

# 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 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 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, and hence lead to increasingly faster response times for 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 Verdict. 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 up to 23.0× 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., [6, 16, 31, 33, 57, 74]), one might be able to change this paradigm and reuse much of the previous work done by the database system 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.*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Unpublished (In submission)

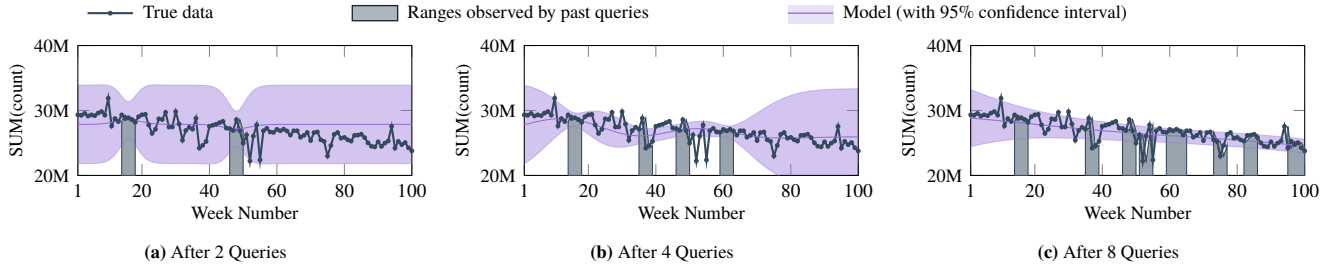
© 2017 ACM. ISBN 123-4567-24-567/08/06...\$15.00

DOI: 10.475/123\_4

This is because the answers to different queries stem from the same (unknown) underlying distribution that has generated 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 (or even a large sample of it), one can use a small sample of it to quickly produce a rough approximate answer, which can then be calibrated and combined with the model to obtain 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 Learning (ML) whereby past observations are used to improve future predictions [14, 15, 60]. 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 refine 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 (in purple) 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 Figure 1(b) and Figure 1(c), DBL further refines its model as more new queries are answered. This approach would allow a DBL-enabled query engine to provide increasingly more accurate estimates, *even for those 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. The goal of this



**Figure 1:** An example of how database learning might continuously refine its model as more queries are processed: after processing (a) 2 queries, (b) 4 queries, and (c) 8 queries. We could deliver more accurate answers if we combined this model with the approximate answers produced by traditional sampling techniques.

simplified example<sup>1</sup> 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.

**Challenges**— To realize DBL’s vision in practice, three key challenges must be overcome. 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, DBL must not make any assumptions about the data distribution; the only valid knowledge must come from past queries and their respective answers. Finally, there is an *efficiency* challenge. DBL needs to strike a balance between the computational complexity of its 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 [29,35,56] or materialized views [7] 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 the 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 [35]) 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. Finally, materialized views<sup>2</sup> are only beneficial when there is a strict structural compatibility—such as query containment or equality—between past and new queries [30].

<sup>1</sup>In general, DBL does not make any *a priori* assumptions regarding correlations (or smoothness) in the data; any correlations present in the data will be naturally revealed through analyzing the answers to past queries, in which case DBL automatically identifies and makes use of them (see Sections 2.4 and 4.3).

<sup>2</sup>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.

The fundamental between DBL and these traditional approaches lead to a few interesting characteristics:

1. Since materialized views, indexing, and caching are for exact query processing, they are only effective when new queries touch previously accessed columns or tuples. On the contrary, DBL works in AQP settings; thus, DBL 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 most likely extrapolation even for unobserved parts of data.
2. Unlike indices and materialized views, DBL incurs *little storage overhead* as it only retains the past  $n$  aggregate queries and their answers.<sup>3</sup> In contrast, indices and materialized views grow in size as the data grows, while DBL’s storage requirement remains *oblivious to the data size*.

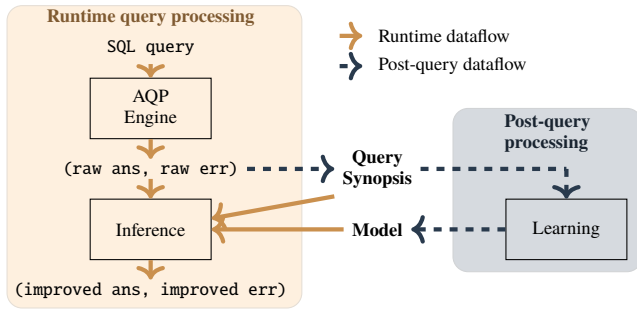
**Our Approach**— Our vision of database learning (DBL) might be achieved in different ways depending on the design decisions made in terms of query generality, data generality, and efficiency. In this paper, besides the introduction of the concept of DBL, we also provide a specific solution for achieving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

From a high-level, Verdict addresses the three challenges—query generality, data generality, and efficiency—as follows. First, Verdict supports SQL queries by decomposing them into simpler atomic units, called *snippets*. The answer to a snippet is a single scalar value; thus, our belief on the answer to each snippet can be expressed as a random variable, which can then be used in our mathematical model. Second, to achieve data generality, Verdict employs a *non-parametric* probabilistic model, which is capable of representing arbitrary underlying distributions. This model is based on a simple intuition: *when two queries share some tuples in their aggregations, their answers must be correlated*. Our probabilistic model is a formal generalization of this idea using *the principle of maximum entropy* [65]. Third, to ensure computational efficiency, we keep our probabilistic model in an analytic form. At query time, we only require a matrix-vector multiplication; thus, the overhead is negligible.

**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 Verdict. Verdict’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. Formally, we also prove that Verdict’s expected errors are never larger than those of existing AQP techniques.

<sup>3</sup>Even if a query outputs too many tuples, DBL retains only a fixed number of them (see Section 2.3)



**Figure 2:** Workflow in Verdict. At query time, the Inference module uses the *Query Synopsis* and the *Model* to improve the query answer and error computed by the underlying AQP engine (i.e., *raw answer/error*) before returning them to the user. Each time a query is processed, the raw answer and error are added to the *Query Synopsis*. The Learning module uses this updated *Query Synopsis* to refine the current *Model* accordingly.

3. We integrate Verdict on top of Spark SQL, and conduct experiments using both benchmark and real-world query traces. Verdict delivers up to 23× speedup and 90% error reduction compared to AQP engines that do not use DBL.

The rest of this paper is organized as follows. Section 2 overviews Verdict’s workflow, supported query types, and query processing. Sections 3 and 4 describe the internals of Verdict in detail, and Section 5 presents Verdict’s formal guarantees. Section 6 discusses Verdict’s deployment scenarios. Section 7 reports our empirical results. Section 8 summarizes related work, and Section 9 concludes the paper with future work.

## 2. VERDICT OVERVIEW

In this section, we overview the system we have built based on database learning (DBL), called Verdict. Section 2.1 explains Verdict’s architecture and overall workflow. Section 2.2 presents the class of SQL queries currently supported by Verdict. Section 2.3 introduces Verdict’s query representation. Section 2.4 describes the intuition behind Verdict’s inference. Lastly, Section 2.5 discusses the limitations of Verdict’s approach.

### 2.1 Architecture and Workflow

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

Verdict converts each incoming query into multiple *query snippets*, which are the basic units of inference in Verdict.

**Definition 1. (Query Snippet)** A query snippet is a supported<sup>4</sup> SQL query whose answer is a *single scalar value*.

Section 2.3 describes how a supported query (whose answer may be a set) is decomposed into possibly multiple query snippets. For the  $i$ -th query snippet  $q_i$ , the AQP engine produces a pair of an approximate answer  $\theta_i$  and a corresponding expected error  $\beta_i$ .  $\theta_i$  and  $\beta_i$  are formally defined in Section 3.1, and are produced by most AQP systems [6, 31, 53, 73, 74, 76]. Now we can formally define the first key component of our system, the *query synopsis*.

**Definition 2. (Query Synopsis)** Let  $n$  be the number of query snippets processed thus far by the AQP engine. The query synopsis  $Q_n$  is defined as the following set:  $\{(q_i, \theta_i, \beta_i) \mid i = 1, \dots, n\}$ .

<sup>4</sup>Supported queries are defined in Section 2.2.

Term	Definition
<b>raw answer</b>	answer computed by the AQP engine
<b>raw error</b>	expected error for raw answer
<b>improved answer</b>	answer updated by Verdict
<b>improved error</b>	expected error for improved answer (by Verdict)
<b>past snippet</b>	supported query snippet processed in the past
<b>new snippet</b>	incoming query snippet

**Table 1:** Terminology.

We call the query snippets in the query synopsis *past snippets*, and call the  $(n + 1)$ -th query snippet the *new snippet*.

The second key component is the *model*, which represents Verdict’s statistical understanding of the underlying data. The model is trained on the query synopsis, and is updated every time a query is added to the synopsis (Section 4).

The query-time workflow of Verdict is as follows. Given an incoming query snippet  $q_{n+1}$ , Verdict invokes the AQP engine to compute a raw answer  $\theta_{n+1}$  and a raw error  $\beta_{n+1}$ . Then, Verdict combines this raw answer/error and the previously computed model to *infer* an *improved answer*  $\hat{\theta}_{n+1}$  and an associated expected error  $\hat{\beta}_{n+1}$ , called *improved error*. Theorem 1 shows that the improved error is never larger than the raw error. After returning the improved answer and the improved error to the user,  $(q_{n+1}, \theta_{n+1}, \beta_{n+1})$  is added to the query synopsis.

A key objective in Verdict’s design is to treat the underlying AQP engine as a black box. This allows Verdict to be used with many of the existing engines without requiring any modifications. From the user’s perspective, the benefit of using Verdict (compared to using the AQP engine alone) is the error reduction and speedup, or only the error reduction, depending on the type of AQP engine used (Section 6).

Lastly, Verdict does not modify non-aggregate expressions or unsupported queries, i.e., it simply returns their raw answers/errors to the user. Table 1 summarizes the terminology defined above. In Section 3, we will recap the mathematical notations defined above.

### 2.2 Supported Queries

Verdict 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<sup>5</sup> and any number of dimension tables are supported. For simplicity, our discussion in this paper is based on a denormalized table.
3. **Selections.** Verdict currently supports equality and inequality comparisons for categorical and numerical attributes (including the *in* operator). Currently, Verdict does not support disjunctions and 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.<sup>6</sup>

**Nested Query Support**— Although Verdict does not directly support nested queries, many queries can be flattened using joins [1] or by creating intermediate views for sub-queries [30]. In fact, this is the process used by Hive for supporting the nested queries of the

<sup>5</sup>Data warehouses typically record measurements (e.g., sales) into fact tables and normalize commonly appearing attributes (e.g., seller information) into dimension tables [64].

