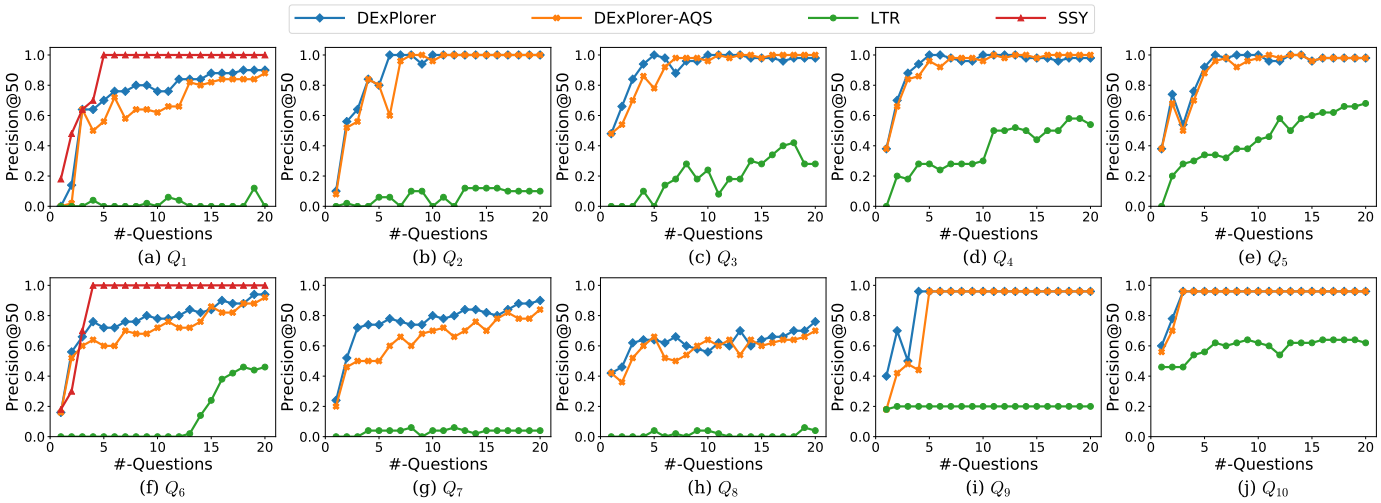



Fig. 4. Effectiveness Study for **IDE** Problem ($x$-axis: #-Questions; $y$-axis: Precision@50)

| Dataset | | Car | | | | | | | | Pub | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Qid | | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ | $Q_7$ | $Q_8$ | $Q_9$ | $Q_{10}$ |
| IQS | AI (sec) | 1.02 | 0.97 | 0.98 | 1.02 | 1.12 | 1.14 | 1.19 | 1.26 | 0.04 | 0.08 |
| | QS (sec) | 0.59 | 0.63 | 0.60 | 0.65 | 0.55 | 0.60 | 0.86 | 0.98 | 0.02 | 0.05 |
| AQS | AI (sec) | 1.03 | 1.04 | 0.88 | 1.01 | 1.05 | 1.12 | 1.18 | 1.20 | 0.05 | 0.08 |
| | QS (sec) | 4178.91 | 4081.05 | 4010.61 | 3969.71 | 3990.66 | 4215.86 | 4072.40 | 4326.84 | 5.57 | 18.61 |

TABLE VII
EFFICIENCY STUDY FOR **IDE** PROBLEM (AI: ANSWER INFERENCE; QS: QUESTION SELECTION)

