Database Learning: Toward a Database that Becomes Smarter Every Time

Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, Barzan Mozafari
University of Michigan, Ann Arbor
{pyongjoo,tajik,michjc,mozafari}@umich.edu

ABSTRACT

In today's databases, previous query answers *rarely* benefit answering future queries. For the first time, to the best of our knowledge, we show how we can change this paradigm in an approximate query processing (AQP) context. We make the following observation: the answer to each query reveals some degree of *knowledge* about the answer to another query because their answers stem from the same underlying distribution that has produced the entire dataset. Exploiting and refining this underlying knowledge should allow us to answer queries more analytically, rather than by reading enormous amounts of raw data. Also, processing more queries should continuously enhance our knowledge of the underlying distribution, leading to faster processing of future queries.

We call this novel idea—learning from past query answers—*Database Learning*. We exploit *the principle of maximum entropy* to produce answers, which are in expectation guaranteed to be more accurate than existing sample-based approximations. Empowered by this idea, we build a query engine on top of Spark SQL, called INTELLI. We conduct extensive experiments on real-world query traces from a large customer of a major database vendor. Our results demonstrate that database learning supports 73.7% of these queries, speeding them up by upto 10.43x for the same accuracy level compared to existing AQP systems.

1. INTRODUCTION

In today's databases, the answer to a previous query is rarely useful for speeding up new queries. Besides a few limited benefits (see Previous Approaches below), the work (both I/O and computation) performed for answering past queries is often wasted afterwards. However, in an approximate query processing context (e.g., SnappyData [4], Presto [3], Druid [2], BlinkDB [9]), one might be able to change this paradigm altogether and re-use much of the previous work done by the database based on the following observation:

> *The answer to each query reveals some* fuzzy knowledge *about the answers to other queries, even if each query accesses a different subset of tuples and columns.*

This is because the answers to different queries stem from the same underlying distribution which has produced the entire dataset. In other words, **each answer reveals a piece of information about this underlying but unknown distribution**. Note that having a concise statistical model of the underlying data can have significant performance benefits. In the ideal case, if we had access to an incredibly precise model of the underlying data, we would no longer have to access the data itself. In other words, we could answer queries more efficiently by analytically evaluating them on our concise model, which would mean reading and manipulating a few kilobytes of model parameters rather than terabytes of raw data. While we may never have a perfect model in practice, even an imperfect model can be quite useful. Instead of using the entire data, one can use a small sample of it to quickly produce a sample-based approximate answer, which can be then calibrated and combined with the model to produce a more accurate approximate answer to the query. **The more precise our model, the less need for actual data, the smaller our sample, and consequently, the faster our response time.** In particular, if we could somehow continuously improve our model—say, by *learning* a bit of information from every query and its answer—we should be able to **answer new queries using increasingly smaller portions of data, i.e., become smarter and faster as we process more queries.**

We call the above goal *Database Learning* (DBL), as it is reminiscent of the inferential goal of machine leaning (ML), where past observations are used to improve future predictions [16, 17, 46]. Likewise, our goal in DBL is to enable a similar principle by **learning from past observations, but in a query processing setting**. Specifically, in DBL, we plan to treat approximate answers to past queries as observations, and use them to update our posterior knowledge of the underlying data, which in turn can be used to speed up future queries.

In Figure 1, we visualize this idea using a real-world Twitter dataset. Here, DBL learns a model for the number of occurrences of certain word patterns (known as *n-grams*, e.g., "bought a car") in tweets. Figure 1(a) shows this model based on the answers to the first two queries asking about the number of occurrences of these patterns, each over a different time range. Since the model is probabilistic, its 95% confidence interval is also shown (the shaded area around the best current estimate). As shown in Figures 1(b) and 1(c), DBL further refines its model every time a new query is answered. This approach allows a DBL-enabled query engine to provide increasingly more accurate estimates, *even for ranges that have never been accessed by previous queries*—this is possible because DBL finds the most likely model of the entire area that fits with the past query answers. This example is to illustrate the possibility of (i) significantly faster response times by processing smaller samples of the data for the same answer quality, or (ii) increasingly more accurate answers for the same sample size and response time.

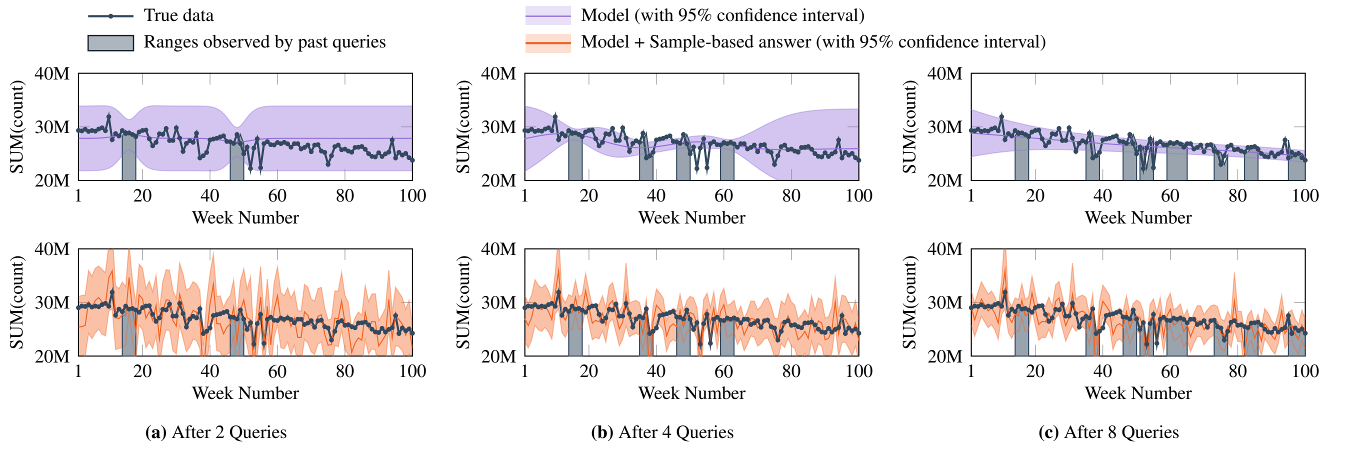


Figure 1: An example of how database learning might continuously refine its model as more queries are processed: (a) after processing 2 queries, (b) after 4 queries, and (c) after 8 queries. Its principle extends to categorical data as well. See that database learning can be effective even when its model does not exactly coincide with the ground-truth data.

Challenges— To realize DBL's vision, three key challenges must be overcome in practice. First, there is a *query generality* challenge. DBL must be able to transform a wide class of SQL queries into appropriate mathematical representations so that they can be fed into statistical models and used for improving the accuracies of new queries. Second, there is a *data generality* challenge. To support arbitrary datasets, no distributional assumptions must be made about the underlying data. In other words, the only valid knowledge must come from past queries and their respective answers. Finally, there is an *efficiency* challenge. We need to strike a balance between the computational complexity of our inference and its ability to reduce the error of query answers. In other words, DBL needs to be both effective and practical.

Previous Approaches— In today's databases, the work performed for answering past queries is rarely beneficial to new queries, except for the following cases:

1. **View selection/Adaptive indexing**: In predictable workloads, columns and expressions commonly used by past queries provide hints on which indices [25,29,43] or materialized views [10] to build.

2. **Caching**: The recently accessed tuples might still be in memory when future queries access the same tuples.

Both techniques, while beneficial, can only reuse previous work to a limited extent. Caching input tuples reduces I/O if data size exceeds memory, but does not reuse query-specific computations. Caching (intermediate) final results can reuse computation only if future (sub-)queries are *identical* to those in the past. While index selection techniques use the knowledge of which columns are commonly filtered on, an index per se does not allow for reusing computation from one query to the next. Adaptive indexing schemes (e.g., database cracking [29]) use each query to incrementally refine an index, to amortize the cost across queries. However, there is still an exponential number of possible column-sets that can be indexed. Also, they do not reuse query-specific computations either. Finally, materialized views[1] are only beneficial when there is a strict structural compatibility—such as query containment or equality—between past and new queries [26].

[1]**DBL can be easily misunderstood with view selection**. Not only do they take fundamentally different approaches (statistical versus exact), they also differ in generality and other aspects, as explained next.

While orthogonal to these existing techniques, DBL is fundamentally different:

1. Unlike indices and materialized views, DBL incurs *little storage overhead* as it only retains the past *n* aggregate queries and their answers.[2] Consequently, indices and materialized views grow in size as the data grows, while DBL remains *oblivious to the data size*.

2. Materialized views, indexing, and caching are exact methods and thus are only effective when new queries touch previously accessed columns or tuples. DBL is strictly *more general* as it can benefit new queries even if they require tuples that were not touched by past queries. This is due to DBL's probabilistic model, which provides extrapolation power spanning the entire data (see Figure 1).

Our Approach— Note that our vision of database learning (DBL) might be achieved in different ways, depending on the design decisions made in terms of query generality, data generality, and efficiency. In this paper, besides the introduction of the concept of DBL, we also provide a specific solution for achieving DBL, which we call INTELLI to distinguish it from DBL as a general vision.

From a high-level, INTELLI overcomes the challenges associated with query generality, data generality, and efficiency, as follows. First, complex SQL queries are decomposed into simpler *snippets*. The answer to each snippet is then modeled as a probabilistic random variable, corresponding to an integration over relevant tuples drawn from an unknown underlying distribution. Second, to achieve data generality, we employ a *non-parametric* probabilistic model, whereby answers to different snippets are related via a joint probability distribution function (pdf). We derive this joint pdf by exploiting a powerful statistical principle, namely the *principle of maximum entropy* [51], which yields the *most likely* pdf given our limited statistical knowledge (coming from past queries and their approximate answers). Third, to ensure the computational efficiency of our system, we restrict ourselves to only the first and second-order statistics of the query answers (i.e., mean, variance, and covariance) when applying the principle of maximum entropy. We show that this instantiation of DBL leads to significant speedup of query processing.

[2]Even if a query outputs too many tuples, DBL retains only a fixed number of them (see §2.3)

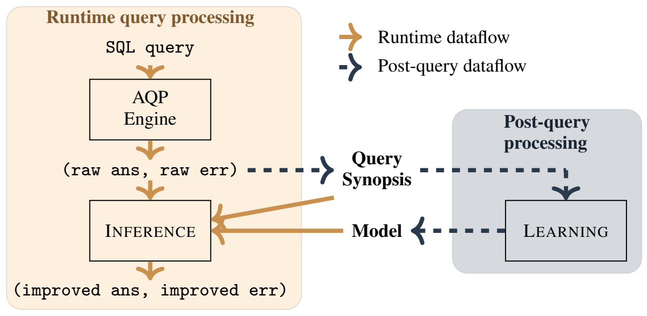


Figure 2: Workflow in INTELLI. At query time, the INFERENCE module improves a query answer obtained by an underlying approximate query processing (AQP) engine (i.e., *raw answer*) using a *Query Synopsis* and a *Model*. Each time a query is processed, the raw query answer, `(raw ans, raw err)`, is added to the query synopsis. The LEARNING module uses this updated query synopsis to refine the current model accordingly.