<sup>6</sup>Note that the underlying AQP engine may affect the cardinality of the result set depending on the *having* clause (i.e., subset/superset error). Verdict simply operates on the result set returned by the AQP engine.

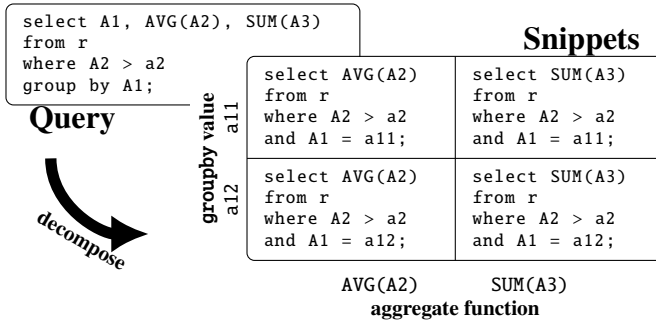


Figure 3: Example of a query’s decomposition into multiple snippets.

TPC-H benchmark [38]. We are currently working to automatically process nested queries and to expand the class of supported queries (see Section 9).

**Unsupported Queries**— Each query, upon its arrival, is inspected by Verdict’s query type checker to determine whether it is supported, and if not, Verdict bypasses the Inference module and simply returns the raw answer to the user. The overhead of the query type checker is negligible (Section 7.5) compared to the runtime of the AQP engine; thus, Verdict does not incur any noticeable runtime overhead, even when a query is not supported.

Only supported queries are stored in Verdict’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

**Decomposing Queries into Snippets**— As mentioned in Section 2.1, each supported query is broken into (possibly) multiple *query snippets* before being added to the query synopsis. Conceptually, each snippet corresponds to a supported SQL query with a single aggregate function, with no other projected columns in its `select` clause, and with no `groupby` clause; thus, the answer to each snippet is a single scalar value. A SQL query with multiple aggregate functions (e.g., `AVG(A2)`, `SUM(A3)`) 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, Verdict only generates snippets for  $N^{max}$  (1,000 by default) groups in the answer set.

For each aggregate function  $g$ , the query synopsis retains a maximum of  $C_g$  query snippets by following a least recently used snippet replacement policy (by default,  $C_g=2,000$ ). This improves the efficiency of the inference process, while maintaining an accurate model based on the recently processed snippet answers.

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

- $AVG(A_k) = AVG(A_k)$
- $COUNT(*) = FREQ(*) \times (\text{table cardinality})$
- $SUM(A_k) = AVG(A_k) \times COUNT(*)$

## 2.4 Intuition: Why and When Verdict Offers Benefit

In this section, we provide the high level intuition behind Verdict’s approach to improving the quality of new snippet answers. Verdict exploits potential correlations between snippet answers to infer the answer of a new snippet. Let  $S_i$  and  $S_j$  be multisets of attribute values such that, when aggregated, they output exact answers to queries  $q_i$  and  $q_j$ , respectively. Then, the answers to  $q_i$  and  $q_j$  are correlated, if:

1.  **$S_i$  and  $S_j$  include common values.**  $S_i \cap S_j \neq \emptyset$  would imply the existence of correlation between the two snippet 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. Verdict captures the likelihood of such attribute value similarities using a statistical measure called *inter-tuple covariance*, which will be formally defined in Section 4.2. In the presence of non-zero inter-tuple covariances, the answers to  $q_i$  and  $q_j$  could be correlated even when  $S_i \cap S_j \neq \emptyset$ . In practice, most real-life datasets tend to have non-zero inter-tuple covariances, i.e., correlated attribute values (see Appendix C for an empirical study).

Verdict formally captures the correlations between pairs of snippets using a probabilistic distribution function. At query time, this probabilistic distribution function is used to infer the most likely answer to the new snippet given the answers to past snippets. We explain this process in detail in Section 3.

## 2.5 Limitations

Verdict’s model is the most likely explanation of the underlying distribution given the limited information stored in the query synopsis. Consequently, when a new snippet involves tuples that have never been accessed by past snippets, it is possible that Verdict’s model might incorrectly represent the underlying distribution, and return incorrect error bounds. To guard against this limitation, Verdict always validates its model-based answer against the (model-free) answer of the AQP engine. We present this model validation step in Section 3.5, confirm its effectiveness in Section 7, and evaluate its robustness in Appendix A.3.

Because Verdict relies on off-the-shelf AQP engines for obtaining raw answers and raw errors, it is naturally bound by the limitations of the underlying engine. For example, it is known that sample-based engines are not apt at supporting arbitrary joins or `MIN/MAX` aggregates. Similarly, the validity of Verdict’s error guarantees are contingent upon the validity of the AQP engine’s raw errors. Fortunately, there are also off-the-shelf diagnostic techniques to verify the validity of such errors [5].

## 3. INFERENCE

In this section, we describe Verdict’s inference process for computing an improved answer (and improved error) for the new snippet. First, we formally state our problem in Section 3.1. A mathematical interpretation of the problem and the overview on Verdict’s approach is described in Section 3.2. Sections 3.3 to 3.5 present the details of the Verdict’s approach to solving the problem. Section 3.6 discusses some challenges in applying Verdict’s approach.

Sym.	Meaning
$q_i$	$i$ -th (supported) query snippet
$n + 1$	index number for a new snippet
$\theta_i$	random variable representing our knowledge of the raw answer to $q_i$
$\theta_i$	(actual) raw answer computed by AQP engine for $q_i$
$\beta_i$	expected error associated with $\theta_i$
$\bar{\theta}_i$	random variable representing our knowledge of the <i>exact</i> answer to $q_i$
$\bar{\theta}_i$	exact answer to $q_i$
$\hat{\theta}_{n+1}$	improved answer to the new snippet
$\hat{\beta}_{n+1}$	improved error to the new snippet

**Table 2:** Mathematical Notations.

### 3.1 Problem Statement

Let  $r$  be a relation<sup>7</sup> drawn from some unknown underlying distribution. Let  $r$ 's attributes be  $A_1, \dots, A_m$ , where  $A_1, \dots, A_l$  are the *dimension attributes* and  $A_{l+1}, \dots, A_m$  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*. Table 2 summarizes the notations we defined earlier in Section 2.1.

Given a query snippet  $q_i$  on  $r$ , an AQP engine returns a raw answer  $\theta_i$  along with an associated expected error  $\beta_i$ . Formally,  $\beta_i^2$  is the expectation of the squared deviation of  $\theta_i$  from the (unknown) exact answer  $\bar{\theta}_i$  to  $q_i$ .<sup>8</sup>

Suppose  $n$  query snippets have been processed, and therefore the query synopsis  $Q_n$  contains the raw answers and raw errors for the past  $n$  query snippets. Without loss of generality, we assume all queries have the same aggregate function  $g$  on  $A_k$  (e.g.,  $\text{AVG}(A_k)$ ), where  $A_k$  is one of the measure attributes. Our problem is then stated as follows: *given  $Q_n$  and  $(\theta_{n+1}, \beta_{n+1})$ , compute the most likely answer to  $q_{n+1}$  with an associated expected error.*

In our discussion, for simplicity, we assume static data, i.e., the new snippet is issued against the same data that has been used for answering past snippets in  $Q_n$ . However, Verdict can also be extended to situations where the relations are subject to new data being added, i.e., each snippet is answered against a potentially different version of the dataset. Appendix B discusses Verdict under data updates.

### 3.2 Inference Overview

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

Our approach uses (probabilistic) random variables to represent *our knowledge of the query answers*. The use of random variables here is a natural choice as our knowledge itself of the query answers is uncertain.<sup>9</sup> Specifically, we denote our knowledge of the raw answer and the exact answer to the  $i$ -th query snippet by random variables  $\theta_i$  and  $\bar{\theta}_i$ , respectively. At this step, the only information available to us regarding  $\theta_i$  and  $\bar{\theta}_i$  is that  $\theta_i$  is an instance of  $\theta_i$ ; no other assumptions are made.

Next, we represent the relationship between the set of random variables  $\theta_1, \dots, \theta_{n+1}, \bar{\theta}_{n+1}$  using a joint probability distribution function (pdf). Note that the first  $n + 1$  random variables are for

the raw answers to past  $n$  snippets and the new snippet, and the last random variable is for the exact answer to the new snippet. We are interested in the relationship among those random variables because our knowledge of the query answers is based on limited information: the raw answers computed by the AQP engine, whereas we aim to find the most likely value for the new snippet's exact answer. This joint pdf represents Verdict's *prior belief* over the query answers. We denote the joint pdf by  $f(\theta_1 = \theta'_1, \dots, \theta_{n+1} = \theta'_{n+1}, \bar{\theta}_{n+1} = \bar{\theta}'_{n+1})$ . For brevity, we also use  $f(\theta'_1, \dots, \theta'_{n+1}, \bar{\theta}'_{n+1})$  when the meaning is clear from the context. (Recall that  $\theta_i$  refers to an actual raw answer from the AQP engine, and  $\bar{\theta}_{n+1}$  refers to the exact answer to the new snippet.) The joint pdf returns the probability that the random variables  $\theta_1, \dots, \theta_{n+1}, \bar{\theta}_{n+1}$  takes a particular combination of the values, i.e.,  $\theta'_1, \dots, \theta'_{n+1}, \bar{\theta}'_{n+1}$ . In Section 3.3, we discuss how to obtain this joint pdf from some statistics available on query answers.

Then, we compute the most likely value for the new snippet's exact answer, namely the most likely value for  $\bar{\theta}_{n+1}$ , by first conditionalizing the joint pdf on the actual observations (i.e., raw answers) from the AQP engine, i.e.,  $f(\bar{\theta}_{n+1} = \bar{\theta}'_{n+1} \mid \theta_1 = \theta_1, \dots, \theta_{n+1} = \theta_{n+1})$ . We then find the value of  $\bar{\theta}'_{n+1}$  that maximizes the conditional pdf. We call this value the *model-based answer* and denote it by  $\hat{\theta}_{n+1}$ . Section 3.4 provides more details of this process. Finally,  $\hat{\theta}_{n+1}$  and its associated expected error  $\hat{\beta}_{n+1}$  are returned as Verdict's improved answer and improved error if they pass the model validation (described in Section 3.5). Otherwise, the (original) raw answer and error are taken as Verdict's improved answer and error, respectively. In other words, if the model validation fails, Verdict simply returns the original raw results from the AQP engine without any improvements.

### 3.3 Prior Belief

In this section, we describe how Verdict obtains a joint pdf  $f(\theta'_1, \dots, \theta'_{n+1}, \bar{\theta}'_{n+1})$  that represents its knowledge of the underlying distribution. Recall that we mentioned in Section 2.4 that the intuition behind Verdict's inference is to make use of possible correlations between pairs of query answers. This section applies such statistical information of query answers (i.e., means, covariances, and variances) for obtaining the most likely joint pdf. Obtaining the query statistics is described in Section 4.