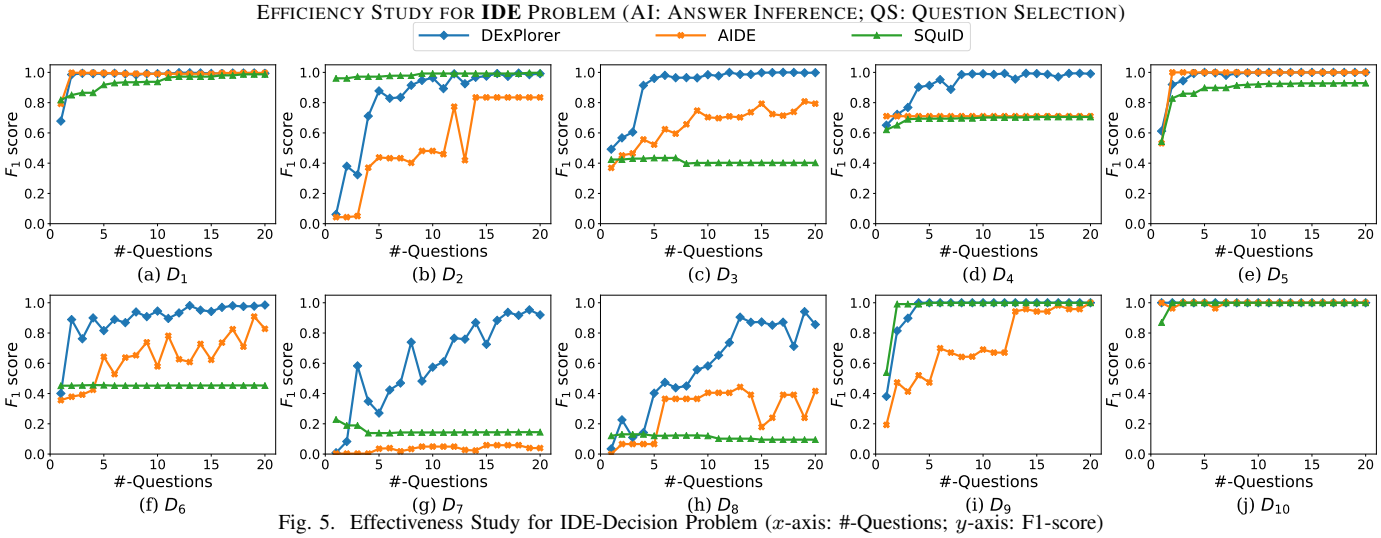



Fig. 5. Effectiveness Study for IDE-Decision Problem ($x$-axis: #-Questions; $y$-axis: F1-score)

average and can return reasonable results.

**Exp-2: Precision@l of IDE:** Figure 4 shows the *precision@50* of the IDE problems $Q_1 - Q_{10}$ by DEXPLORER, DEXPLORER-AQS, SSY, and LTR respectively. Since SSY only returns **ORDER BY** clauses for $Q_1$ and $Q_6$, we only report *precision@50* of SSY for $Q_1$ and $Q_6$. Figure 4 tells us that: DEXPLORER and DEXPLORER-AQS perform similarly, and they finally achieve 0.928 and 0.916 *precision@50* on average after asking 20 questions, while LTR has a poor performance for finding the top-$l$ tuples. SSY can only capture simple ranking clause of $Q_1$ and $Q_6$, so it only has good performance in $Q_1$ and $Q_6$.

**Exp-3: Treating IDE-Decision and IDE-Ranking separately (denoted by DEXPLORER-SEP) vs. holistically (denoted by DEXPLORER-SEP)** in each round of user interaction. We show the number of user operations in Table VI (*i.e.,* the number of "clicking" and "dragging") and *accuracy* of DEXPLORER when it gets converge (*i.e.,* the *accuracy* difference of two consecutive rounds is less than 1%). For DEXPLORER-SEP, we show users $k$ ($k = 10$) tuples in each round. In the first few rounds, users only perform click operations, and in the later rounds, users only perform drag operations. The question selection algorithm of DEXPLORER-SEP is shown in Section V. For ease of comparison, we perform the same number of "click" operations as DEXPLORER, then perform more (or same) number of "drag" operations than DEXPLORER, and report the *accuracy* of DEXPLORER-SEP. We can see that the *accuracy* of DEXPLORER is much higher than DEXPLORER-SEP: DEXPLORER averagely achieves 0.96 *accuracy* on 10 problems, while DEXPLORER-SEP averagely achieves 0.79 *accuracy*, that is, DEXPLORER outperforms DEXPLORER-SEP by 22%. This is because DEXPLORER-SEP only asks questions related to IDE-Decision problem in the first few rounds, without considering the IDE-Ranking problem, thus obtaining worse results.

*2) **Efficiency**:* **Exp-4: Efficiency:** We also test the efficiency of DEXPLORER on IDE problems $Q_1 - Q_{10}$. We repeat all experiments ten times to compute the average results. Table VII shows the running time of answer inference and question selection by algorithms IQS and AQS.

We make the following observations: (1) IQS significantly outperforms the baseline – AQS on all IDE problems (*i.e.,* $Q_1$ - $Q_{10}$). This observation also matches the result of the time complexity discussion in Section IV. (2) The results on answer inference and question selection of the tested algorithms do not vary a lot for different IDE problems on the same dataset. The reason is that the algorithms are independent of the complexity of the SQL queries. However, the complex SQL queries usually take more rounds to converge. (3) It takes less than 2 seconds on dataset `Car` to infer answers and select questions for the next round using IQS, which is feasible in practice. For the small dataset `Pub`, it only takes $\sim$0.1 second.