Contributions— This paper makes the following contributions:

1. We introduce the novel concept of *database learning* (DBL)—by learning from past query answers, DBL allows DBMS to continuously become smarter and faster at answering new queries.

2. We provide a concrete instantiation of DBL, called INTELLI, using the principle of maximum entropy and a *kernel*-based statistical modeling technique. INTELLI's strategies cover 63.6% of TPC-H queries and 73.7% of a real-world query trace from a leading vendor of analytical DBMS. We formally show that the estimated errors of INTELLI's answers are never worse than the estimated errors of existing AQP techniques.

3. We integrate INTELLI atop an open-source query engine and conduct extensive experiments using both benchmark and real-world traces, showing up to 10.43x speedup and 98% error reduction compared to existing AQP engines that do not use INTELLI.

The rest of this paper is organized as follows. §2 overviews INTELLI's workflow, supported query types, and query processing. §3-§5 describe the internals of INTELLI in detail. §6 reports our empirical results. §7 summarizes related work, and §8 concludes the paper with future work.

2. INTELLI OVERVIEW

In this section, we overview the system we have built based on database learning (DBL), called INTELLI. §2.1 explains INTELLI's architecture and overall workflow. §2.2 describes supported SQL query types. §2.3 and §2.4 overview INTELLI's query representation and computation, respectively. §2.5 presents the deployment scenarios of INTELLI. Lastly, §2.6 discusses INTELLI's current limitations.

2.1 Architecture

INTELLI consists of a *query synopsis*, a *model*, and three processing modules: an off-the-shelf approximate query processing (AQP) engine, an INFERENCE module, and a LEARNING module. Figure 2 depicts the connection between these components.

The *query synopsis* contains a summary of past queries[3] and their approximate answers computed by the underlying AQP engine. The query synopsis is initially empty when INTELLI is first launched. Once the i-th supported[4] query is processed, INTELLI adds a triplet $(q_i, \tilde{\theta}, \beta_i)$ to the query synopsis, where q_i is the i-th query, $\tilde{\theta}_i$ is an (approximate) answer to q_i, and β_i is an estimated error (§3.1) for $\tilde{\theta}_i$. $\tilde{\theta}_i$ and β_i are obtained from the underlying AQP engine. The queries stored in the query synopsis are called *past queries*.

The second key component is a *model* representing INTELLI's statistical understanding of the underlying data. The model is trained on the *query synopsis*, and is updated each time a query is added to the synopsis (§3 and §4).

At query time, for an incoming query (which we call a *new query*), INTELLI invokes the AQP engine to compute a pair of an approximate answer $\tilde{\theta}_i$ and an estimated error β_i for the new query, called the *raw answer* and the *raw error*, respectively. Then, INTELLI combines this raw answer and the previously trained model to *infer* an *improved answer* and an updated estimated error, called the *improved error*. Note that the improved error is never larger than the raw error in expectation (Theorem 1).

Lastly, INTELLI does not modify non-aggregate expressions or unsupported queries. Table 1 summarizes the terminology.

Term	Definition
true answer	exact answer
raw answer	answer computed by the AQP engine
raw error	estimated error for raw answer (by AQP)
improved answer	answer updated by INTELLI
improved error	estimated error for improved answer (by INTELLI)
past query	supported query processed in the past
new query	incoming query whose answer is to be computed

Table 1: Terminology.

2.2 Supported Queries

INTELLI supports aggregate queries that are flat (i.e., no derived tables or sub-queries) with the following conditions:

1. **Aggregates**. Any number of `SUM`, `COUNT`, or `AVG` aggregates can appear in the `select` clause. The arguments to these aggregates can also be a *derived attribute*.

2. **Joins**. Foreign-key joins between a fact table[5] and any number of dimension tables are supported. For simplicity, our discussion in this paper is based on a denormalized table.

3. **Selections**. INTELLI currently supports equality and inequality comparisons for categorical and numerical attributes (including the `in` operator). Currently, INTELLI does not support any disjunctions or textual filters (e.g., `like '%Apple%'`) in the `where` clause.

4. **Grouping**. `groupby` clauses are supported for both stored and derived attributes. The query may also include a `having` clause.[6]

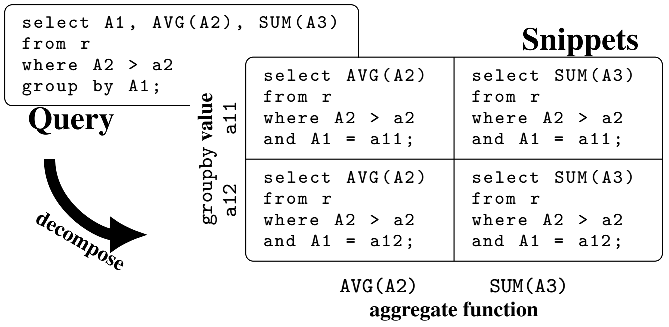
Nested Query Support— Although INTELLI does not directly support nested queries; many queries can be flattened using joins [1] or by creating intermediate views for sub-queries [26]. In fact, this is the process used by Hive for supporting the nested queries in

[3]DBL is focused on analytical (i.e., aggregate) queries only. Thus, we use 'analytical queries' and 'queries' interchangeably in this paper.

[4]Supported queries are defined in §2.2.

[5]In a data warehouse setting, measurements (e.g., sales record) are recorded into fact tables, and commonly appearing dimension attributes (e.g., seller information) are normalized into dimension tables [50].

[6]Note that the underlying AQP engine may affect the cardinality of the result set depending on the `having` clause (i.e., subset/superset error [38]). INTELLI simply operates on the result set returned by the AQP engine.



```
select A1, AVG(A2), SUM(A3)
from r
where A2 > a2
group by A1;
```

```
select AVG(A2)
from r
where A2 > a2
and A1 = a11;
```

```
select SUM(A3)
from r
where A2 > a2
and A1 = a11;
```

```
select AVG(A2)
from r
where A2 > a2
and A1 = a12;
```

```
select SUM(A3)
from r
where A2 > a2
and A1 = a12;
```

Figure 3: Example of a query's decomposition into multiple snippets.

TPC-H benchmark [31]. We are currently working to automatically process nested queries and to expand the class of supported queries (see §8).

Unsupported Queries— Upon its arrival, each query is inspected by INTELLI's query type checker to determine whether it can be supported, and if not, INTELLI bypasses the INFERENCE module and simply returns the raw answer. The overhead of the query type checker is negligible (§6.8) compared to the runtime of the AQP engine; thus, INTELLI does not incur any noticeable runtime overhead even when a query is not supported.

Only supported queries are stored in INTELLI's query synopsis and used to improve the accuracy of answers to future supported queries; that is, the class of queries that can be improved is equivalent to the class of queries that can be used to improve other queries.

2.3 Internal Representation

Query Synopsis— When a query (along with its raw answer and raw error) is inserted into the query synopsis, it is broken into multiple individual records. We call each of those records a *snippet*. Conceptually, each snippet corresponds to a supported SQL query with a single aggregate function and no other projected columns in its `select` clause, and with no `groupby` clause; thus, the answer to each snippet is a single real number. A SQL query with multiple aggregate functions or a `groupby` clause is converted to a set of multiple snippets for all combinations of each aggregate function and each `groupby` column value. As shown in the example of Figure 3, each `groupby` column value is added as an equality predicate in the `where` clause. The number of generated snippets can be extremely large, e.g., if a `groupby` clause includes a primary key. To ensure that the number of snippets added per each query is bounded, INTELLI only generates snippets for N^{max} groups in the answer set.[7]

For each aggregate function g, the query synopsis retains a maximum of C_g snippets by following a least recently used snippet replacement policy (by default, C_g=1000). This improves the efficiency of the inference process, while maintaining an accurate model based on the recently processed query answers.

Since INTELLI's inference and learning processes work with snippets, for ease of presentation we use *query* to mean *snippet* in the rest of this paper.

Aggregate Computation— INTELLI uses two aggregate functions to perform its internal computations: `AVG`(A_k) and `FREQ(*)`. As stated earlier, the attribute A_k can be either a stored attribute (e.g.,

[7]Currently, INTELLI simply chooses the first N^{max} (500 by default) groups in the output relation.

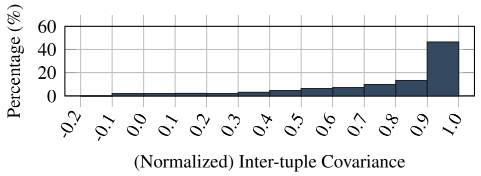


Figure 4: *Inter-tuple Covariances* for 16 real-life UCI datasets.

`revenue`) or a derived one (e.g., `revenue * discount`). At runtime, INTELLI combines these two types of aggregates to compute its supported aggregate functions as follows:

- `AVG`(A_k) = `AVG`(A_k)
- `COUNT(*)` = `FREQ(*)` × (table cardinality)
- `SUM`(A_k) = `AVG`(A_k) × `COUNT(*)`

2.4 Inference Intuition

In this section, we provide the high level intuition behind INTELLI's approach to improving the quality of new query answers. INTELLI exploits potential correlations between query answers to infer an answer to a new query. Conceptually, let S_i and S_j be multisets of attribute values such that, when aggregated, output exact answers to queries q_i and q_j, respectively. Then, the answers to q_i and q_j should be correlated, if:

1. **S_i and S_j include common values.** $S_i \cap S_j \neq \phi$ would imply the existence of correlation between the two query answers. For instance, computing the average revenue of the years 2014 and 2015 and the average revenue of the years 2015 and 2016 will be correlated since these averages include some common values (here, the 2015 revenue). In the TPC-H benchmark, 12 out of the 14 supported queries share common values in their aggregations.

2. **S_i and S_j include correlated values.** For instance, the average prices of a stock over two consecutive days are likely to be similar even though they do not share common values. When the compared days are farther apart, the similarity in their average stock prices might be lower. INTELLI captures the likelihood of such attribute value similarities using a statistical measure called *inter-tuple covariance*, which will be formally defined in §4.2. In the presence of non-zero inter-tuple covariances, the answers to q_i and q_j could be correlated even when $S_i \cap S_j \neq \phi$. It is common in many datasets to have non-zero inter-tuple covariances. Figure 4 reports the percentage of attributes with non-zero inter-tuple covariances across 16 real-life datasets used in our previous work [39] from the UCI repository [35]. We have observed similar results in Twitter data [11] and TPC-H benchmark.

2.5 Deployment Scenarios

INTELLI can be used with any AQP engine. However, to speed up queries as a middleware (see §2.1), INTELLI must be used with the following two types of AQP systems.