To obtain the joint pdf, Verdict relies on the principle of maximum entropy (ME) [12, 65], a simple but powerful statistical tool for determining a pdf of random variables given some statistical information available.<sup>10</sup> According to the ME principle, given 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*:

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

where  $\vec{\theta} = (\theta'_1, \dots, \theta'_{n+1}, \bar{\theta}'_{n+1})$ .

Note that the joint pdf maximizing the above entropy differs depending on the kinds of given testable information, i.e., query statistics in our context. For instance, the maximum entropy pdf given means of random variables is different from the maximum entropy pdf given means and (co)variances of random variables. 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

<sup>7</sup> $r$  can be a join or Cartesian product of multiple tables.

<sup>8</sup>Here, the expectation is made over  $\theta_i$  since the value of  $\theta_i$  depends on samples.

<sup>9</sup>Using random variables to represent degrees of belief is a standard approach in Bayesian inference.

<sup>10</sup>The ME principle has been also used for determining surprising information [61].



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 more accurate 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.

Among possible options, Verdict uses the first and the second order statistics of the random variables, i.e., mean, variances, and covariances. The use of second-order statistics enables us to capture the relationship among the answers to different query snippets, while the joint pdf that maximizes the entropy can be expressed in an analytic form. The uses of analytic forms provides computational efficiency. Specifically, the joint pdf that maximizes the entropy while satisfying the given means, variances, and covariances is a multivariate normal with the corresponding means, variances, and covariances [65].

**Lemma 1.** Let  $\vec{\theta} = (\theta_1, \dots, \theta_{n+1}, \bar{\theta}_{n+1})$  be a vector of  $n+2$  random variables with mean values  $\vec{\mu} = (\mu_1, \dots, \mu_{n+1}, \bar{\mu}_{n+1})$  and a  $(n+2) \times (n+2)$  covariance matrix  $\Sigma$  specifying their variances and pairwise covariances. The joint pdf  $f$  over these random variables that maximizes  $h(f)$  while satisfying the provided means, variances, and covariances is the following function:

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

and this solution is unique.

In the following section, we describe how Verdict computes the most likely answer to the new snippet using this joint pdf in equation (2). We call the most likely answer a *model-based answer*. In Section 3.5, this model-based answer is chosen as an improved answer if it passes a model validation. Finally, in Section 3.6, we discuss the challenges involved in obtaining  $\vec{\mu}$  and  $\Sigma$ , i.e., the query statistics required for deriving the joint pdf.

### 3.4 Model-based Answer

In the previous section, we formalized the relationship among query answers, namely  $(\theta_1, \dots, \theta_{n+1}, \bar{\theta}_{n+1})$ , using a joint pdf. In this section, we exploit this joint pdf to infer the most likely answer to the new snippet. In other words, we aim to find the most likely value for  $\bar{\theta}_{n+1}$  (the random variable representing  $q_{n+1}$ 's exact answer), given the observed values for  $\theta_1, \dots, \theta_{n+1}$ , i.e., the raw answers from the AQP engine. We call the most likely value a *model-based answer* and its associated expected error a *model-based error*. Mathematically, Verdict's model-based answer  $\bar{\theta}_{n+1}$  to  $q_{n+1}$  can be expressed as:

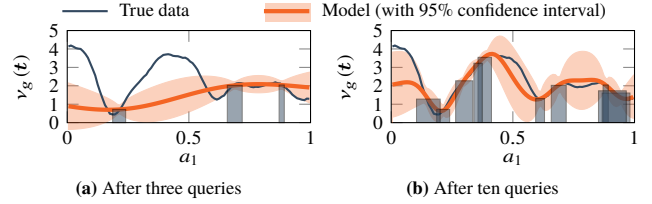
$$\bar{\theta}_{n+1} = \underset{\bar{\theta}_{n+1}}{\text{Arg Max}} f(\bar{\theta}_{n+1} | \theta_1 = \theta_1, \dots, \theta_{n+1} = \theta_{n+1}). \quad (3)$$

That is,  $\bar{\theta}_{n+1}$  is the value at which the conditional pdf has its maximum value. The conditional pdf,  $f(\bar{\theta}_{n+1} | \theta_1, \dots, \theta_{n+1})$ , is obtained by conditioning the joint pdf obtained in Section 3.3 on the observed values, i.e., raw answers to the past snippets and the new snippet.

Computing a conditional pdf may be a computationally expensive task. However, a conditional pdf of a multivariate normal distribution is analytically computable; it is another normal distribution. Specifically, the conditional pdf in equation (3) is a normal distribution with the following mean  $\mu_c$  and variance  $\sigma_c^2$  [13]:

$$\mu_c = \bar{\mu}_{n+1} + \vec{k}_{n+1}^\top \Sigma_{n+1}^{-1} (\vec{\theta}_{n+1} - \vec{\mu}_{n+1}) \quad (4)$$

$$\sigma_c^2 = \bar{\kappa}^2 - \vec{k}_{n+1}^\top \Sigma_{n+1}^{-1} \vec{k}_{n+1} \quad (5)$$



**Figure 4:** An example of (a) overly optimistic confidence intervals due to incorrect estimation of the underlying distribution, and (b) its resolution with more queries processed. Verdict relies on a model validation to avoid the situation as in (a).

where:

- $\vec{k}_{n+1}$  is a column vector of length  $n+1$  whose  $i$ -th element is  $(i, n+2)$ -th entry of  $\Sigma$ ;
- $\Sigma_{n+1}$  is a  $(n+1) \times (n+1)$  submatrix of  $\Sigma$  consisting of  $\Sigma$ 's first  $n+1$  rows and columns;
- $\vec{\theta}_{n+1} = (\theta_1, \dots, \theta_{n+1})^\top$ ;
- $\vec{\mu}_{n+1} = (\mu_1, \dots, \mu_{n+1})$ ; and
- $\bar{\kappa}^2$  is the  $(n+2, n+2)$ -th entry of  $\Sigma$

Since the mean of a normal distribution is the value at which the pdf takes a maximum value, we take  $\mu_c$  as our model-based answer  $\bar{\theta}_{n+1}$ . Likewise, the expectation of the squared deviation of the value  $\bar{\theta}_{n+1}$ , which is distributed according to the conditional pdf in equation (3), from the model-based answer  $\bar{\theta}_{n+1}$  coincides with the variance  $\sigma_c^2$  of the conditional pdf. Thus, we take  $\sigma_c$  as our model-based error  $\bar{\beta}_{n+1}$ .

Computing each of equations (4) and (5) requires  $O(n^3)$  time complexity at query time. However, Verdict uses alternative forms of these equations that require only  $O(n^2)$  time complexity at query time (Section 5).

Note that, since the conditional pdf is a normal distribution, the error bound at confidence  $\delta$  is expressed as  $\alpha_\delta \cdot \bar{\beta}_{n+1}$ , where  $\alpha_\delta$  is a non-negative number such that a random number drawn from a standard normal distribution would fall within  $(-\alpha_\delta, \alpha_\delta)$  with probability  $\delta$ . We call  $\alpha_\delta$  the confidence interval multiplier for probability  $\delta$ . That is, the exact answer  $\bar{\theta}_{n+1}$  is within the range  $(\bar{\theta}_{n+1} - \alpha_\delta \cdot \bar{\beta}_{n+1}, \bar{\theta}_{n+1} + \alpha_\delta \cdot \bar{\beta}_{n+1})$  with probability  $\delta$ , according to Verdict's model.

### 3.5 Model Validation

Verdict aims to provide correct error bounds even when its model differs significantly from the true data. Verdict achieves this goal by rejecting its model—the most likely explanation of the underlying distribution given the answers to past snippets—if there is evidence that its model-based error is likely to be incorrect.

Figure 4(a) illustrates an example of this situation in which Verdict's model produces incorrect error bounds. Here, after the first three queries, the model is consistent with past query answers (shown as gray boxes); however, it incorrectly estimates the distribution of the unobserved data, leading to overly optimistic confidence intervals. Figure 4(b) shows that the model becomes more consistent with the data as more queries are processed.

Verdict rejects (and does not use) its own model in situations such as Figure 4(a) by validating its model-based answers against the *model-free* answers obtained from the AQP engine. Specifically, we define a *likely region* as the range in which the AQP engine's answer would fall with high probability (99% by default) if Verdict's model were to be correct. If the AQP's raw answer  $\theta_{n+1}$  falls outside this

likely region, Verdict considers its model unlikely to be correct. In such cases, Verdict drops its model-based answer/error, and simply returns the raw answer to the user unchanged.<sup>11</sup>

Although no improvements are made in such cases, we take this conservative approach to ensure the correctness of our error guarantees. (See Section 5 and Appendix A.2 for formal guarantees and empirical results, respectively).

Formally, let  $t \geq 0$  be the value for which the AQP engine's answer would fall within  $(\tilde{\theta}_{n+1} - t, \tilde{\theta}_{n+1} + t)$  with probability  $\delta_v$  (0.99 by default) if  $\tilde{\theta}_{n+1}$  were the exact answer. We call the  $(\tilde{\theta}_{n+1} - t, \tilde{\theta}_{n+1} + t)$  range the likely region. To compute  $t$ , we must find the value closest to  $\tilde{\theta}_{n+1}$  that satisfies the following expression:

$$\Pr(|X - \tilde{\theta}_{n+1}| < |t - \tilde{\theta}_{n+1}|) \geq \delta_v \quad (6)$$

where  $X$  is a random variable representing the AQP engine's possible answer to the new snippet if the exact answer to the new snippet was  $\tilde{\theta}_{n+1}$ .<sup>12</sup> The probability in equation (6) can be easily computed using either the central limit theorem or the Chebyshev's inequality [47]. Once the value of  $t$  is computed, Verdict rejects its model if  $\theta_{n+1}$  falls outside the likely region  $(\tilde{\theta}_{n+1} - t, \tilde{\theta}_{n+1} + t)$ .

In summary, the pair of Verdict's improved answer and improved error,  $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$ , is set to  $(\tilde{\theta}_{n+1}, \tilde{\beta}_{n+1})$  if  $\theta_{n+1}$  is within the range  $(\tilde{\theta}_{n+1} - t, \tilde{\theta}_{n+1} + t)$ , and is set to  $(\theta_{n+1}, \beta_{n+1})$  otherwise. In either case, the error bound at confidence  $\delta$  remains the same as  $\alpha_\delta \cdot \hat{\beta}_{n+1}$ , where  $\alpha_\delta$  is the confidence interval multiplier for probability  $\delta$ .