### B. Effectiveness of IDE-Decision Problem

**Metrics.** The IDE-Decision problem is actually a classification problem: the tuples in the inferred answer $R$ of the exploration system are classified as positive, and others as negative. Thus we use F1-score to evaluate the effectiveness of the IDE-Decision problem, which is defined as: $F1\text{-}score = \frac{2\times p\times r}{p+r}$, where $p = \frac{TP}{TP+FP}, r = \frac{TP}{TP+FN}, TP = |R \cap Q(T)|, FP = |R \cap\ Q'(T)|, FN = |R' \cap Q(T)|$.

**Comparisons.** We compare (1) DEXPLORER with: (2) AIDE [6] uses a decision tree for answer inference and selects $k$ tuples from diverse data areas to the user for question selection, and (3) SQUID [9] is a state-of-the-art SQL query intent discovery system, which takes a set of positive tuples as input examples and outputs the inferred SQL whose query result possibly includes these tuples. We select questions by randomly choosing $k$ positive tuples.

**Exp-5: Effectiveness on IDE-Decision:** We test DEXPLORER, AIDE, SQUID on $D_1 - D_{10}$. Figure 5 shows the results and tells us the followings:

(1) DEXPLORER outperforms AIDE and SQUID, among all 10 IDE-Decision problems. DEXPLORER averagely achieves 0.97 F1-score with 20 questions as training data, while AIDE and SQUID only averagely achieve 0.76 and 0.67 F1-score, respectively. In summary, DEXPLORER outperforms AIDE and SQUID by 28% and 45%, respectively.

(2) AIDE shows poor performance on complex queries (*i.e.,* $D_7, D_8$), as mentioned in Section III-B, which is due to the use of decision tree. What's more, AIDE only captures the diversity between selected tuples but without uncertainty, resulting in a poor performance.

(3) SQUID can only support the AND SQL queries. Thus it performs well in $D_1, D_2, D_9$. But for the OR, NO, MIXED SQL queries, its performance (F1-score) does not improve even with the increase of #-tuples for training.

### C. Effectiveness of IDE-Ranking Problem

**Metrics.** We also use *accuracy* defined in Section VI-A, with the only difference that all tuple pairs need to be ranked.

**Comparisons.** We compare (1) DEXPLORER with: (2) LearnPreference [26] uses Ranking SVM for answer inference and we apply our question selection algorithm (*i.e.,* Algorithm 1) because our hybrid ranking algorithm is essentially Ranking SVM, and LambdaMART is only used for feature enrichment, and (3) RankNet [1] is a learning-to-rank algorithm. We used RankNet for answer inference and do question selection by randomly selecting $k$ tuples for users to label.

**Exp-6: Effectiveness on IDE-Ranking:** Figure 6 shows the results of different approaches on $R_1 - R_5$, where $x$-axis denotes the number of questions answered by users, and $y$-axis denotes *accuracy*. We make the following observations:

(1) DEXPLORER outperforms LearnPreference and RankNet among all 5 IDE-Ranking problems, and they finally achieve 0.94, 0.93 and 0.88 *accuracy* finally. In summary, DEXPLORER outperforms LearnPreference and RankNet by 1.1% and 6.8%.

(2) DEXPLORER and LearnPreference finally achieve similar *accuracy* after several rounds of labeling, but DEXPLORER outperforms LearnPreference in the first few rounds. Hence, the LambdaMART algorithm can help to solve the cold start problem of Ranking SVM.

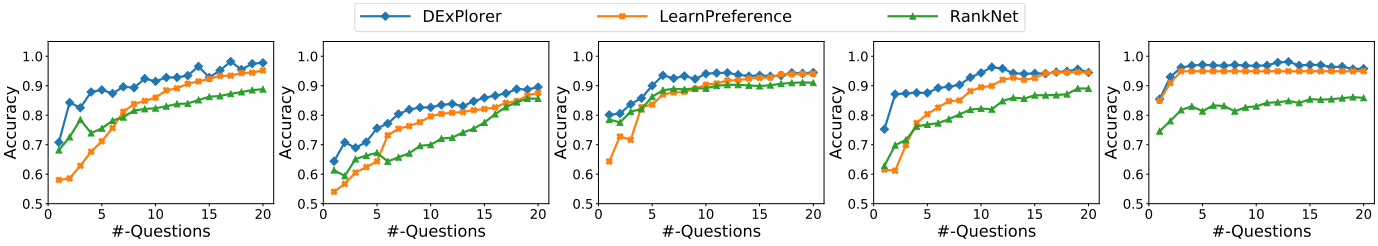
(3) RankNet behaves better than LearnPreference in the first few rounds, but it is likely to overfit with the number of labeled tuples increasing, such that the accuracy is lower than LearnPreference in the following rounds.

## VII. RELATED WORK

**Keyword-based tuple search** allows users to get interesting tuples by issuing keywords on relational databases without knowing the schema of databases. The language Reflective SQL [24] is an extension of SQL, which can be used to search on the relational databases as in a web search engine. DISCOVER [18] is a development of Reflective SQL which can support keyword queries on multiple tuples. There have also been many works [17], [23] on improving the efficiency and effectiveness of keyword search results.