1. **AQP engines that support online aggregation** [27, 41, 54, 55]: Online aggregation continuously refines its approximate answer as new tuples are processed, until users are satisfied with the current accuracy or when the entire dataset is processed. In these types of engines, every time the online aggregation provides an updated answer (and error estimate), INTELLI generates an improved answer with a higher accuracy. As soon as this accuracy meets the user requirement,

the online aggregation can be stopped. In other words, the online aggregation's continuous processing will stop earlier than it would without INTELLI.

2. **AQP engines that support error-bounds**: These engines either use closed forms [9, 18, 19, 22, 28, 33] or analytical bootstrap [57] error estimation to predict the minimum sample size that would meet the user's accuracy requirement. For these engines, INTELLI will simply replace the user's original error bound e_1 with a larger value e_2 before passing it down to the underlying AQP engine. This is possible because, as we will show in equation (16) of §5, INTELLI can compute e_2 even before seeing the approximate answer and error from the underlying engine.

For AQP engines that use black-box implementation of bootstrap (e.g., [8, 44]), a tighter integration of INTELLI and the AQP engine is needed. In other words, INTELLI cannot be used as a middleware for such engines.

2.6 Limitations

Since INTELLI uses the principle of maximum entropy (ME) for inferring its improved answers, its accuracy benefits are dependent on the validity of the ME principle and the effectiveness of the statistical model derived from it. Although discussing the validity of the ME principle in general is beyond our scope, we empirically validate it for our use cases (§6.5). Our statistical model derived from the ME principle exploits possible correlations between query answers; thus, if a new query answer is uncorrelated with all previous quers, INTELLI will not improve it (INTELLI will not worsen it either as shown in Theorem 1)

Second, INTELLI relies on an off-the-shelf underlying AQP engine for obtaining raw answers and raw errors. Consequently, INTELLI is bound by the limitations of the AQP engine. For example, sample-based engines are not apt at supporting arbitrary joins or `MIN`/`MAX` aggregates. Similarly, INTELLI's confidence interval guarantees are contingent upon the validity of the AQP engine's raw errors. [8]

3. INFERENCE

In this section, we describe our inference process for computing an improved answer (and improved error) for a new query. First, we formally state our problem in §3.1 and its mathematical translation in §3.2. To solve this problem, we apply the principle of maximum entropy to capture the relationship among query answers using a joint probability distribution (§3.3). Then, we exploit the answers to past queries to conditionalize this probability distribution and infer improved answers to new queries (§3.4).

3.1 Problem Statement

Let r be a relation[9] drawn from some unknown underlying distribution. Let r's attributes be $A_1, \ldots, A_m$, where $A_1, \ldots, A_l$ are the *dimension attributes* and $A_{l+1}, \ldots, A_m$ are the *measure attributes*. Dimension attributes cannot appear inside aggregate functions while measure attributes can. Dimension attributes can be numerical or categorical, but measure attributes are numerical. Measure attributes can also be *derived attributes*.

Given a query q_i on r, an AQP engine returns an approximate answer $\tilde{\theta}_i$ along with an associated estimated error β_i. Let $\bar{\theta}_i^*$ be the (unknown) true answer to q_i, then $\beta_i^2 = E[(\tilde{\theta}_i - \bar{\theta}_i^*)^2]$ (rigorous definition in §3.2). β_i is independent of $\bar{\theta}_i^*$; also, β_i and β_j are uncorrelated for $i \neq j$.[10] Without loss of generality, we assume all queries have the same aggregate function g on A_k (e.g., `AVG(`A_k`)`), where A_k is one of the measure attributes.

[8]However, off-the-shelf error diagnostic techniques can be used [8].

[9]r can be a join or Cartesian product of multiple tables.

Sym.	Meaning
q_i	i-th (supported) query to database learning
$n+1$	index number for a new query
$\boldsymbol{\theta}_i$	random variable for possible raw answers to q_i
$\tilde{\theta}_i$	(actual) raw answer to q_i
β_i	estimated error associated with $\tilde{\theta}_i$
$\bar{\boldsymbol{\theta}}_i$	random variable for possible true answers to q_i

Table 2: Mathematical Notations.

Suppose INTELLI's query synopsis contains n records, namely $Q_n = \{(q_1, \tilde{\theta}_1, \beta_1), \ldots, (q_n, \tilde{\theta}_n, \beta_n)\}$. Let q_{n+1} be the *new query*. First, the AQP engine computes a raw answer and its raw error for q_{n+1}, as $(\tilde{\theta}_{n+1}, \beta_{n+1})$. Then, INTELLI computes an improved answer $\hat{\theta}_{n+1}$ and its improved error $\hat{\beta}_{n+1}$ using Q_n and $(\tilde{\theta}_{n+1}, \beta_{n+1})$. With this notation, our problem is stated as follows. Given Q_n and $(\tilde{\theta}_{n+1}, \beta_{n+1})$, compute a $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$ such that $\hat{\beta}_{n+1} \ll \beta_{n+1}$.

3.2 Mathematical Interpretation

In this section, we provide our random variable interpretation of query answers and a high-level overview of INTELLI's inference process.

Using Bayesian probability, we treat the (unknown) tuples in r as random variables. We can then encode the possible values of the raw and true answers as random variables $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_i$, respectively, since they are unknown to us. Then, the estimated error β_i associated with a raw answer $\tilde{\theta}_i$ is formally defined as $\beta_i^2 = E[(\tilde{\theta}_i - \bar{\boldsymbol{\theta}}_i)^2]$.[11] Based on this random variable interpretation, we can restate our problem as follows: find the most likely value for $\bar{\boldsymbol{\theta}}_{n+1}$ given the available assigned values, i.e., $\boldsymbol{\theta}_i = \tilde{\theta}_i$ for $i = 1, \ldots, n+1$.

To solve this problem, INTELLI first expresses the relationship among those random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$ using a joint probability distribution function (pdf), i.e., $f(\theta_1, \ldots, \theta_{n+1}, \bar{\theta}_{n+1})$. The symbols θ_i and $\bar{\theta}_i$ (not in bold) denote the particular values assigned to the random variables for the raw and true answers, $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_i$, respectively. Intuitively, this joint pdf encodes INTELLI's *prior belief* over the chance that a particular combination of values, i.e., $\theta_1, \ldots, \theta_{n+1}, \bar{\theta}_{n+1}$, are assigned to their corresponding random variables. Determining this joint pdf is a challenging task given that we cannot directly inspect the tuples in r. INTELLI overcomes this challenge by applying the principle of maximum entropy, assuming certain statistics are provided (§3.3). The best values for those statistics are estimated from past queries and their answers (§4.3).

Once the joint pdf is determined, the most likely value for $\bar{\boldsymbol{\theta}}_{n+1}$ given past query answers is estimated by computing a conditional pdf, i.e., $f_{\bar{\boldsymbol{\theta}}_{n+1}}(\bar{\theta}_{n+1} \mid \boldsymbol{\theta}_1 = \tilde{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1} = \tilde{\theta}_{n+1})$, and then finding the value of $\bar{\theta}_{n+1}$ at which the conditional pdf is maximized. INTELLI returns this value as an improved answer. This inference process will be presented in more detail in the following sections.

3.3 Prior Belief on Query Answers

In this section, we describes how INTELLI obtains its prior belief over possible query answers using a joint pdf $f(\theta_1, \ldots, \theta_{n+1}.\bar{\theta}_{n+1})$. Here, we assume certain query statistics (means, covariances, and

[10]This condition is satisfied if a new random sample is used for each query [34, 52]. Otherwise, we need minor modifications in equation (7).

[11]Technically, $\beta_i^2 = E[(\tilde{\theta}_i - \bar{\boldsymbol{\theta}}_i \mid \tilde{\theta}_i)^2]$ where $\bar{\boldsymbol{\theta}}_i \mid \tilde{\theta}_i$ indicates our belief on $\bar{\boldsymbol{\theta}}_i$ given $\tilde{\theta}_i$. However, we use $\bar{\boldsymbol{\theta}}_i$ instead of $\bar{\boldsymbol{\theta}}_i \mid \tilde{\theta}_i$ when its meaning is clear from the context.

variances among query answers) are available, deferring their computation to §4.

To obtain this joint pdf, INTELLI relies on the principle of maximum entropy (ME) [14, 51], a simple but powerful statistical tool for determining a pdf of random variables given a certain amount of statistical information available.[12] The ME principle states that, subject to some testable information on random variables associated with a pdf in question, the pdf that best represents the current state of our knowledge is the one that maximizes the following expression, called *entropy*:

$$h(f) = -\int f(\vec{\theta}) \cdot \log f(\vec{\theta}) \, d\vec{\theta} \tag{1}$$

where $\vec{\theta} = (\theta_1, \ldots, \theta_{n+1}, \bar{\theta}_{n+1})$.

Note that, according to the principle of ME, different amounts of statistical information on our random variables result in different pdfs. In fact, there are two conflicting considerations when applying this principle. On one hand, the resulting pdf can be computed more efficiently if the provided statistics are simple or few, i.e., simple statistics reduce the computational complexity. On the other hand, the resulting pdf can describe the relationship among the random variables more accurately if richer statistics are provided, i.e., the richer the statistics, the better our improved answers. Therefore, we need to choose an appropriate degree of statistical information to strike a balance between the computational efficiency of pdf evaluation and its accuracy in describing the relationship among query answers.

To strike this balance, INTELLI's INFERENCE module relies only on the first and the second order statistics of the random variables, i.e., mean, variances, and covariances. Then, a well-known result in information theory [51] guarantees that the resulting pdf is a multivariate normal distribution with the corresponding mean, variance, and covariance values.

Lemma 1. Let $\vec{\boldsymbol{\theta}} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$ be a vector of n+2 random variables with mean values $\vec{\mu} = (\mu_1, \ldots, \mu_{n+1}, \bar{\mu}_{n+1})$ and a $(n+2)\times(n+2)$ covariance matrix Σ specifying their variances and pairwise covariances. The only pdf f over these random variables that maximizes $h(f)$ while satisfying the provided means, variances, and covariances is the following function:

$$f(\vec{\theta}) = \frac{1}{\sqrt{(2\pi)^{n+2}|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{\theta}-\vec{\mu})^T \Sigma^{-1} (\vec{\theta}-\vec{\mu})\right). \tag{2}$$

Note that the above equation also provides a way for computing the likelihood of a certain combination of answers $\vec{\theta}$ given some statistics ($\vec{\mu}$ and Σ) on query answers. This property is used in §4.3 for finding the most likely statistics given the observed answers to past queries. Later, in §3.5, we discuss why obtaining $\vec{\mu}$ and Σ is itself a challenge. However, even if we somehow obtain $\vec{\mu}$ and Σ, the next question is how to use pdf (2) to obtain an improved answer to a new query.