### 3.6 Key Challenges

As mentioned in Section 3.3, obtaining the joint pdf in Lemma 1 (which represents Verdict's prior belief on query answers) requires the knowledge of means, variances, and covariances of the random variables  $(\theta_1, \dots, \theta_{n+1}, \tilde{\theta}_{n+1})$ . 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  $\theta_1, \dots, \theta_{n+1}$ , namely  $\theta_1, \dots, \theta_{n+1}$ . 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  $\tilde{\theta}_{n+1}$  (recall that  $\tilde{\theta}_{n+1}$  represents our knowledge of the exact answer to the new snippet, i.e.,  $\tilde{\theta}_{n+1}$ ). In Section 4, we present Verdict's approach to solving these challenges.

## 4. ESTIMATING QUERY STATISTICS

As described in Section 3, Verdict expresses its prior belief on the relationship among query answers as a joint pdf over a set of random variables  $(\theta_1, \dots, \theta_{n+1}, \tilde{\theta}_{n+1})$ . In this process, we need to know the means, variances, and covariances of these random variables.

Verdict uses the arithmetic mean of the past query answers for the mean of each random variable,  $\theta_1, \dots, \theta_{n+1}, \tilde{\theta}_{n+1}$ . 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. Computing (co)variances between a pair of snippet answers relies on a straightforward observation: *the covariance between two query snippet answers is computable using the covariances between the attribute values involved in computing those answers*. For instance, we can easily compute the covariance between (i) the average revenue of

the years 2014 and 2015 and (ii) the average revenue of the years 2015 and 2016, as long as we know the covariance between the average revenues of every pair of days in 2014–2016. The following sections formalize this idea.

In Section 4.1, we present a decomposition of the (co)variances between pairs of query snippet answers into *inter-tuple covariance* terms. Then, in Section 4.2, we describe how inter-tuple covariances can be estimated analytically using parameterized functions. In Section 4.3, we discuss the problem of determining optimal parameters for those covariance functions.

### 4.1 Covariance Decomposition

To compute the variances and covariances between query snippet answers (i.e.,  $\theta_1, \dots, \theta_{n+1}, \tilde{\theta}_{n+1}$ ), Verdict relies on our proposed *inter-tuple covariances*, which express 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 snippet can be mathematically represented in terms of the underlying distribution. This representation then naturally leads us to the decomposition of the covariance between query answers into smaller units, which we call inter-tuple covariances.

Let  $g$  be an aggregate function on attribute  $A_k$ , and  $\mathbf{t} = (a_1, \dots, a_l)$  be a vector of length  $l$  comprised of the values for  $r$ 's dimension attributes  $A_1, \dots, A_l$ . We define a function  $v_g(\mathbf{t})$  for every aggregate function  $g$  (e.g.,  $\text{AVG}(A_k)$ ,  $\text{FREQ}(\ast)$ ) such that, when integrated, it produces answers to query snippets. That is (omitting possible normalization and weight terms for simplicity):

$$\bar{\theta}_i = \int_{\mathbf{t} \in F_i} v_g(\mathbf{t}) d\mathbf{t} \quad (7)$$

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

$$\text{cov}(\theta_i, \theta_j) = \text{cov}(\bar{\theta}_i, \bar{\theta}_j) + \delta(i, j) \cdot \beta_i^2 \quad (8)$$

where  $\delta(i, j)$  returns 1 if  $i = j$  and 0 otherwise. Using equation (7) and the linearity of covariance, we can decompose  $\text{cov}(\bar{\theta}_i, \bar{\theta}_j)$  into:

$$\begin{aligned} \text{cov}(\bar{\theta}_i, \bar{\theta}_j) &= \text{cov}\left(\int_{\mathbf{t} \in F_i} v_g(\mathbf{t}) d\mathbf{t}, \int_{\mathbf{t}' \in F_j} v_g(\mathbf{t}') d\mathbf{t}'\right) \\ &= \int_{\mathbf{t} \in F_i} \int_{\mathbf{t}' \in F_j} \text{cov}(v_g(\mathbf{t}), v_g(\mathbf{t}')) d\mathbf{t} d\mathbf{t}' \end{aligned} \quad (9)$$

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.

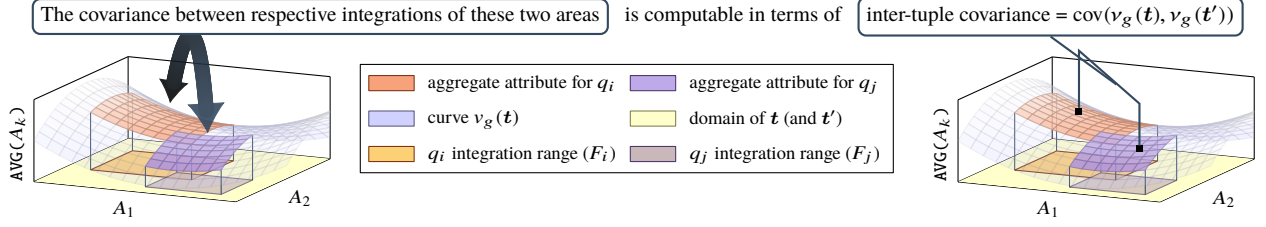
To use equation (9), we must be able to compute the inter-tuple covariance terms. However, computing these inter-tuple covariances is challenging, as we only have a single observation for each  $v_g(\mathbf{t})$ . Moreover, even if we had a way to compute the inter-tuple covariance for arbitrary  $\mathbf{t}$  and  $\mathbf{t}'$ , 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 (9), we propose using *analytical covariance functions*, a well-known technique in statistical literature for approximating covariances [13]. In particular, Verdict uses squared exponential

<sup>11</sup>This process is akin to hypothesis testing in statistics literatures [26].

<sup>12</sup>The AQP engine's answer can be treated as a random variable since it may differ depending on the random samples used.



**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 9).

covariance functions, which are proven to be *universal* [50], i.e., capable of approximating any target function arbitrarily closely as the number of observations (here, query answers) increases. The squared exponential covariance function  $\rho_g(\mathbf{t}, \mathbf{t}')$  is defined as:

$$\text{cov}(v_g(\mathbf{t}), v_g(\mathbf{t}')) \approx \rho_g(\mathbf{t}, \mathbf{t}') = \sigma_g^2 \cdot \prod_{i=1}^l \exp\left(-\frac{\Delta(a_i, a'_i)^2}{l_{g,i}^2}\right) \quad (10)$$

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  $i=1 \dots m$  and  $\sigma_g^2$  are tunable *correlation parameters* to be learned from past queries and their answers (Section 4.3).

Intuitively, when  $\mathbf{t}$  and  $\mathbf{t}'$  are similar, i.e.,  $\Delta(a_i, a'_i)$  is small for most  $A_i$ , then  $\rho_g(\mathbf{t}, \mathbf{t}')$  returns a larger value (closer to  $\sigma_g^2$ ), indicating that the expected values of  $g$  for  $\mathbf{t}$  and  $\mathbf{t}'$  are highly correlated. Since the double-integration of equation (10) is still an analytic expression, Verdict can efficiently compute the covariances between pairs of query answers in equation (9).

### 4.3 Optimal Correlation Parameters

In this section, we describe how to find the most likely values of the correlation parameters defined in Section 4.2. In this process, we exploit the joint pdf defined in equation (2), as it allows us to compute the likelihood of a certain combination of query answers given relevant statistics. Let  $\vec{\theta}_{\text{past}}$  denote a vector of raw answers to past snippets. 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, determining the most likely correlation parameters (which determine  $\Sigma$ ) given past queries amounts to finding the values for  $l_{g,1}, \dots, l_{g,m}, \sigma_g^2$  that maximize the below log-likelihood function:

$$\begin{aligned} \log \Pr(\vec{\theta}_{\text{past}} \mid \Sigma) &= \log f(\vec{\theta}_{\text{past}}) \\ &= -\frac{1}{2} \vec{\theta}_{\text{past}}^\top \Sigma^{-1} \vec{\theta}_{\text{past}} - \frac{1}{2} \log |\Sigma| - \frac{n}{2} \log 2\pi \end{aligned} \quad (11)$$

where  $f(\vec{\theta}_{\text{past}})$  is the joint pdf from equation (2), and  $\Sigma$  is the  $n \times n$  covariance matrix whose  $(i, j)$ -th entry is the (co)variance between  $\theta_i$  and  $\theta_j$ . The values of  $l_{g,1}, \dots, l_{g,m}$ , and  $\sigma_g^2$  that maximize the 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 Verdict's setting, since all 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 ac-

counted for by including additive error terms in their (co)variances (equation 8).

## 5. FORMAL GUARANTEES

Next, we formally show that the error bounds of Verdict's improved answers are never larger than the error bounds of the AQP engine's raw answers.

**Theorem 1.** Let Verdict's improved answer and improved error to the new snippet be  $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$  and the AQP engine's raw answer and raw error to the new snippet be  $(\theta_{n+1}, \beta_{n+1})$ . Then,

$$\hat{\beta}_{n+1} \leq \beta_{n+1}$$

and the equality occurs when the raw error is zero, or when Verdict's query synopsis is empty, or when Verdict's model-based answer is rejected by the model validation step.

*Proof.* Recall that  $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$  is set either to Verdict's model-based answer/error, i.e.,  $(\bar{\theta}_{n+1}, \bar{\beta}_{n+1})$ , or to the AQP engine's raw answer/error, i.e.,  $(\theta_{n+1}, \beta_{n+1})$ , depending on the result of the model validation. In the latter case, it is trivial that  $\hat{\beta}_{n+1} \leq \beta_{n+1}$ , and hence it is enough to show that  $\bar{\beta}_{n+1} \leq \beta_{n+1}$ .

Recall that computing  $\bar{\beta}_{n+1}$  involves an inversion of the covariance matrix  $\Sigma_{n+1}$ , where  $\Sigma_{n+1}$  includes the  $\beta_{n+1}$  term on one of its diagonal entries. We show  $\bar{\beta}_{n+1} \leq \beta_{n+1}$  by directly simplifying  $\bar{\beta}_{n+1}$  into the form that involves  $\beta_{n+1}$  and other terms.

We first define notations. Let  $\Sigma$  be the covariance matrix of the vector of random variables  $(\theta_1, \dots, \theta_{n+1}, \bar{\theta}_{n+1})$ ;  $\vec{k}_n$  be a column vector of length  $n$  whose  $i$ -th element is the  $(i, n+1)$ -th entry of  $\Sigma$ ;  $\Sigma_n$  be an  $n \times n$  submatrix of  $\Sigma$  that consists of  $\Sigma$ 's first  $n$  rows/columns;  $\bar{\kappa}^2$  be a scalar value at the  $(n+2, n+2)$ -th entry of  $\Sigma$ ; and  $\vec{\theta}_n$  be a column vector  $(\theta_1, \dots, \theta_n)^\top$ .