DEXPLORER differs from keyword search in the following aspects: (1) It can refine the query results by further interaction with the system, while keyword search provides one-shot answers. (2) It can capture complex combination of predicates, which are hard to express using keywords. (3) It ranks tuples by users' hidden ranking intent, while keyword search ranks tuples by the relevance/proximity of keywords.

**Query-by-example.** QBE is a database query language [34] for relational databases, with the motivation that a parser can convert a user's actions into statements expressed in a database manipulation language, such as SQL. Discovering queries based on example tuples in a data warehouse has also been studied by Surajit et al. [28], which computes the minimal




(a) $R_1$ (b) $R_2$ (c) $R_3$ (d) $R_4$ (e) $R_5$

Fig. 6. Effectiveness Study for IDE-Ranking problem ($x$-axis: #-Questions; $y$-axis: Accuracy)

project join query that contains the given example tuples in its output. Exemplar queries [25] relax the requirement of evaluation engines to discover queries that produce results similar to given examples.

DExPlorer differs from QBE in many aspects: (1) It considers results to be ranked, but SQL results are typically not ranked. (2) It considers both good and bad examples. (3) It is designed for data browsing – requiring real-time response.

**Ranking database tuples.** There have been works on tuple ranking: Chaudhuri et al. [2] rank categorical SQL query results based on intuitions inferred from past workloads, which cannot support ranking on both categorical and numeric attributes. Qian et al. [26] and Xie et al. [32] rank tuples by computing a weighted sum score for each tuple, and the weight vector is learned from users' labeled partial orders. DExPlorer supports more general cases than the above approaches, as shown in Table I.

## VIII. Conclusion

We have built a database exploration system DExPlorer. We have implemented a user-friendly front-end that allows the user to select and rank tuples. On the back-end, we have developed a well-performed answer inference model, based on which we select a set of tuples to be answered by the user. We have proved that the optimal question selection problem is NP-hard and proposed an efficient and effective heuristic algorithm. We have also conducted extensive experiments to show the effectiveness of DExPlorer.

## References


[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.

[2] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[3] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[4] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD*, pages 395–406, 2006.

[5] P. Diaconis. Group representations in probability and statistics. *Ims Lecture Notesmonograph*, 72(2):7–108, 1988.

[6] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, pages 517–528, 2014.

[7] Dwork, Cynthia, Kumar, Ravi, Naor, Moni, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.

[8] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on discrete mathematics*, 17(1):134–160, 2003.

[9] A. Fariha and A. Meliou. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *PVLDB*, 12(11):1262–1275, 2019.

[10] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[11] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[12] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations research letters*, 21(3):133–137, 1997.

[13] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526. ACM, 2002.

[14] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. Applied machine learning at facebook: A datacenter infrastructure perspective. In *HPCA*, 2018.

[15] C. He, C. Wang, Y.-X. Zhong, and R.-F. Li. A survey on learning to rank. In *2008 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1734–1739. Ieee, 2008.

[16] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela. Practical lessons from predicting clicks on ads at facebook. In *ADKDD*, pages 5:1–5:9, 2014.

[17] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[18] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[19] E. Huang, L. Peng, L. D. Palma, A. Abdelkafi, A. Liu, and Y. Diao. Optimization for active learning-based interactive database exploration. *PVLDB*, 12(1):71–84, 2018.

[20] T. Joachims. Training linear svms in linear time. In *SIGKDD*, pages 217–226, 2006.

[21] M. Kahng, S. B. Navathe, J. T. Stasko, and D. H. P. Chau. Interactive browsing and navigation in relational databases. *PVLDB*, 9(12):1017–1028, 2016.

[22] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[23] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[24] U. Masermann and G. Vossen. Design and implementation of a novel approach to keyword searching in relational databases. In *Current Issues in databases and information systems*, pages 171–184. 2000.

[25] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.

[26] L. Qian, J. Gao, and H. Jagadish. Learning user preferences by adaptive pairwise comparison. *PVLDB*, 8(11):1322–1333, 2015.

[27] X. Qin, C. Chai, Y. Luo, N. Tang, and G. Li. Interactively discovering and ranking desired tuples without writing sql queries. In *SIGMOD*, pages 2745–2748, 2020.

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

[29] R. Singh, V. V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, and N. Tang. Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202, 2017.

[30] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[31] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

[32] M. Xie, T. Chen, and R. C.-W. Wong. Findyourfavorite: An interactive system for finding the user's favorite tuple in the database. In *SIGMOD*, pages 2017–2020, 2019.

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

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