3.4 Posterior Belief on Query Answers: Computing Improved Answers

In the previous section, we formalized the relationship among query answers, namely $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$, using a joint pdf. In this section, we exploit this joint pdf to infer an improved answer to q_{n+1}. In other words, we find the most likely value for $\bar{\boldsymbol{\theta}}_{n+1}$ (the random variable representing q_{n+1}'s true answer), given the observed values for $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$. Mathematically, INTELLI's improved answer $\hat{\theta}_{n+1}$ to q_{n+1} can be expressed as:

$$\hat{\theta}_{n+1} = \underset{\bar{\theta}_{n+1}}{\text{Arg Max}} \; f_{\bar{\boldsymbol{\theta}}_{n+1}}(\bar{\theta}_{n+1} \mid \boldsymbol{\theta}_1 = \tilde{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1} = \tilde{\theta}_{n+1}). \tag{3}$$

Fortunately, a well-known result shows that conditionalizing the pdf (2) results in another normal distribution, which allows us to solve optimization (3) analytically [15]:

$$\hat{\theta}_{n+1} = \bar{\mu}_{n+1} + k_{sub}^T \Sigma_{sub}^{-1} (\vec{\theta}_{sub} - \vec{\mu}_{sub}) \tag{4}$$

where:

- k_{sub} is a column vector of length $n+1$ whose i-th element is $(i, n+2)$-th entry of Σ;
- Σ_{sub} is a $(n+1)\times(n+1)$ submatrix of Σ consisting of Σ's first $n+1$ rows and columns;
- $\vec{\theta}_{sub} = (\tilde{\theta}_1, \ldots, \tilde{\theta}_{n+1})^T$; and $\vec{\mu}_{sub} = (\mu_1, \ldots, \mu_{n+1})$.

Likewise, the improved error $\hat{\beta}_{n+1}$ can be similarly obtained:

$$\hat{\beta}_{n+1}^2 = E[(\hat{\theta}_{n+1} - \bar{\boldsymbol{\theta}}_{n+1})^2] = \bar{\kappa}^2 - k_{sub}^T \Sigma_{sub}^{-1} k_{sub} \tag{5}$$

where $\bar{\kappa}^2$ is the $(n+2, n+2)$-th entry of Σ. Computing each of equations (4) and (5) requires $O(n^3)$ time complexity at query time. However, INTELLI uses alternative forms of these equations that require only $O(n^2)$ time complexity at query time (§5).

Note that, since the conditional pdf is a normal distribution, the error bound containing the true answer with probability δ (which we call δ-*confidence interval*) can be computed via multiplying $\hat{\beta}_{n+1}$ by a certain constant α_δ, where α_δ is a value such that a random number sampled from a standard normal distribution would fall within the $(-\alpha_\delta, \alpha_\delta)$ range with probability δ. We call α_δ the *confidence interval multiplier* for probability δ.

In the next section, we discuss several key challenges involved in using these equations.

3.5 Key Challenges in Prior Belief

As mentioned in §3.3, obtaining the joint pdf of in Lemma 1 (which represents INTELLI's prior belief on query answers) requires the knowledge of means, variances, and covariances of the random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$. However, acquiring these statistics is a non-trivial task for two reasons. First, we have only observed one value for each of the random values $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$, namely $\tilde{\theta}_1$, $\ldots$, $\tilde{\theta}_{n+1}$. Estimating variances and covariances of random variables from a single value is nearly impossible. Second, we do not have any observation for the last random variable, i.e., $\bar{\boldsymbol{\theta}}_{n+1}$. In §4, we present INTELLI's approach to solving these challenges.

4. APPROXIMATE PRIOR BELIEF

As described in §3, INTELLI expresses its prior belief on the relationship among query answers as a joint pdf over a set of random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$. In this process, we need to know the means, variances, and covariances of these random variables.

INTELLI uses the arithmetic mean of the past query answers for the mean of each random variable, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$. Note that this only serves as a prior belief, and will be updated in the process of conditioning the prior belief using the observed query answers. In this section, without loss of generality, we assume the mean of the past query answers is zero.

Thus, in the rest of this section, we focus on obtaining the variances and covariances of these random variables. In §4.1, we propose a decomposition of the (co)variances between pairs of query

[12]The ME principle has been also used for determining surprising information [47].

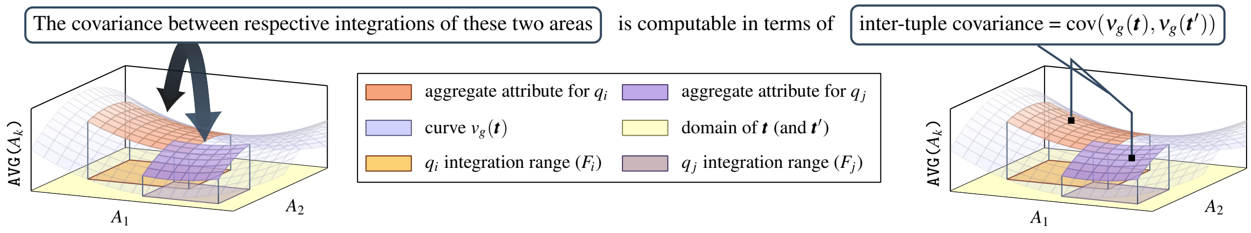

The covariance between respective integrations of these two areas
is computable in terms of
inter-tuple covariance = cov($\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}')$)
aggregate attribute for q_i
aggregate attribute for q_j
curve $\nu_g(\boldsymbol{t})$
domain of $\boldsymbol{t}$ (and $\boldsymbol{t}'$)
q_i integration range (F_i)
q_j integration range (F_j)
AVG(A_k)
A_1
A_2

Figure 5: The covariance between the answers to a pair of queries q_i and q_j (i.e., the statistics we need for applying the ME principle) is equal to the double-integration of inter-tuple covariances over respective selection predicate ranges F_i and F_j, due to the linearity of covariance (equation 8).

answers into *inter-tuple covariance* terms. Then, in §4.2, we explain how inter-tuple covariances can be estimated analytically using parameterized functions. In §4.3, we discuss the problem of determining optimal parameters for those covariance functions. Lastly, in §4.4, we present a safeguard against potential overfitting.

4.1 Covariance Decomposition

To compute the variances and covariances between query answers (i.e., $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$), INTELLI relies on our proposed *inter-tuple covariances*, which capture the statistical properties of the underlying distribution. Before presenting the inter-tuple covariances, our discussion starts with the fact that the answer to a supported query can be mathematically represented in terms of the underlying distribution.

Let g be an aggregate function on attribute A_k, and $\boldsymbol{t} = (a_1, \ldots, a_l)$ be a vector of length l comprised of the values for r's dimension attributes $A_1, \ldots, A_l$. We define a function $\nu_g(\boldsymbol{t})$ for every aggregate function g (e.g., `AVG(`A_k`)`, `FREQ(*)`) such that, when integrated, it produces answers to queries. That is (omitting possible normalization and weight terms for simplicity; see appendix A):

$$\bar{\boldsymbol{\theta}}_i = \int_{\boldsymbol{t} \in F_i} \nu_g(\boldsymbol{t})\, d\boldsymbol{t} \tag{6}$$

where F_i is a set of ranges for which $\boldsymbol{t} \in F_i$ satisfies the selection predicates of q_i. Due to the independence between estimated errors and true query answers, we have:

$$\mathrm{cov}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) + \delta(i,j) \cdot \beta_i^2 \tag{7}$$

where $\delta(i,j)$ returns 1 if $i = j$ and 0 otherwise. Using (6) and the linearity of covariance, we can further decompose $\mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ into:

$$\begin{aligned} \mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) &= \mathrm{cov}\left(\int_{\boldsymbol{t} \in F_i} \nu_g(\boldsymbol{t})\, d\boldsymbol{t}, \int_{\boldsymbol{t}' \in F_j} \nu_g(\boldsymbol{t}')\, d\boldsymbol{t}' \right) \\ &= \int_{\boldsymbol{t} \in F_i} \int_{\boldsymbol{t}' \in F_j} \mathrm{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}'))\, d\boldsymbol{t}\, d\boldsymbol{t}' \end{aligned} \tag{8}$$

As a result, the covariance between query answers can be broken into an integration of the covariances between tuple-level function values, which we call *inter-tuple covariances*. Figure 5 illustrates this idea using two queries ranging over the orange and purple areas, respectively. For a detailed discussion of how to express the covariances between query answers in terms of inter-tuple covariances (and how to compute them efficiently), see appendix A.

To use (8), we must be able to compute the inter-tuple covariance terms. However, computing these inter-tuple covariances is challenging, as we only have a single observation for each $\nu_g(\boldsymbol{t})$. Moreover, even if we had a way to compute the inter-tuple covariance for arbitrary $\boldsymbol{t}$ and $\boldsymbol{t}'$, computing it for all possible pairs of tuples would be too costly and impractical. In the next section, we present an efficient alternative for estimating these inter-tuple covariances.

4.2 Analytic Inter-tuple Covariances

To efficiently estimate the inter-tuple covariances, and thereby compute equation (8), we propose using *analytical covariance functions*, a well-known technique in statistical literature for approximating covariances [15]. In particular, INTELLI uses squared exponential covariance functions, which are proven to be *universal* [37], i.e., capable of approximating any target function arbitrary closely as the number of observations (here, query answers) increases. The squared exponential covariance function $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ is defined as:

$$\mathrm{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}')) \approx \rho_g(\boldsymbol{t}, \boldsymbol{t}') = \sigma_g^2 \cdot \prod_{i=1}^{l} \exp\left(-\frac{\Delta(a_i, a_i')^2}{l_{g,i}^2} \right) \tag{9}$$

where

$$\Delta(a_i, a_i') = \begin{cases} |a_i - a_i'| & \text{if } A_i \text{ is numerical} \\ 1 - \delta(a_i, a_i') & \text{if } A_i \text{ is categorical.} \end{cases}$$

Here, $l_{g,i}$ for i=1…m and σ_g^2 are tunable *correlation parameters* to be learned from past queries and their answers (§4.3).

Intuitively, when $\boldsymbol{t}$ and $\boldsymbol{t}'$ are similar, i.e., $\Delta(a_i, a_i')$ is small for most A_i, then $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ returns a larger value (closer to σ_g^2), indicating that the expected values of g for $\boldsymbol{t}$ and $\boldsymbol{t}'$ are highly correlated. By double-integrating (9) over the ranges of the participating queries' selection predicates, INTELLI can compute the covariances between pairs of query answers (equation 8) analytically.