Then, we can express  $\vec{k}_{n+1}$  and  $\Sigma_{n+1}$  in equations (4) and (5) in block forms as follows:

$$\vec{k}_{n+1} = \begin{pmatrix} \vec{k}_n \\ \bar{\kappa}^2 \end{pmatrix}, \quad \Sigma_{n+1} = \begin{pmatrix} \Sigma_n & \vec{k}_n \\ \vec{k}_n^\top & \bar{\kappa}^2 + \beta_{n+1}^2 \end{pmatrix}, \quad \vec{\theta}_{n+1} = \begin{pmatrix} \vec{\theta}_n \\ \theta_{n+1} \end{pmatrix}$$

Here, it is important to note that  $\vec{k}_{n+1}$  can be expressed in terms of  $\vec{k}_n$  and  $\bar{\kappa}^2$  because  $(i, n+1)$ -th element of  $\Sigma$  and  $(i, n+2)$ -th element of  $\Sigma$  have the same values for  $i = 1, \dots, n$ . They have the same values because the covariance between  $\theta_i$  and  $\theta_{n+1}$  and the covariance between  $\theta_i$  and  $\bar{\theta}_{n+1}$  are same for  $i = 1, \dots, n$  due to equation (8).

Using the formula of block matrix inversion [37], we can obtain the following alternative forms of equations (4) and (5) (here, we assume zero means to simplify the expressions):

$$\gamma^2 = \bar{\kappa}^2 - \vec{k}_n^\top \Sigma_n^{-1} \vec{k}_n, \quad \theta = \vec{k}_n^\top \Sigma_n^{-1} \vec{\theta}_n \quad (12)$$

$$\bar{\theta}_{n+1} = \frac{\beta_{n+1}^2 \cdot \theta + \gamma^2 \cdot \theta_{n+1}}{\beta_{n+1}^2 + \gamma^2}, \quad \bar{\beta}_{n+1}^2 = \frac{\beta_{n+1}^2 \cdot \gamma^2}{\beta_{n+1}^2 + \gamma^2} \quad (13)$$



Note that  $\ddot{\beta}_{n+1}^2 < \beta_{n+1}$  for  $\beta_{n+1} > 0$  and  $\gamma^2 < \infty$ , and  $\ddot{\beta}_{n+1}^2 = \beta_{n+1}$  if  $\beta_{n+1} = 0$  or  $\gamma^2 \rightarrow \infty$ .  $\square$

Interestingly, the above theorem still holds even if the relation is subject to data additions (see Appendix B).

In proving the above theorem, we observe an interesting fact. In equations (12) and (13), the raw answer and the error to the new snippet are not required for computing the value of  $\gamma^2$  and  $\theta$ . This observation helps to keep Verdict’s inference process at query time fast, as described in the following lemma.

**Lemma 2.** Computing Verdict’s model-based answer/error (in equations (4) and (5)) can be computed in  $O(n^2)$  time at query time.

Verdict pre-computes the matrix inversion for  $\gamma^2$  and  $\theta$  in equation (12) to speed up online query processing time.

## 6. DEPLOYMENT SCENARIOS

Verdict is designed to support a large class of AQP engines. However, depending on the type of AQP engine used, Verdict may provide both speedup and error reduction, or only error reduction.

**1. AQP engines that support online aggregation** [31, 53, 73, 74]: 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), Verdict generates an improved answer with a higher accuracy (by paying small runtime overhead). As soon as this accuracy meets the user requirement, the online aggregation can be stopped. With Verdict, the online aggregation’s continuous processing will stop earlier than it would without Verdict. This is because Verdict reaches a target error bound much earlier by combining its model with the raw answer of the AQP engine.

**2. AQP engines that support time-bounds** [6, 16, 22, 33, 41, 57, 76]: Instead of continuously refining approximate answers and reporting them to the user, these engines simply take a time-bound from the user, and then they predict the largest sample size that they can process within the requested time-bound; thus, they minimize error bounds within the allotted time. For these engines, Verdict simply replaces the user’s original time bound  $t_1$  with a slightly smaller value  $t_1 - \epsilon$  before passing it down to the AQP engine, where  $\epsilon$  is the time needed by Verdict for inferring the improved answer and improved error. Thanks to the efficiency of Verdict’s inference,  $\epsilon$  is typically a small value, e.g., a few milliseconds (see Section 7.5). Since Verdict’s inference brings larger accuracy improvements on average compared to the benefit of processing more tuples within the  $\epsilon$  time, Verdict achieves significant error reductions over traditional AQP engines.

In this paper, we use an online aggregation engine to demonstrate Verdict’s both speedup and error reduction capabilities (Section 7). However, for interested readers, we also provide evaluations on a time-bound engine in Appendix A.4.

Note that some AQP engines also support error-bound queries but do not offer an online aggregation interface [6]. For these engine, Verdict currently only benefits their time-bound queries, leaving their answer to error-bound queries unchanged. Supporting the latter would require either adding an online aggregation interface to the AQP engine, or a tighter integration of Verdict and the AQP engine itself. Such modifications are beyond the scope of this paper, as one of our design goals is to treat the underlying AQP engine as a black box (Figure 2), so that Verdict can be used alongside a larger number of existing engines.

## 7. EXPERIMENTS

We conducted experiments to (i) quantify the percentage of real-world queries that benefit from Verdict (Section 7.2), (ii) study Verdict’s average speedup and error reductions over an AQP engine (Section 7.3), (iii) test the reliability of Verdict’s error bounds (Section 7.4), (iv) measure Verdict’s computational overhead and memory footprint (Section 7.5), and (v) study the impact of different workloads and data distributions on Verdict’s effectiveness (Section 7.6). In summary, our results indicated the following:

- Verdict supported a large fraction (73.7%) of aggregate queries in a real-world workload, and produced significant speedups (up to 23.0 $\times$ ) compared to a sample-based AQP solution.
- Given the same processing time, Verdict reduces the baseline’s approximation error on average by 75.8%–90.2%.
- Verdict’s runtime overhead was <10 milliseconds on average (0.02%–0.48% of total time) and its memory footprint was negligible.
- Verdict’s approach was robust against various workloads and data distributions.

We have also included supplementary experiments in Appendix A to demonstrate (vi) the benefits of model-based inference in comparison to a strawman approach that simply caches all past query answers (Appendix A.1), (vii) the effectiveness of Verdict’s model validation (Appendix A.2), and (viii) the accuracy of Verdict’s parameter learning process (Appendix A.3). We have also studied Verdict’s benefits using a time-bound AQP engine (Appendix A.4).

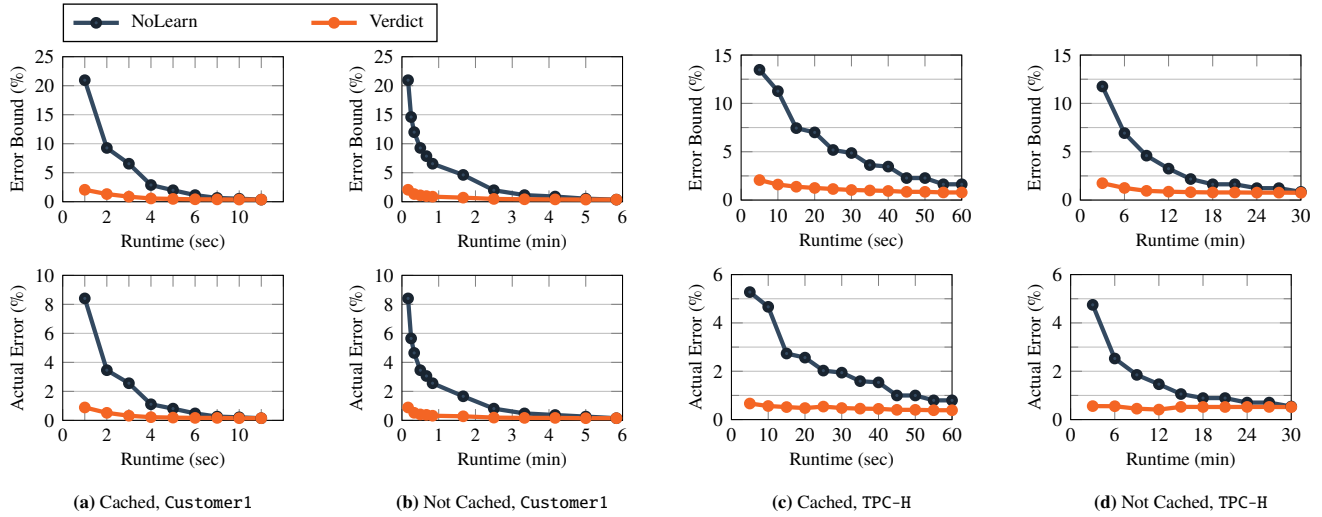
### 7.1 Experimental Setup

**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 are analytical queries supported by Spark SQL. We did not have the customer’s original dataset but had access to their data distribution, which we used to generate a 536 GB 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 100 GB. We generated a total of 500 queries using TPC-H’s workload generator with its default settings. The queries in this dataset include joins of up to 6 tables.
3. **Synthetic**: For more controlled experiments, we also generated large-scale synthetic datasets with different distributions (see Section 7.6 for details).

**Implementation**—For comparative analysis, we implemented two systems on top of Spark SQL [9] (version 1.5.1):

1. **NoLearn**: This system is an online aggregation engine that creates random samples of the original tables offline and splits them into multiple batches of tuples. To compute increasingly accurate answers to a new query, NoLearn first computes an approximate answer and its associated error bound on the first batch of tuples, and then continues to refine its answer and error bound as it processes additional batches. This is the same approach taken by many online aggregation engines [31, 53, 73, 74].
2. **Verdict**: This system is an implementation of our proposed approach, which uses NoLearn as its AQP engine. In other words, each time NoLearn yields a raw answer and error, Verdict



**Figure 6:** The relationship (i) between runtime and error bounds (top row), and (ii) between runtime and actual errors (bottom row), for both systems: NoLearn and Verdict.

Dataset	Total # of Queries with Aggregates	# of Supported Queries	Percentage
Customer1	3,342	2,463	73.7%
TPC-H	21	14	63.6%

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

computes an improved answer and error using our proposed approach. Naturally, Verdict incurs a runtime overhead but yields answers that are much more accurate in general.

**Experimental Environment—** We used a Spark cluster (for both NoLearn and Verdict) 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 [62] for Spark’s data loading. For experiments with cached datasets, we distributed Spark’s RDDs evenly across the nodes using Spark SQL DataFrame repartition function.

## 7.2 Generality of Verdict

To quantify the generality of our approach, we measured the coverage of our supported queries in practice. We analyzed the real-world SQL queries in Customer1. From the original 15.5K queries, Spark SQL was able to process 3.3K of the aggregate queries. Among those 3.3K queries, Verdict supported 2.4K queries, i.e., 73.7% of the analytical queries could benefit from Verdict. In addition, we analyzed the 21 TPC-H queries<sup>13</sup> and found 14 queries supported by Verdict. 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 Verdict can support a large class of analytical queries in practice. Next, we quantified the extent to which these supported queries benefitted from Verdict.