4.3 Optimal Correlation Parameters

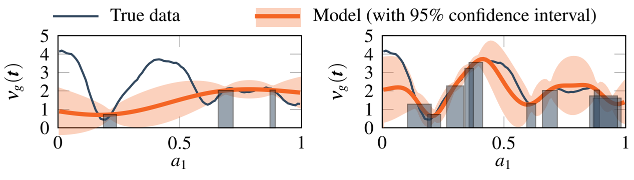
In this section, we describe how to find the most likely values for the correlation parameters defined in §4.2. Recall that the pdf (2) provides a way for computing the likelihood of a certain combination of query answers given relevant statistics. Here, we exploit the pdf (2) in searching for the most likely correlation parameters given past query answers. Let $\vec{\theta}_{\text{past}}$ denote a vector of raw answers to past queries. Then, by Bayes' theorem:

$$Pr(\Sigma \mid \vec{\theta}_{\text{past}}) \propto Pr(\Sigma) \cdot Pr(\vec{\theta}_{\text{past}} \mid \Sigma)$$

where $\propto$ indicates that the two values are proportional. Therefore, without any preference over parameter values, the problem of finding the most likely correlation parameters (which determine Σ) given past queries is equivalent to finding the values for $l_{g,1}, \ldots, l_{g,m}$, σ_g^2 that maximize the below log-likelihood function:

$$\begin{aligned} \log Pr(\vec{\theta}_{\text{past}} \mid \Sigma) &= \log f(\vec{\theta}_{\text{past}}) \\ &= -\frac{1}{2} \vec{\theta}_{\text{past}}^T \Sigma^{-1} \vec{\theta}_{\text{past}} - \frac{1}{2} \log |\Sigma| - \frac{n}{2} \log 2\pi \end{aligned} \tag{10}$$

where $f(\vec{\theta}_{\text{past}})$ is the joint pdf from (2), and Σ is the $n \times n$ covariance matrix whose (i,j)-th entry is the (co)variance between $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$. The values of $l_{g,1}$, …, $l_{g,m}$, and σ_g^2 that maximize the



(a) After three queries **(b)** After ten queries

Figure 6: An example of (a) overly optimistic confidence intervals due to overfitting, and (b) its resolution as additional queries are processed. INTELLI relies on equation (11) to avoid overfitting.

above equation can be obtained using standard gradient-descent algorithms. Although gradient-descent algorithms are typically slower than closed-form solutions, they do not pose a challenge in INTELLI's setting, since these parameters are computed *offline*, i.e., prior to the arrival of new queries. Lastly, our use of approximate answers as the constraints for the ME principle is properly accounted for by including additive error terms in their (co)variances (equation 7).

One caveat of using the above technique for choosing parameter values is the risk of overfitting, when the number of past observations (i.e., past query answers) is insufficient. We address this challenge next.

4.4 Overfitting Safeguard

Insufficient number of past queries may lead to parameter values that do not accurately reflect the characteristics of the underlying data, as demonstrated in Figure 6. Let θ'_{n+1} denote INTELLI's (approximate) answer to a new query *without* using the raw answer $\tilde{\theta}_{n+1}$ from the AQP engine.[13] Intuitively, θ'_{n+1} should be close to $\tilde{\theta}_{n+1}$ as they are estimating the answer to the same query q_{n+1}. Therefore, to detect and prevent overfitting, INTELLI uses the following before returning the final answer $\check{\theta}_{n+1}$ to the user:

$$\check{\theta}_{n+1} = \begin{cases} \hat{\theta}_{n+1} & \text{if } |\theta'_{n+1} - \tilde{\theta}_{n+1}| \le \alpha_\delta \cdot \beta_{n+1}/2 \\ \tilde{\theta}_{n+1} & \text{otherwise} \end{cases} \tag{11}$$

where $\hat{\theta}_{n+1}$ is the improved answer (equation 4), and α_δ the confidence interval multiplier for probability δ. In other words, INTELLI ignores its model-based answer if it deviates too much from the raw answer. This safeguard serves as a condition for ensuring the probabilistic correctness of INTELLI's approximate answers, as will be formally shown in §5.

5. FORMAL GUARANTEES

Let $(\hat{\theta}^*_{n+1}, \hat{\beta}^*_{n+1})$ be a pair of improved answer and improved error computed from the ME distribution based on true statistics (which are typically unknown). Also, let θ'_{n+1} be the answer obtained solely from INTELLI's model (§4.4) for the new query.

Theorem 1. Assume the probabilistic correctness of $(\hat{\theta}^*_{n+1}, \hat{\beta}^*_{n+1})$, namely:

$$Pr(|\hat{\theta}^*_{n+1} - \bar{\theta}^*_{n+1}| \le \alpha_\delta \cdot \hat{\beta}^*_{n+1}) \;=\; \delta.$$

where $\bar{\theta}^*_{n+1}$ is the (unknown) true answer, and α_δ is the confidence interval multiplier for probability δ. If the two conditions $\hat{\beta}^*_{n+1} \le \hat{\beta}_{n+1}$ and $|\theta'_{n+1} - \tilde{\theta}_{n+1}| \le \alpha_\delta \cdot \beta_{n+1}/2$ hold true, then INTELLI's answer $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$ satisfies the following:

$$\hat{\beta}_{n+1} \le \beta_{n+1} \tag{12}$$
$$Pr(|\hat{\theta}_{n+1} - \bar{\theta}^*_{n+1}| \le \alpha_\delta \cdot \hat{\beta}_{n+1}) \;\ge\; \delta \tag{13}$$

This theorem states that, as long as INTELLI's estimated errors are conservative, INTELLI's confidence intervals are both (i) probabilistically correct and (ii) never larger than those returned by the underlying AQP engine. This implies that INTELLI's actual errors are also smaller than the underlying AQP engine's actual errors *in expectation*, since the actual errors of both systems are bounded by their respective confidence intervals with the same probability.

Our empirical study (§6.5) shows that Theorem 1's assumption regarding the probabilistic correctness of the ME distribution based on true statistics and the condition $\hat{\beta}^*_{n+1} \le \hat{\beta}_{n+1}$ are satisfied in most test cases. The last condition $|\theta'_{n+1} - \tilde{\theta}_{n+1}| \le \alpha_\delta \cdot \beta_{n+1}/2$ is from our safeguard (§4.4). Although we do not have a formal guarantee for the case where $\hat{\beta}^*_{n+1} > \hat{\beta}_{n+1}$, we empirically show that INTELLI's probabilistic correctness hold true (§6.6). Next, we present the proof of the above theorem.

Proof. Besides showing (12) directly, we show (13) by proving $|\hat{\theta}_{n+1} - \hat{\theta}^*_{n+1}| \le \alpha_\delta(\hat{\beta}_{n+1} - \hat{\beta}^*_{n+1})$, which is a sufficient condition for (13). We show each of them after presenting common parts.

COMMON PARTS: Let Σ be the covariance matrix of the vector $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$; k_s be a column vector of length n whose i-th element is the $(i, n+1)$-th entry of Σ; Σ_s be an $n \times n$ submatrix of Σ that consists of Σ's first n rows/columns; $\bar{\kappa}^2$ be a scalar value at the $(n+2, n+2)$-th entry of Σ; and $\vec{\theta}_s$ be a column vector $(\tilde{\theta}_1, \ldots, \tilde{\theta}_n)^T$.

Then, we can express k_{sub} and Σ_{sub} of equation (4) in block forms as follows:

$$k_{sub} = \begin{pmatrix} k_s \\ \bar{\kappa}^2 \end{pmatrix}, \quad \Sigma_{sub} = \begin{pmatrix} \Sigma_s & k_s \\ k_s^T & \bar{\kappa}^2 + \beta_{n+1}^2 \end{pmatrix}, \quad \vec{\theta}_{sub} = \begin{pmatrix} \vec{\theta}_s \\ \tilde{\theta}_{n+1} \end{pmatrix}$$

Using the formula of block matrix inversion [30], we can obtain the following alternative forms (using zero means without loss of generality) from (4) and (5):

$$\gamma^2 = \bar{\kappa}^2 - k_s^T \Sigma_s^{-1} k_s, \qquad \theta'_{n+1} = k_s^T \Sigma_s^{-1} \vec{\theta}_s \tag{14}$$
$$\hat{\theta}_{n+1} = (\beta_{n+1}^2 \cdot \theta'_{n+1} + \gamma^2 \cdot \tilde{\theta}_{n+1})/(\beta_{n+1}^2 + \gamma^2) \tag{15}$$
$$\hat{\beta}_{n+1}^2 = \beta_{n+1}^2 \cdot \gamma^2/(\beta_{n+1}^2 + \gamma^2) \tag{16}$$

PROOF FOR (12): We can easily show $\hat{\beta}_{n+1}^2 - \beta_{n+1}^2 \le 0$ using (16); thus, the inequality follows.

PROOF FOR (13): Note that $\hat{\beta}_{n+1}^2$ is an increasing function of γ^2. Therefore, it suffices to show that

$$\partial(\hat{\theta}_{n+1} + \alpha_\delta \cdot \hat{\beta}_{n+1})/\partial\gamma^2 \ge 0, \tag{17}$$
$$\partial(\hat{\theta}_{n+1} - \alpha_\delta \cdot \hat{\beta}_{n+1})/\partial\gamma^2 \le 0 \tag{18}$$

since we can show that the second inequality holds for arbitrary γ (and $\hat{\beta}_{n+1}$) by integrating (17) and (18).

We can find that (17) holds if $|\theta'_{n+1} - \tilde{\theta}_{n+1}| \le \alpha_\delta \cdot \beta_{n+1}^2/2\hat{\beta}_{n+1}$ by computing the derivatives of (15) and (16). Observe that this inequality always holds since $|\theta'_{n+1} - \tilde{\theta}_{n+1}| \le \alpha_\delta \cdot \beta_{n+1}/2$ and $\hat{\beta}_{n+1} \le \beta_{n+1}$. We can show (18) similarly. □

Lemma 2. INTELLI's inference (i.e., equations 4 and 5) can be computed with $O(n^2)$ time complexity at query time. This is because equation (14) can be computed prior to the arrival of new queries, and (15) and (16) can be used in place of (4) and (5).

[13] Mathematically, this value can be obtained using equation (14).

6. EXPERIMENTS

Our experiments aim to (i) quantify the percentage of real-world queries that benefit from INTELLI and their average speedup (§6.2 and §6.3), (ii) study the benefits of INTELLI's correlation modeling (§6.4), (iii) test the reliability of INTELLI's error estimates (§6.6 and §6.5), (iv) study the impact of different workloads and data distributions on INTELLI's effectiveness (§6.7), and (v) measure INTELLI's computational overhead and memory footprint (§6.8).

In summary, our results indicate the following:

- INTELLI supports a large fraction (73.7%) of aggregate queries in a real-world workload, bringing significant speedups (up to 10.43x) compared to existing (sample-based) AQP solutions.
- Given the same processing time, INTELLI reduces the baseline's approximation error on average by 56–95%.
- INTELLI's run-time overhead is $<$10 milliseconds (0.02–0.48% of total time) and its memory footprint is negligible.
- INTELLI's approach is robust against different workloads and data distributions.

6.1 Experiment Setup

Datasets and Query Workloads— For our experiments, we used the three datasets described below:

1. `Customer1`: This is a real-world query trace from one of the largest customers (anonymized) of a leading vendor of analytic DBMS. This dataset contains 310 tables and 15.5K timestamped queries issued between March 2011 and April 2012, 3.3K of which queries are analytical queries supported by SparkSQL. We did not have the customer's original dataset but had access to their data distribution, which we used to generate a 536GB dataset.
2. `TPC-H`: This is a well-known analytical benchmark with 22 query types, 21 of which contain at least one aggregate function (including 2 queries with `min` or `max`). We used a scale factor of 100, i.e., the total data size was 100GB. We generated a total of 500 queries using `TPC-H`'s workload generator with its default settings.
3. `Synthetic`: For more controlled experiments, we also generated large-scale synthetic datasets with different distributions (see §6.4, §6.5, and §6.7 for details).

Implementation— For comparative analysis, we implemented three systems on top of SparkSQL [12] (ver 1.5.1):

1. NOLEARN: This system runs queries on *samples* of the original tables to obtain fast but approximate query answers and their associated estimated errors. This is the same approach taken by existing AQP engines, such as [4,5,9,19,44,45,56]. Specifically, NOLEARN maintains uniform random samples created offline (10% of the original tables), and uses the smallest samples that are large enough to satisfy the requested error bounds. For join queries, NOLEARN uses a sample only for the largest relation.
2. BASELINE2: This system is similar to NOLEARN but returns a cached answer if (i) the new query is identical to one of the past ones, and (ii) the new query's error requirement is looser than that of the past query. When there are multiple instances of the same query, BASELINE2 caches the one with the lowest estimated error.

Dataset	# Analyzed	# Supported	Percentage
`Customer1`	3,342	2,463	73.7%
`TPC-H`	21	14	63.6%

Table 3: Generality of INTELLI. INTELLI supports a large fraction of real-world and benchmark queries.

3. INTELLI: This system invokes NOLEARN to obtain raw answers/errors but modifies them to produce improved answers/errors using our proposed inference process. INTELLI translates the user's requested error into an appropriate error requirement for NOLEARN, using equation (16), so that INTELLI's improved error meets the user's requested error.

Experimental Environment— We used a Spark cluster (for both NOLEARN and INTELLI) using 5 Amazon EC2 `m4.2xlarge` instances, each with 2.4 GHz Intel Xeon E5 processors (8 cores) and 32GB of memory. Our cluster also included SSD-backed HDFS [48] for Spark's data loading. For experiments with cached datasets, we distributed Spark's RDDs evenly across the nodes using SparkSQL DataFrame `repartition` function.

6.2 Generality of INTELLI

To quantify the generality of our approach, we analyzed the real-world SQL queries in `Customer1`. From the original 15.5K queries, SparkSQL was only able to process 3.3K of the aggregate queries. Among those 3.3K queries, INTELLI supported 2.4K queries, i.e., 73.9% of the analytical queries could benefit from INTELLI. In addition, we analyzed the 21 `TPC-H` queries and found 14 queries supported by INTELLI. Others could not be supported due to textual filters or disjunctions in the `where` clause. These statistics are summarized in Table 3. This analysis proves that INTELLI can support a large class of analytical queries in practice. Next, we quantify how much these supported queries benefit from INTELLI.

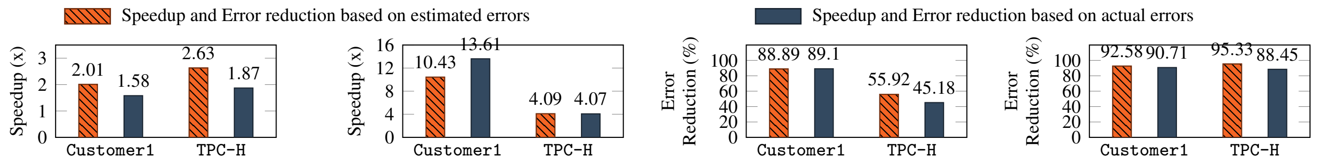
6.3 Query Speedup

In this section, we study INTELLI's query speedup over NOLEARN: how faster can queries achieve the same level of accuracy under INTELLI compared to NOLEARN? For this experiment, we used each of `Customer1` and `TPC-H` datasets in two different settings. In one setting, all samples were cached in the memories of the cluster, while in the second, SparkSQL had to read the data from SSD-backed HDFS.

We allowed both systems to process half of the queries (since `Customer1` queries were timestamped, we used the first half). While processing those queries, NOLEARN simply returned the query answers but INTELLI also learned its model parameters. Then, for the second half of the queries, we recorded both systems' query response times (i.e., latencies), approximate query answers, and estimated errors. For analysis, we also computed the actual errors. All reported errors in this paper are relative errors.

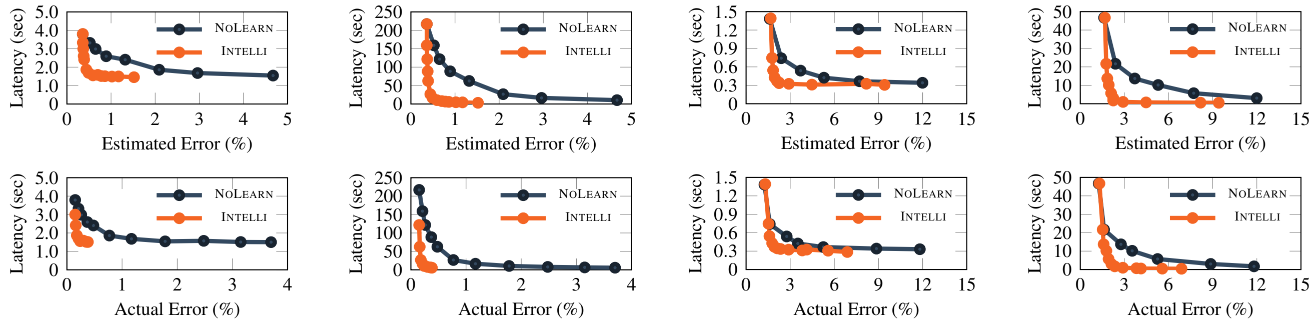
Figures 7(a-b) shows the average latencies of the two systems for different datasets and caching scenarios, for a target error of 2%. For `Customer1` queries on non-cached data, INTELLI delivered a 10.43x speedup on average (i.e., more than 90% reduction in query response time) for the same target error. This is because INTELLI could deliver highly-accurate answers using significantly smaller sample sizes by leveraging the inferred knowledge from past query answers. In all cases, INTELLI achieved at least 2x speedup over NOLEARN.

Figures 7(c-d) shows the average error reduction of INTELLI compared to NOLEARN, for a fixed time budget (i.e., target latency). In this experiment, the target latencies were 1.5 sec for



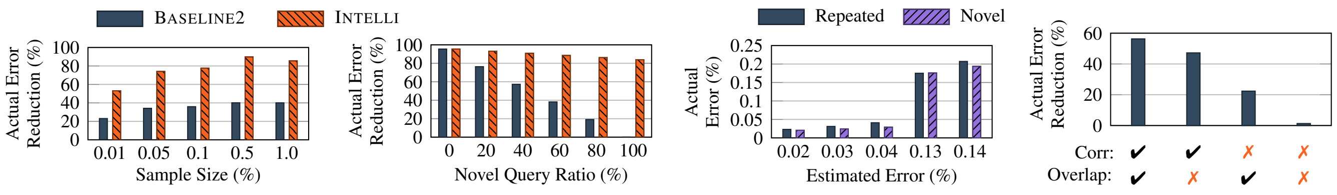
(a) Fully Cached **(b)** No Cache **(c)** Fully Cached **(d)** No Cache

Figure 7: Speedup of INTELLI over NOLEARN for a target error of 2% (a-b) and average error reduction of queries by INTELLI (compared to NOLEARN) for the same time budget (c-d).



(a) Fully Cached, `Customer1` **(b)** No Cache, `Customer1` **(c)** Fully Cached, `TPC-H` **(d)** No Cache, `TPC-H`

Figure 8: The trade-off between error and latency for NOLEARN and INTELLI: estimated error (top) and actual error (bottom).



(a) Sample Sizes for Past Queries **(b)** Query Workload Composition **(c)** Repeat vs. Novel **(d)** With and without correlation/overlap

Figure 9: (a) Comparison of INTELLI and BASELINE2 for different sample sizes used by past queries, (b) comparison of INTELLI and BASELINE2 for different ratios of novel queries in the workload, (c) the relationship between actual and estimated errors for repeated and novel queries, and (d) INTELLI's performance with and without inter-tuple correlations in the data, and with and without overlap between new and past queries.

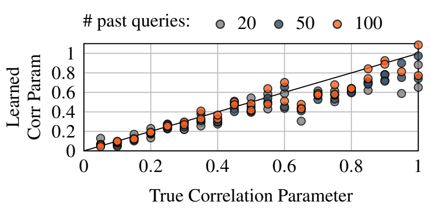
cached `Customer1`, 3.5 sec for non-cached `Customer1`, and 0.5 sec for both cached and non-cached `TPC-H`. (Other target latencies will be reported in Figure 8.) As shown in Figures 7(c-d), for the same time budget, INTELLI reduced NOLEARN's estimated error by at least 55.92% and up to 95.33%.

Figure 8 presents a detailed study of the trade-off between average query latencies and average errors in both systems. In all experiments, the latency-error graphs exhibit consistent patterns: (i) INTELLI produced smaller errors even when target latencies were very small, and (ii) INTELLI showed faster query response times for the same target errors. Due to the asymptotic nature of errors, achieving extremely accurate answers (e.g., less than 0.5%) required relatively large sample sizes (and processing times) even for INTELLI. Also, the reason that INTELLI's speedups were slightly lower for cached settings was that the default overhead of SparkSQL was relatively large compared to its overall data processing time due to the sample size. In other words, even if INTELLI reduced the query processing time to zero, it would not be possible to achieve more than 3–5x speedups due to SparkSQL's overhead of running an empty query. However, INTELLI still achieved 2.63x speedup even for fully-cached datasets.

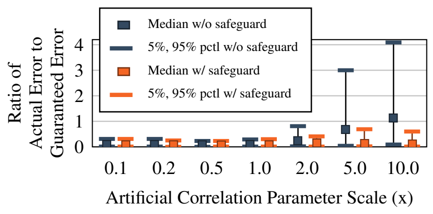
6.4 Benefits of Model-based Inference

To study the benefits of INTELLI's model-based inference, we compared the performance of INTELLI and BASELINE2, using the `TPC-H` dataset. Figure 9(a) reports the average actual error reductions of INTELLI and BASELINE2 (over NOLEARN), when different sample sizes were used for past queries. Here, the same samples were used for new queries. The result shows that both systems' error reductions were large when large sample sizes were used for the past queries. However, INTELLI consistently achieved higher error reductions compared to BASELINE2, due to its ability to benefit novel queries as well as repeated queries (i.e., the queries that have appeared in the past).

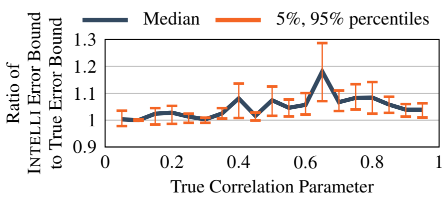
Figure 9(b) compares INTELLI and BASELINE2 by changing the ratio of novel queries in the workload. Understandably, both INTELLI and BASELINE2 were more effective for workloads with fewer novel queries (i.e., more repeated queries); however, INTELLI was also effective for workloads with many novel queries. Figure 9(c) further confirms that INTELLI did in fact benefit both novel and repeated queries, and their actual errors were low when the new queries were strongly correlated (i.e., low estimated errors).