## 7.3 Speedup and Error Reduction

In this section, we first study the relationship between the processing time and the size of error bounds for both systems, i.e., NoLearn and Verdict. Based on this study, we then analyze Verdict’s speedup and error reductions over NoLearn.

In 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, the data had to be read 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 Verdict also learned its model parameters. Then, for the second half of the queries, we recorded both systems’ query processing times (i.e., runtime), approximate query answers, and error bounds. Since both NoLearn and Verdict are online aggregation systems, and Verdict produces improved answers for every answer returned from NoLearn, both systems naturally produced more accurate answers (i.e., answers with smaller error bounds) as query processing continued. Approximate query engines, including both NoLearn and Verdict, are only capable of producing expected errors in terms of error bounds. However, for analysis, we also computed the actual errors by comparing those approximate answers against the exact answers. In the following, we report their relative errors.

Figure 6 shows the relationship between runtime and average error bound (top row) and the relationship between runtime and average actual error (bottom row). Here, we also considered two cases: when the entire data is cached in memory and when it resides on SSD. In all experiments, the runtime-error graphs exhibited a consistent pattern: (i) Verdict produced smaller errors even when runtime was very large, and (ii) Verdict showed faster runtime for the same target errors. Due to the asymptotic nature of errors, achieving extremely accurate answers (e.g., less than 0.5%) required relatively long processing time even for Verdict.

Using these results, we also analyzed Verdict’s speedups and error reduction over NoLearn. For speedup, we compared how long each system took until it reached a target error bound. For error reduction, we compared the lowest error bounds that each system produced within a fixed allotted time. Table 4 reports the results for each combination of dataset and location (in memory or on SSD).

For the Customer1 dataset, Verdict achieved a larger speedup when the data was stored on SSD (up to 23.0×) compared to when it was fully cached in memory (7.7×). The reason was that, for cached data, the I/O time was no longer the dominant factor and Spark SQL’s default overhead (e.g., parsing the query and reading

<sup>13</sup>21 out of the 22 TPC-H queries include at least one aggregate function in their SQL.

	Cached?	Error Bound	Time Taken		Speedup
			NoLearn	Verdict	
Customer1	Yes	2.5%	4.34 sec	0.57 sec	7.7×
		1.0%	6.02 sec	2.45 sec	2.5×
	No	2.5%	140 sec	6.1 sec	23.0×
		1.0%	211 sec	37 sec	5.7×
TPC-H	Yes	4.0%	26.7 sec	2.9 sec	9.3×
		2.0%	34.2 sec	12.9 sec	2.7×
	No	4.0%	456 sec	72 sec	6.3×
		2.0%	524 sec	265 sec	2.1×

	Cached?	Runtime	Achieved Error Bound		Error Reduction
			NoLearn	Verdict	
Customer1	Yes	1.0 sec	21.0%	2.06%	90.2%
		5.0 sec	1.98%	0.48%	75.8%
	No	10 sec	21.0%	2.06%	90.2%
		60 sec	6.55%	0.87%	86.7%
TPC-H	Yes	5.0 sec	13.5%	2.13%	84.2%
		30 sec	4.87%	1.04%	78.6%
	No	3.0 min	11.8%	1.74%	85.2%
		10 min	4.49%	0.92%	79.6%

**Table 4:** Speedup and error reductions by Verdict compared to NoLearn.

the catalog) accounted for a considerable portion of the total data processing time. For TPC-H, on the contrary, the speedups were smaller when the data was stored on SSD. This difference stems from the different query forms between Customer1 and TPC-H. The TPC-H dataset includes queries that join several tables, some of which are large tables that were not sampled by NoLearn. (Similar to most sample-based AQP engines, NoLearn only samples fact tables, not dimension tables.) Consequently, those large tables had to be read each time NoLearn processed such a query. When the data resided on SSD, loading those tables became a major bottleneck that could not be reduced by Verdict (since they were not sampled). However, on average, Verdict still achieved an impressive 6.3× speedup over NoLearn. In general, Verdict’s speedups over NoLearn reduced as the target error bounds became smaller; however, even for 1% target error bounds, Verdict achieved an average of up to 5.7× speedup over NoLearn.

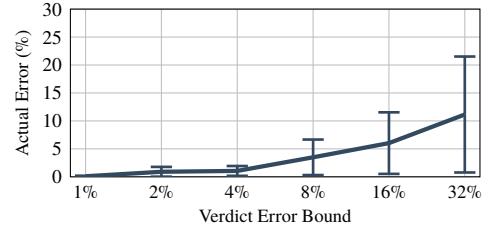
Table 4 also reports Verdict’s error reductions over NoLearn. For all target runtime budgets we examined, Verdict achieved massive error reductions compared to NoLearn.

The performance benefits of Verdict depends on several important factors, such as the accuracy of past query answers and workload characteristics. These factors are further studied in Section 7.6 and Appendix A.1.

## 7.4 Confidence Interval Guarantees

To confirm the validity of Verdict’s error bounds, we configured Verdict to produce error bounds at 95% confidence and compared them to the actual errors. We ran Verdict for an amount of time long enough to sufficiently collect error bounds of various sizes.

By definition, the error bounds at 95% confidence are probabilistically correct if the actual errors are smaller than the error bounds in at least 95% of the cases. Figure 7 shows the 5th percentile, median, and 95th percentile of the actual errors for different sizes of error bounds (from 1% to 32%). In all cases, the 95th percentile of the actual errors were lower than the error bounds produced by Verdict, which confirms the probabilistic correctness of Verdict’s error bound guarantees.



**Figure 7:** The comparison between Verdict’s error bound at 95% confidence and the actual error distribution (5th, 50th, and 95th percentiles are reported for actual error distributions).

Latency	Cached	Not Cached
NoLearn	2.083 sec	52.50 sec
Verdict	2.093 sec	52.51 sec
Overhead	0.010 sec (0.48%)	0.010 sec (0.02%)

**Table 5:** The runtime overhead of Verdict.

## 7.5 Memory and Computational Overhead

In this section, we study Verdict’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 only 150KB per query for Customer1, and only 8 KB per query for TPC-H. This is because Verdict only stores query answers and does not retain any of the input tuples.

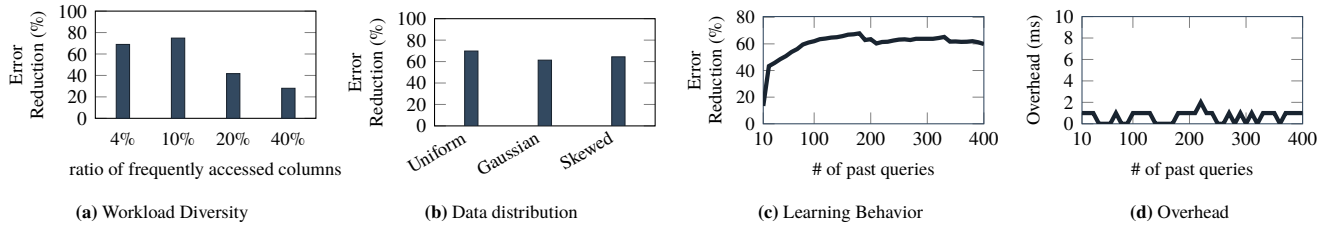
To measure Verdict’s runtime overhead, we recorded the time spent for its regular query processing (i.e., NoLearn) and the additional time spent for the inference and updating the final answer. As summarized in Table 5, the runtime overhead of Verdict was negligible compared to the overall query processing 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  $C_g=2,000$  by default; see Section 2.3.)

Besides the runtime overhead, we also measured Verdict’s offline overhead. Verdict’s offline processing includes the parameter learning (Section 4.3) and the pre-computation of matrix inversions (lemma 2). To learn its parameters, Verdict on average took 57.4 seconds each time 50 queries were processed, and took 380 milliseconds for its pre-computations. Note that these tasks occur after returning the answers to the user, and hence do not affect the query latencies. Nonetheless, even the offline overhead was quite small.

## 7.6 Impact of Different Data Distributions and Workload Characteristics

In this section, we generated various synthetic datasets and queries to fully understand how Verdict’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 proportions 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 accessed 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 proportion of frequently accessed columns was 20%, the first 20% of the columns (i.e., 10 columns) appeared with equal probability in each query, but the probability of appearance reduced by half for every remaining column. Figure 8(a) shows that as the pro-



**Figure 8:** The effectiveness of Verdict 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 Verdict’s overhead for different number of past queries.

portion of frequently accessed columns increased, Verdict’s relative error reduction over NoLearn gradually decreased (the number of past queries were fixed to 100). This is expected as Verdict constructs its model based on the columns appearing in the past. In other words, to cope with the increased diversity, more past queries are needed to understand the complex underlying distribution that generated 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, Verdict continued to learn more about the underlying distribution, and produced larger error reductions even when the workload was extremely diverse (Figure 8(c)).

Second, to study Verdict’s potential sensitivity, we generated three tables using three different probability distributions: uniform, Gaussian, and a log-normal (skewed) distribution. Figure 8(b) shows Verdict’s error reductions when queries were run against each table. Because of the power and generality of the maximum entropy principle taken by Verdict, it delivered a consistent performance irrespective of the underlying distribution.

Third, we varied the number of past queries observed by Verdict before running our test queries. For this study, we used a highly diverse query set (its proportion of frequently accessed columns was 20%). Figure 8(c) demonstrates that the error reduction continued increasing as more queries were processed, but its increment slowed down. This is because, after observing enough information, Verdict already had a good knowledge of the underlying distribution, and processing more queries barely improved its knowledge. This result indicates that Verdict is able to deliver reasonable performance without having to observe too many queries.

This is because, after observing enough information, Verdict already has a good knowledge of the underlying distribution, and processing more queries barely improves its knowledge. This result indicates that Verdict is able 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 Verdict’s overhead. Since Verdict’s inference consists of a small matrix multiplication, we did not observe a noticeable increase in its runtime overhead even when the number of queries in the query synopsis increased (Figure 8(d)).

## 8. RELATED WORK

**Approximate Query Processing**— There has been substantial work on sampling-based approximate query processing [3, 4, 6, 10, 16, 18, 25, 27, 32, 42, 49, 51, 58, 63, 69]. Some of these systems differ in their sample generation strategies. For instance, STRAT [16] and AQUA [4] create a single stratified sample, while BlinkDB creates samples based on *column sets*. Online Aggregation (OLA) [17, 31, 53, 71] continuously refines its answers during query execution. Others have focused on obtaining faster or more reliable error estimates [5, 72]. These are orthogonal to our work,

as reliable error estimates from an underlying AQP engine will also benefit DBL.