(a) Correlation Param Learning

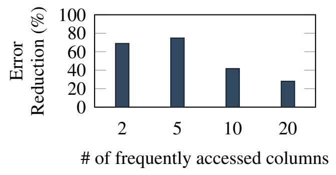


(b) Validity of ME Principle / Effect of Safeguard

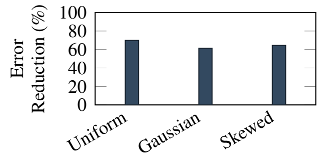


(c) INTELLI's Error Bound Size compared to True

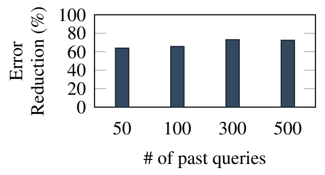
Figure 10: (a) The accuracy of INTELLI's correlation parameter learning, (b) the validity of the ME principle (Theorem 1's assumption) and the effectiveness of our safeguard, and (c) the ratio of INTELLI's error bounds to true error bounds (Theorem 1's condition). In (b), 95% percentiles being lower than 1 indicates their probabilistic correctness. In (c), the condition in Theorem 1 is satisfied if the ratio is equal to or larger than 1.



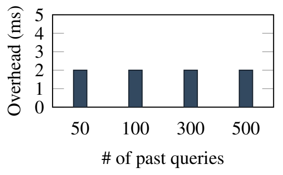
(a) Workload Diversity



(b) Data distribution



(c) Learning Behavior



(d) Overhead Consistency

Figure 11: The effectiveness of INTELLI in reducing NOLEARN's error for different (a) levels of diversity in the queried columns, (b) data distributions, and (c) number of past queries observed. Figure (d) shows INTELLI's overhead for different number of past queries.

Lastly, we examined if INTELLI can still improve the quality of new queries even if there are no inter-tuple correlations in the data. For this experiment, we executed two sets of new queries: the queries in the first set overlapped with the past ones while the queries in the second set did not. We ran each set of queries against two synthetic datasets with strong and extremely weak[14] inter-tuple correlations, respectively. Figure 9(d) shows that INTELLI could successfully reduce the errors of the query answers from the underlying AQP engine even when inter-tuple correlations were extremely weak. As expected, the only case in which INTELLI was not effective was when the new queries did not overlap with the past ones, and inter-tuple correlations were close to zero.

6.5 Correlation Parameter Learning

In this section, we examine the correctness of INTELLI's correlation parameter learning (§4.3) using synthetic datasets with known correlation parameters.

As shown in Figure 10(a), INTELLI's likelihood-based parameter search was accurate in finding values close to the true correlation parameters. In Figure 10(b), we also show the ratios between the actual errors and INTELLI's error bounds (i.e., 95%-confidence intervals). Based on this plot, we can make two important observations: (i) the maximum entropy (ME) distribution derived from true statistics (i.e., scale 1.0x) produced probabilistically correct error bounds (assumption in Theorem 1), (ii) the error bound violations stemming from highly overestimated parameters were successfully prevented by our safeguard. Lastly, Figure 10(c) shows that INTELLI's error bounds were slightly larger than their respective true error bounds, indicating that the condition in Theorem 1 was satisfied in most cases.

6.6 Confidence Interval Guarantees

To confirm the validity of INTELLI's probabilistic error guarantees using δ-confidence intervals, we configured INTELLI to run each query with different 95%-confidence intervals and measured the actual error in each case. Figure 12 shows the 5% percentile, mean, and 95% percentile of the actual errors across different queries. The results demonstrate that INTELLI's confidence interval guarantees were almost never violated.

[14]We did not use zero correlation dataset, since in that case, the summations over zero-correlated data (i.e., white noise) would tend to be trivial values (i.e., mostly zeros).

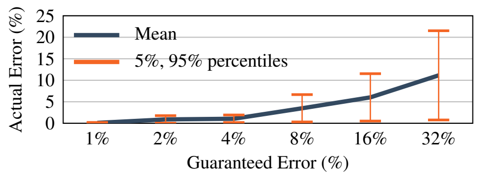


Figure 12: Correctness of INTELLI's estimated error guarantees. Guaranteed errors were obtained using 95% confidence intervals.

6.7 Impact of Different Data Distributions and Workload Characteristics

In this section, we generated various synthetic datasets and queries to fully understand how INTELLI's effectiveness changes for different data distributions, query patterns, and number of past queries.

First, we studied the impact of having queries with a more diverse set of columns in their selection predicates. We produced a table of 50 columns and 5M rows, and generated four different query workloads with varying number of frequently accessed columns. The columns used for the selection predicates were chosen according to a power-law distribution. Specifically, a fixed number of columns (called *frequently access columns*) had the same probability of being accessed, but the access probability of the remaining columns decayed according to the power-law distribution. For instance, if the number of frequently access columns was 10, the first ten columns appeared with equal probability in each query, but the probability of appearance reduced by half for every remaining column. Figure 11(a) shows that as the number of frequently accessed columns increased, INTELLI's relative error reduction over NOLEARN gradually decreased (the number of past queries were fixed to 100). This is expected as INTELLI constructs its internal model based on the columns appearing in the past. In other words, to cope with the more diversity, more past queries are

Latency	Cached	No-Cache
NoLearn	2.083 sec	52.50 sec
Intelli	2.093 sec	52.51 sec
Overhead	0.010 sec (0.48%)	0.010 sec (0.02%)

Table 4: The runtime overhead of Intelli.

needed to understand the complex underlying distribution generating the data. Note that, according to the analytic queries in the Customer1 dataset, most of the queries included less than 5 distinct selection predicates. However, by processing more queries, Intelli continues to learn more about the underlying distribution, producing larger error reductions even when the workload is extremely diverse.

Second, to study Intelli's potential sensitivity, we generated three tables using three different probability distributions: uniform, Gaussian, and a log-normal (skewed) distribution. Figure 11(b) shows Intelli's error reductions when queries were run against each table. Intelli delivered a consistent performance regardless of the underlying distribution. This is due to the power and generality of the maximum entropy principle taken by Intelli.

Third, we varied the number of past queries observed by Intelli before running our test queries. Figure 11(c) demonstrates that the error reduction keeps increasing until seeing 300 queries and then leveled off. This is due to the asymptotic nature of estimated errors, enabling Intelli to deliver reasonable performance without having to observe too many queries.

Lastly, we studied the negative impact of increasing the number of past queries on Intelli's overhead. Since Intelli's inference consists of a small matrix multiplication, we did not observe a noticeable increase as the number of queries in the query synopsis increased (Figure 11(d)).

6.8 Memory and Computational Overhead

In this section, we study Intelli's additional memory footprint (due to query synopsis) and its runtime overhead (due to inference). The total size of the generated snippets was on average 150KB per query for Customer1, and 8 KB per query for TPC-H. This is because Intelli only stores query answers and does not keep any of the input tuples.

To measure Intelli's runtime overhead, we recorded the time spent for its regular query processing (i.e., NoLearn) and the additional time spent for the inference and updating the final answer. As summarized in Table 4, the runtime overhead of Intelli was negligible compared to the overall query processing time. This is because multiplying a vector by a $C_g \times C_g$ matrix does not take much time compared to regular query planning, processing, and network commutations among the distributed nodes. (Note that C_g=1000 by default; see §2.3.)

7. RELATED WORK

Approximate Query Processing— There has been substantial work on sampling-based approximate query processing [6,7,9,13,19,24,40,49]. Some of these systems differ in their sample generation strategies. For instance, STRAT [19] and AQUA [7] create a single stratified sample, while BlinkDB creates samples based on *column sets*. Online Aggregation (OLA) [20,27,41,52] continuously refine its answers during query execution. Other have focused on obtaining faster or more reliable error estimates [8,53]. These are orthogonal to our work, as reliable error estimates from an underlying AQP engine will also benefit DBL.

Adaptive Indexing, View Selection— Adaptive Indexing and database cracking [29,43] incrementally update indices as part of query processing in order to speed up future queries accessing previously accessed tuples. Materialized views are another means of speeding up future queries [23,26,32]. While these techniques are orthogonal to DBL (i.e., they can be used in the underlying AQP engine), they are fundamentally different than DBL (refer to Previous Approaches in Section 1).

Pre-computation— COSMOS [52] stores the results of past queries as multi-dimensional cubes. These aggregated cubes are then reused if they are contained in the new query's input range, while boundary tuples are read from the database. This approach is not probabilistic and is limited to low-dimensional data due to the exponential explosion in the number of possible cubes. Also, similar to view selection, COSMOS relies on strict query containment.

Model-based and Statistical Databases— Statistical approaches have been used in databases for various goals. MauveDB [21] constructs views that express a statistical model, hiding the possible irregularities of the underlying data. MauveDB's goal is to support statistical modeling, such as regression or interpolation, rather than speeding up future queries. BayesDB [36] provides a SQL-like language that enables non-statisticians to declaratively use various statistical models.

8. CONCLUSION AND FUTURE WORK

In this paper, we presented database learning, a novel approach to exploit past queries' (approximate) answers in speeding up new queries using a principled statistical methodology. We presented a prototype of this vision, called Intelli, on top of SparkSQL. Through extensive experiments on real-world and benchmarks query logs, we demonstrated that Intelli supports 73.7% of real-world analytical queries, speeding them up by up to 10.43x compared to existing sampling-based approximation engines.

Exciting lines of future work include: (i) study other inferential techniques for realizing database learning, (ii) develop an idea of *active database learning*, whereby the engine itself proactively executes certain approximate queries that can best improve its internal model, and (iii) extend Intelli to support visual analytics [42].

9. REFERENCES

[1] https://db.apache.org/derby/docs/10.6/tuning/ctuntransform36368.html.

[2] Fast, approximate analysis of big data (yahoo's druid). http://yahooeng.tumblr.com/post/135390948446/data-sketches.

[3] Presto: Distributed SQL query engine for big data. https://prestodb.io/docs/current/release/release-0.61.html.

[4] SnappyData. http://www.snappydata.io/.

[5] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, 1999.

[6] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[7] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. In *SIGMOD*, 1999.

[8] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when

you're wrong: Building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[9] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[10] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.

[11] D. Antenucci, M. R. Anderson, P. Zhao, and M. Cafarella. A query system for social media signals. *ICDE*, 2016.

[12] M. Armbrust et al. Spark sql: Relational data processing in spark. In *SIGMOD*, 2015.

[13] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *VLDB*, 2003.

[14] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 1996.

[15] C. M. Bishop. Pattern recognition. *Machine Learning*, 2006.

[16] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In *Machine learning*. 1983.

[17] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[18] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards Estimation Error Guarantees for Distinct Values. In *PODS*, 2000.

[19] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.

[20] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, 2010.

[21] A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *SIGMOD*, 2006.

[22] A. Dobra, C. Jermaine, F. Rusu, and F. Xu. Turbo-charging estimate convergence in dbo. *PVLDB*, 2009.

[23] A. El-Helw, I. F. Ilyas, and C. Zuzarte. Statadvisor: Recommending statistical views. *VLDB*, 2009.

[24] V. Ganti, M.-L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, 2000.

[25] G. Graefe and H. Kuno. Adaptive indexing for relational keys. In *ICDEW*, 2010.

[26] A. Y. Halevy. Answering queries using views: A survey. *VLDBJ*, 2001.

[27] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[28] Y. Hu, S. Sundara, and J. Srinivasan. Estimating Aggregates in Time-Constrained Approximate Queries in Oracle. In *EDBT*, 2009.

[29] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *CIDR*, 2007.

[30] J. S. R. Jang. General formula: Matrix inversion lemma. `http://www.cs.nthu.edu.tw/~jang/book/addenda/matinv/matinv/`.

[31] Y. Jia. Running tpc-h queries on hive. `https://issues.apache.org/jira/browse/HIVE-600`.

[32] S. Joshi and C. Jermaine. Materialized sample views for database approximation. *TKDE*, 2008.

[33] S. Joshi and C. Jermaine. Sampling-Based Estimators for Subset-Based Queries. *VLDB J.*, 18(1), 2009.

[34] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, 2016.

[35] M. Lichman. UCI machine learning repository, 2013.

[36] V. Mansinghka et al. Bayesdb: A probabilistic programming system for querying the probable implications of data. *arXiv*, 2015.

[37] C. A. Micchelli, Y. Xu, and H. Zhang. Universal kernels. *JMLR*, 2006.

[38] B. Mozafari and N. Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 2015.

[39] B. Mozafari, P. Sarkar, M. J. Franklin, M. I. Jordan, and S. Madden. Scaling up crowd-sourcing to very large datasets: A case for active learning. *PVLDB*, 8, 2014.

[40] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed. Interactive Analysis of Web-Scale Data. In *CIDR*, 2009.

[41] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4, 2011.

[42] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. *ICDE*, 2016.

[43] E. Petraki, S. Idreos, and S. Manegold. Holistic indexing in main-memory column-stores. In *SIGMOD*, 2015.

[44] A. Pol and C. Jermaine. Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, 2005.

[45] F. Rusu, C. Qin, and M. Torres. Scalable analytics model calibration with online aggregation. *IEEE Data Eng. Bull.*, 2015.

[46] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959.

[47] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, 2000.

[48] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *MSST*, 2010.

[49] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *CIDR*, 2011.

[50] A. Silberschatz, H. F. Korth, S. Sudarshan, et al. *Database system concepts*. 1997.

[51] J. Skilling. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2006.

[52] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous Sampling for Online Aggregation over Multiple Queries. In *SIGMOD*, pages 651–662, 2010.

[53] F. Xu, C. Jermaine, and A. Dobra. Confidence bounds for sampling-based group by estimates. *TODS*, 2008.

[54] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, 2015.

[55] K. Zeng, S. Agarwal, and I. Stoica. iolap: Managing uncertainty for efficient incremental olap. 2016.

[56] K. Zeng, S. Gao, J. Gu, B. Mozafari, and C. Zaniolo. Abs: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, 2014.

[57] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.

APPENDIX

A. MATHEMATICAL DETAILS OF COMPUTING COVARIANCES BETWEEN QUERY ANSWERS

Let g be an aggregate function on attribute A_k, $\boldsymbol{a}_k$ be a random variable representing the possible values of attribute A_k, $\boldsymbol{t} = (a_1, \ldots, a_l)$ be a vector of length l comprised of values for r's dimension attributes $A_1, \ldots, A_l$, and $\omega(\boldsymbol{t})$ be the fraction of tuples in r that have the same values for their dimension attributes as $\boldsymbol{t}$.

We use $\nu_g(\boldsymbol{t})$ to denote the expected value of aggregate function g given the values of dimension attributes. Thus, we have:

$$\nu_g(\boldsymbol{t}) = \begin{cases} E[\boldsymbol{a}_k \mid \boldsymbol{t}] & \text{if } g \text{ is } \texttt{AVG(}A_k\texttt{)} \\ \omega(\boldsymbol{t}) & \text{if } g \text{ is } \texttt{FREQ(*)} \end{cases}$$

Therefore, $\nu_g(\boldsymbol{t})$ is a random variable dependent on $\boldsymbol{t}$ (since relation r is drawn from an underlying distribution).

Let F_i denote the set of tuples satisfying the selection predicates of q_i. Then, the random variable $\bar{\boldsymbol{\theta}}_i$ representing possible true answers to q_i can be expressed in terms of $\nu_g(\boldsymbol{t})$:

$$\bar{\boldsymbol{\theta}}_i = \begin{cases} \dfrac{1}{\mathscr{Z}_i} \displaystyle\int_{F_i} \nu_g(\boldsymbol{t})\, \omega(\boldsymbol{t})\, d\boldsymbol{t} & \text{if } g \text{ is } \texttt{AVG(}A_k\texttt{)} \\ \displaystyle\int_{F_i} \nu_g(\boldsymbol{t})\, d\boldsymbol{t} & \text{if } g \text{ is } \texttt{FREQ(*)} \end{cases} \tag{19}$$

where $\mathscr{Z}_i$ is a normalization term defined as $\mathscr{Z}_i = \int_{F_i} \omega(\boldsymbol{t})\, d\boldsymbol{t}$.

Based on equation (19), the covariance between random variables $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ can be expressed as follows. (This quantity corresponds to the variance of $\boldsymbol{\theta}_i$ if $i = j$.)

$$E\left[(\boldsymbol{\theta}_i - \mu_i)(\boldsymbol{\theta}_j - \mu_j)\right] = E\left[(\bar{\boldsymbol{\theta}}_i + \boldsymbol{\varepsilon}_i - \mu_i)(\bar{\boldsymbol{\theta}}_j + \boldsymbol{\varepsilon}_j - \mu_j)\right]$$
$$= \begin{cases} \dfrac{1}{\mathscr{Z}_i \mathscr{Z}_j} \displaystyle\int_{F_i}\int_{F_j} \operatorname{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}'))\, \omega(\boldsymbol{t})\omega(\boldsymbol{t}')\, d\boldsymbol{t}' d\boldsymbol{t} \\ \qquad + \delta(i,j)\cdot\beta_i \qquad \text{if } g \text{ is } \texttt{AVG(}A_k\texttt{)} \\ \displaystyle\int_{F_i}\int_{F_j} \operatorname{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}'))\, d\boldsymbol{t}' d\boldsymbol{t} \;+\; \delta(i,j)\cdot\beta_i \\ \qquad\qquad \text{if } g \text{ is } \texttt{FREQ(*)} \end{cases} \tag{20}$$

where δ is 1 if $i = j$ and 0 otherwise; and $\boldsymbol{\varepsilon}_i$ is a random variable with variance β_i representing the deviation of the raw answer from q_i's true answer.

B. DATA UPDATES

INTELLI supports tuple insertions.[15] A naïve strategy would be to re-execute all past queries every time new tuples are added to the database to obtain their updated answers. This solution is obviously impractical.

Instead, INTELLI still makes use of answers to past queries even when new tuples have been added since computing their answers. The basic idea is to simply lower our confidence in the raw answers of those past queries.

B.1 Error Adjustment

Assume that q_i (whose aggregate function is on A_k) is computed on an old relation r, and a set of new tuples r' has since been added to r to form an updated relation r^u. Let $\bar{\boldsymbol{\theta}}'_i$ be a random variable representing the possible values for q_i's true answer on r', and $\bar{\theta}^u_i$ be q_i's true answer on r^u.

[15]Other forms of data updates (e.g., deletion) are not supported, as INTELLI is currently implemented atop SparkSQL which (similar to other HDFS-based engines) is an append-only database.

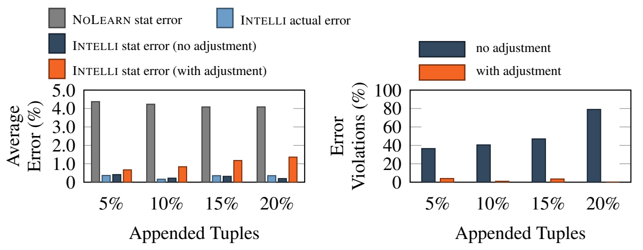


Figure 13: Data append technique (appendix B) is highly effective in delivering correct error estimates in face of new data.

We represent the possible difference between A_k's values in r and those in r' by a random variable $\boldsymbol{s}_k$ with mean α_k and variance η_k^2. Thus:

$$\bar{\boldsymbol{\theta}}'_i = \bar{\theta}_i + \boldsymbol{s}_k$$

The values of α_k and η_k^2 are estimated using small samples of r and r'. INTELLI uses the following lemma to update the raw answer and raw error for q_i.

Lemma 3.

$$E[\bar{\theta}^u_i - \boldsymbol{\theta}_i] = \alpha_s \cdot \frac{|r'|}{|r| + |r'|}$$

$$E[(\bar{\theta}^u_i - \boldsymbol{\theta}_i - \alpha_s \cdot \frac{|r'|}{|r|+|r'|})^2] = \beta_i^2 + \left(\frac{|r'|}{|r|+|r'|}\cdot\eta_k\right)^2$$

where $|r|$ and $|r'|$ are the number of tuples in r and r', respectively.

Once the raw answers and the raw errors of past queries are updated using this lemma, the remaining inference process remains the same.

B.2 Experiment

In this section, we study the impact of new data (i.e., tuple insertions) on INTELLI's effectiveness. Similar to the previous section, we generated an initial synthetic table with 5M tuples and appended additional tuples to generate different versions of the table. The newly inserted tuples were generated such that their attribute values gradually diverged from the attribute values of the original table. We distinguish between these different versions by the ratio of their newly inserted tuples, e.g., a 5% appended table means that 250K (5% of 5M) tuples were added. We then ran the queries and recorded the average estimated errors of INTELLIADJUST and INTELLINOADJUST (our approach with and without the technique introduced in appendix B). We also measured the estimated error of NOLEARN and the actual error.

As shown in Figure 13(a), INTELLINOADJUST produced overly-optimistic estimated errors (i.e., lower than the actual error) for 15% and 20% appends, whereas INTELLIADJUST produced valid estimated errors in all cases. Since this figure shows the *average* estimated errors across all queries, we also computed the fraction of individual queries for which each method's estimated error was violated. In Figure 13(b), the Y-axis indicates those cases where the actual error was higher than the guaranteed estimated error. This figure shows more error violations for INTELLINOADJUST, which increased with the number of new tuples. In contrast, INTELLIADJUST produced valid estimated errors in most cases, while delivering substantial error reductions compared to NOLEARN.