There is also AQP techniques developed for specific domain, e.g., sequential data [8, 19, 46, 55, 67], probabilistic data [28, 44, 52], and RDF data [34, 66]. However, our focus in this paper is on general (SQL-based) AQP engines.

**Adaptive Indexing, View Selection**— Adaptive Indexing and database cracking [35, 56] 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, 30, 40]. 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 [71] stores the results of past queries as multi-dimensional cubes, which 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 query processing. BayesDB [48] provides a SQL-like language that enables non-statisticians to declaratively use various statistical models.

## 9. 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 Verdict, on top of Spark SQL. Through extensive experiments on real-world and benchmark query logs, we demonstrated that Verdict supported 73.7% of real-world analytical queries, speeding them up by up to 23× compared to an online aggregation AQP engine.

Exciting lines of future work include: (i) the study of other inferential techniques for realizing database learning, (ii) the development of *active database learning*, whereby the engine itself proactively executes certain approximate queries that can best improve its internal model, and (iii) the extension of Verdict to support visual analytics [11, 20, 24, 36, 39, 43, 54, 68, 70].

## 10. REFERENCES

- [1] <https://db.apache.org/derby/docs/10.6/tuning/ctuntransform36368.html>.
- [2] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, 1999.

- [3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.
- [4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. In *SIGMOD*, 1999.
- [5] 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.
- [6] 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.
- [7] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.
- [8] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.
- [9] M. Armbrust et al. Spark sql: Relational data processing in spark. In *SIGMOD*, 2015.
- [10] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *VLDB*, 2003.
- [11] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. 2015.
- [12] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 1996.
- [13] C. M. Bishop. Pattern recognition. *Machine Learning*, 2006.
- [14] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In *Machine learning*. 1983.
- [15] 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.
- [16] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.
- [17] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, 2010.
- [18] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.
- [19] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 2005.
- [20] A. Crotty, A. Galakatos, E. Zraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *VLDB*, 2015.
- [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] M. El-Hindi, Z. Zhao, C. Binnig, and T. Kraska. Vistrees: fast indexes for interactive data exploration. In *HILDA*, 2016.
- [25] W. Fan, F. Geerts, Y. Cao, T. Deng, and P. Lu. Querying big data by accessing small data. In *PODS*, 2015.
- [26] D. Freedman, R. Pisani, and R. Purves. *Statistics*. 2007.
- [27] V. Ganti, M.-L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, 2000.
- [28] W. Gatterbauer and D. Suciu. Approximate lifted inference with probabilistic databases. *PVLDB*, 2015.
- [29] G. Graefe and H. Kuno. Adaptive indexing for relational keys. In *ICDEW*, 2010.
- [30] A. Y. Halevy. Answering queries using views: A survey. *VLDBJ*, 2001.
- [31] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.
- [32] K. Hose, D. Klan, and K.-U. Sattler. Distributed data summaries for approximate query processing in pdms. In *IDEAS*, 2006.
- [33] Y. Hu, S. Sundara, and J. Srinivasan. Estimating Aggregates in Time-Constrained Approximate Queries in Oracle. In *EDBT*, 2009.
- [34] H. Huang, C. Liu, and X. Zhou. Approximating query answering on rdf databases. *WWW*, 2012.
- [35] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *CIDR*, 2007.
- [36] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, 2015.
- [37] J. S. R. Jang. General formula: Matrix inversion lemma. <http://www.cs.nthu.edu.tw/~jang/book/addenda/matinv/matinv/>.
- [38] Y. Jia. Running tpc-h queries on hive. <https://issues.apache.org/jira/browse/HIVE-600>.
- [39] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Interactive data exploration with smart drill-down. In *ICDE*, 2016.
- [40] S. Joshi and C. Jermaine. Materialized sample views for database approximation. *TKDE*, 2008.
- [41] S. Joshi and C. Jermaine. Sampling-Based Estimators for Subset-Based Queries. *VLDB J.*, 18(1), 2009.
- [42] 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.
- [43] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 2015.
- [44] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Approximation trade-offs in markovian stream processing: An empirical study. In *ICDE*, 2010.
- [45] M. Lichman. UCI machine learning repository, 2013.
- [46] X. Lin, J. Xu, Q. Zhang, H. Lu, J. X. Yu, X. Zhou, and Y. Yuan. Approximate processing of massive continuous quantile queries over high-speed data streams. *TKDE*, 2006.
- [47] M. Lovric. *International Encyclopedia of Statistical Science*. Springer, 2011.
- [48] V. Mansinghka et al. Bayesdb: A probabilistic programming system for querying the probable implications of data. *arXiv*, 2015.
- [49] A. Meliou, C. Guestrin, and J. M. Hellerstein. Approximating sensor network queries using in-network summaries. In *IPSN*, 2009.
- [50] C. A. Micchelli, Y. Xu, and H. Zhang. Universal kernels. *JMLR*, 2006.
- [51] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed. Interactive Analysis of Web-Scale Data. In *CIDR*, 2009.
- [52] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.
- [53] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4, 2011.
- [54] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. *ICDE*, 2016.
- [55] K. S. Perera, M. Hahmann, W. Lehner, T. B. Pedersen, and C. Thomsen. Efficient approximate olap querying over time series. In *IDEAS*, 2016.
- [56] E. Petraki, S. Idreos, and S. Manegold. Holistic indexing in main-memory column-stores. In *SIGMOD*, 2015.
- [57] A. Pol and C. Jermaine. Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, 2005.
- [58] N. Potti and J. M. Patel. Daq: a new paradigm for approximate query processing. *PVLDB*, 2015.
- [59] F. Rusu, C. Qin, and M. Torres. Scalable analytics model calibration with online aggregation. *IEEE Data Eng. Bull.*, 2015.
- [60] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959.
- [61] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, 2000.
- [62] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *MSST*, 2010.
- [63] L. Sidiropoulos, M. L. Kersten, and P. A. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *CIDR*, 2011.
- [64] A. Silberschatz, H. F. Korth, S. Sudarshan, et al. *Database system concepts*. 1997.
- [65] J. Skilling. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2006.
- [66] A. Souihli and P. Senellart. Optimizing approximations of dnf query lineage in probabilistic xml. In *ICDE*, 2013.
- [67] Y. Tao, D. Papadias, and C. Faloutsos. Approximate temporal aggregation. In *ICDE*, 2004.
- [68] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 2015.
- [69] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, 2014.
- [70] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. *PVLDB*, 2014.
- [71] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous Sampling for Online Aggregation over Multiple Queries. In *SIGMOD*, pages 651–662, 2010.
- [72] F. Xu, C. Jermaine, and A. Dobra. Confidence bounds for sampling-based group by estimates. *TODS*, 2008.
- [73] 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.
- [74] K. Zeng, S. Agarwal, and I. Stoica. iolap: Managing uncertainty for efficient incremental olap. 2016.
- [75] K. Zeng, S. Gao, J. Gu, B. Mozafari, and C. Zaniolo. Abs: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, 2014.
- [76] 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

In appendix, we have the following materials:

1. Appendix A includes additional experiments for demonstrating the effectiveness and accuracy of Verdict.



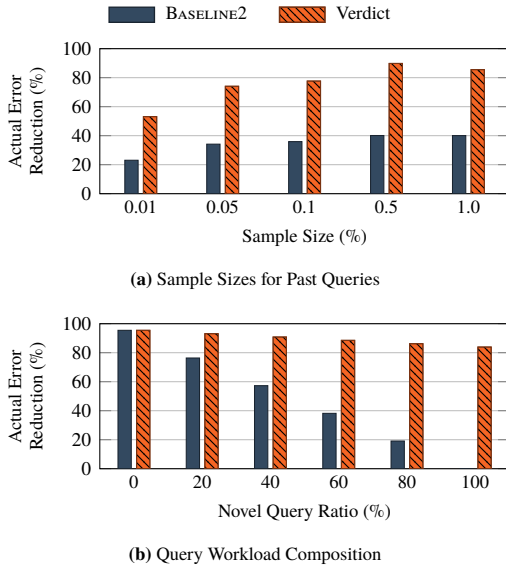
2. In Appendix B, we present Verdict’s approach to supporting a database with appending data.
3. In Appendix C, we analyze the inter-tuple covariances (Section 4.2) present in real-world datasets.

## A. ADDITIONAL EXPERIMENTS

This section contains additional experiments we have not included in the main part of our paper due to space limitations. Concretely, we describe the following:

1. We study the impact of two factors that affect the performance benefits of Verdict (Appendix A.1).
2. We demonstrate that Verdict’s model validation step could effectively prevent the answers with incorrect error bounds (Appendix A.2).
3. Recall that, in Section 4.3, we presented Verdict’s approach to estimating the correlation parameters from the answers to past snippets. In Appendix A.3, we empirically study the effectiveness of Verdict’s parameter estimation process.

### A.1 Verdict vs. Simple Answer Caching



**Figure 9:** (a) Comparison of Verdict and BASELINE2 for different sample sizes used by past queries and (b) comparison of Verdict and BASELINE2 for different ratios of novel queries in the workload.

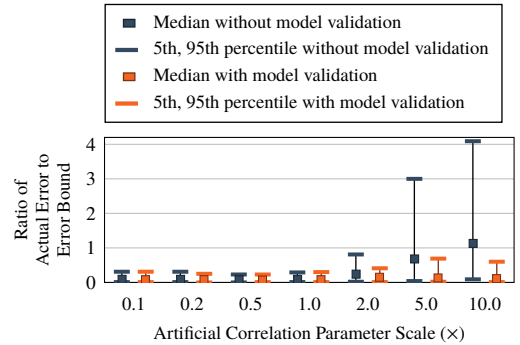
To study the benefits of Verdict’s model-based inference, we consider another system BASELINE2, and make comparisons between Verdict and BASELINE2, using the TPC-H dataset. BASELINE2 is similar to NoLearn but returns a cached answer if the new query is identical to one of the past ones. When there are multiple instances of the same query, BASELINE2 caches the one with the lowest expected error.

Figure 9(a) reports the average actual error reductions of Verdict 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, Verdict 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 Verdict and BASELINE2 by changing the ratio of novel queries in the workload. Understandably, both Verdict and BASELINE2 were more effective for workloads with fewer novel queries (i.e., more repeated queries); however, Verdict was also effective for workloads with many novel queries.

### A.2 Effect of Model Validation

This section studies the effect of Verdict’s model validation described in Section 3.5. For this study, we first generated synthetic datasets with several predetermined correlation parameters values. Note that one can generate such synthetic datasets by first determining a joint probabilistic distribution function with predetermined correlation parameter values and sampling attribute values from the joint probability distribution function. In usual usage scenario, Verdict estimates those correlation parameters from past snippets; however, in this section, we manually set the values for the correlation parameters in Verdict’s model, to test the behavior of Verdict running with possibly incorrect correlation parameter values.



**Figure 10:** Effect of model validation. For Verdict’s error bounds to be correct, the 95th percentile should be below 1.0. One can find that, with Verdict’s model validation, the improved answers and the improved errors were probabilistically correct even when largely incorrect correlation parameters were used.

Figure 10 report the experiment results when Verdict was tested without a model validation step and with a model validation step, respectively. In the figure, the value on the X-axis indicates the value multiplied to the true correlation parameters before setting the values in Verdict’s model. For instance, if a true correlation parameter was 5.0, and “artificial correlation parameter scale” was 0.2, Verdict’s model was set 1.0 for the correlation parameter. Thus, the values of correlation parameters were the same as the true correlation parameters when “artificial correlation parameter scale” was 1.0. Since the Y-axis is reporting the ratio of the actual error to Verdict’s error bound, Verdict’s error bound was correct when the value on the Y-axis is below 1.0.

In the figure, one can observe that, Verdict used without the model validation produced incorrect error bounds when the correlation parameters used for the model deviated largely from the true correlation parameter values. However, Verdict’s model validation could successfully identify incorrect model-based answers and provide correct error bounds by replacing those incorrect model-based answers with the raw answers computed by the AQP engine.

### A.3 Accuracy of Parameter Learning

In this section, we examine the effectiveness of Verdict’s correlation parameter estimation process presented in Section 4.3. Similar to Appendix A.2, we used the synthetic datasets generated from predetermined correlation parameters to see how close Verdict could

estimate the values of those correlation parameters. We let Verdict estimate the correlation parameter values using three different numbers of past snippets (20, 50, and 100) for various datasets with different correlation parameter values.

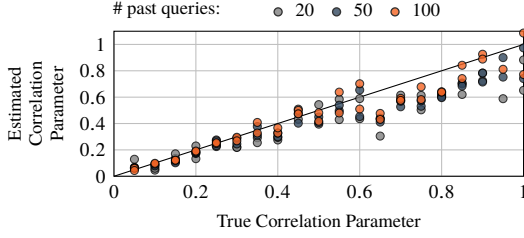


Figure 11: Correlation Param Learning

Figure 11 shows the results. In general, the correlation parameter values discovered by Verdict’s estimation process were consistent with the true correlation parameter values. Also, the estimated values tended to be closer to the true values when a larger number of past snippets were used for the estimation process. This result indicates that Verdict can effectively learn statistical characteristics of the underlying distribution just based on the answers to the past queries.

#### A.4 Error Reductions for Time-Bound AQP Engines

Recall that, in Section 7, we demonstrated Verdict’s speedup and error reductions over an online aggregation system. In this section, we show Verdict’s error reductions over a time-bound AQP system. First, we describe our experiment setting in Appendix A.4. Next, we present our experiment results in Appendix A.4.

**Setup**—Here, we describe two systems, NoLearn and Verdict, that we compare in this section:

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 [2, 6, 16, 57, 59, 75]. Specifically, NoLearn maintains uniform random samples created offline (10% of the original tables), and uses the largest samples that are small enough to satisfy the requested time bounds.
2. Verdict: This system invokes NoLearn to obtain raw answers/errors but modifies them to produce improved answers/errors using our proposed inference process. Verdict translates the user’s requested time bound into an appropriate time bound for NoLearn.

Observe that we are abusing the term NoLearn here to indicate a time-bound AQP system in this section. (In Section 7, we used NoLearn for an online aggregation system.)

**Experiment Results**—This section presents the error reduction by Verdict compared to NoLearn. For experiments, we ran the same set of queries as in Section 7 with both Verdict and NoLearn described above. For comparison, we used the identical time-bounds for both Verdict and NoLearn. Specifically, we set the time-bounds as 2 seconds and 0.5 seconds for the Customer1 and TPC-H datasets cached in memory, respectively; and set the time-bounds as 5.0 seconds for both Customer1 and TPC-H datasets loaded from SSD. Figure 12 reports Verdict’s error reductions over NoLearn for each of four different combinations of dataset and cache setting.

In Figure 12, one can observe that Verdict achieved large error reductions (63%–86%) over NoLearn. These results indicate that

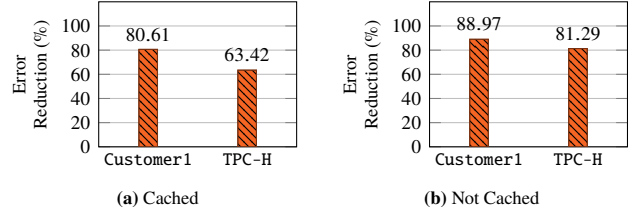


Figure 12: Average error reduction by Verdict (compared to NoLearn) for the same time budget.

the users of Verdict can obtain much more precise answers compared to the users of NoLearn within the same time-bounds.

## B. GENERALIZATION OF VERDICT UNDER DATA ADDITIONS

Thus far, we have discussed our approach based on the assumption that the database is static, i.e., no tuples are deleted, added, or updated. In this section, we suggest the possibility of using Verdict even for the database that allows an important kind of data update: data append.

### B.1 Our Method

A naïve strategy to supporting tuples insertions 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, Verdict 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.

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

We represent the possible difference between  $A_k$ ’s values in  $r$  and those in  $r^a$  by a random variable  $s_k$  with mean  $\mu_k$  and variance  $\eta_k^2$ . Thus:

$$\bar{\theta}_i^a = \bar{\theta}_i + s_k$$

The values of  $\mu_k$  and  $\eta_k^2$  can be estimated using small samples of  $r$  and  $r^a$ . Verdict uses the following lemma to update the raw answer and raw error for  $q_i$ :

**Lemma 3.**

$$E[\bar{\theta}_i^u - \theta_i] = \mu_k \cdot \frac{|r^a|}{|r| + |r^a|}$$

$$E[(\bar{\theta}_i^u - \theta_i - \frac{|r^a| \mu_k}{|r| + |r^a|})^2] = \beta_i^2 + \left( \frac{|r^a| \eta_k}{|r| + |r^a|} \right)^2$$

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

*Proof.* Since we represented a snippet answer on the appended relation using a random variable  $\bar{\theta}_i^u$ , we can also represent a snippet answer on the updated relation  $r^u$  using a random variable. Let the snippet answer on  $r^u$  be  $\bar{\theta}_i^u$ . Then,

$$E[\bar{\theta}_i^u - \theta_i] = E \left[ \frac{|r| \bar{\theta}_i}{|r| + |r^a|} + \frac{|r^a| s_k}{|r| + |r^a|} \right] - \bar{\theta}_i = \frac{|r^a| \mu_k}{|r| + |r^a|}$$

Also,

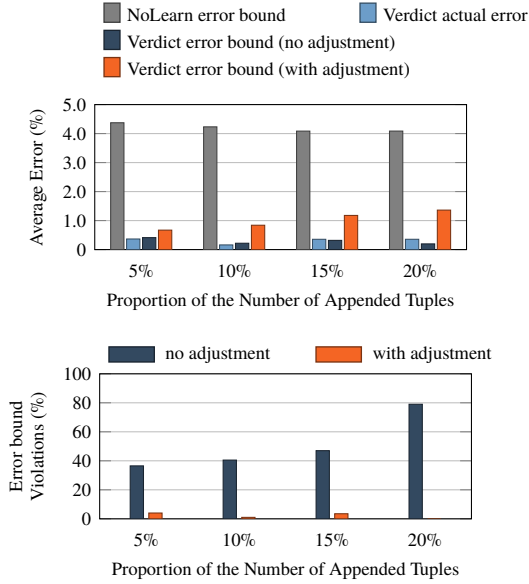
$$\begin{aligned}
& E \left[ (\bar{\theta}_i^u - \theta_i - \frac{|r^a| \mu_k}{|r| + |r^a|})^2 \right] \\
&= E \left[ \left( \bar{\theta}_i + \frac{|r^a| s_k}{|r| + |r^a|} - \theta_i - \frac{|r^a| \mu_k}{|r| + |r^a|} \right)^2 \right] \\
&= E \left[ (\bar{\theta}_i - \theta_i)^2 + \left( \frac{|r^a|}{|r| + |r^a|} \right)^2 (s_k - \mu_k)^2 \right. \\
&\quad \left. + 2 \left( \frac{|r^a|}{|r| + |r^a|} \right) (\bar{\theta}_i - \theta_i)(s_k - \mu_k) \right] \\
&= \beta_i^2 + \left( \frac{|r^a| \eta_k}{|r| + |r^a|} \right)^2
\end{aligned}$$

where we used to independence between  $(\bar{\theta}_i - \theta_i)$  and  $(s_k - \mu_k)$ .  $\square$

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

## B.2 Empirical Evaluation

In this section, we empirically study the impact of new data (i.e., tuple insertions) on Verdict’s effectiveness. We generated an initial synthetic table with 5 million 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 5 million) tuples were added. We then ran the queries and recorded the error bounds of VERDICTADJUST and VERDICTNOADJUST (our approach with and without the technique introduced in Appendix B.1). We also measured the error bounds of NoLearn and the actual error.



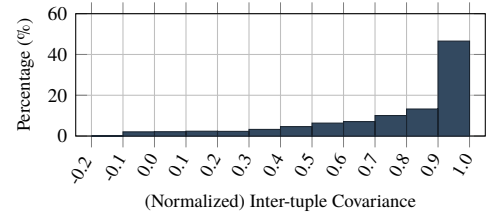
**Figure 13:** Data append technique (Appendix B) is highly effective in delivering correct error estimates in face of new data.

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

## C. PREVALENCE OF INTER-TUPLE CO-VARIANCES IN REAL-WORLD

Recall that, in Section 2.4, Verdict exploits the possible correlations between pairs of attribute values by using inter-tuple covariances. In this section, we demonstrate the existence of the inter-tuple covariances in many of real-world datasets by analyzing well-known datasets from the UCI repository [45]. We analyzed the following well-known 16 datasets: cancer, glass, haberman, ionosphere, iris, mammographic-masses, optdigits, parkinsons, pima-indians-diabetes, segmentation, spambase, steel-plates-faults, transfusion, vehicle, vertebral-column, and yeast [45].

We extracted numeric attributes (or equivalently, columns) from those datasets and composed each of the datasets into a relational table. Suppose a dataset has  $m$  attributes. Then, we computed the correlation between adjacent attribute values in the  $i$ -th column when the column is sorted in order of another  $j$ -th column— $i$  and  $j$  are the values (inclusively) between 1 and  $m$ , and  $i \neq j$ . Note that there are  $m(m-1)/2$  number of pairs of attributes for a dataset with  $m$  attributes. We analyzed all of those pairs for each of 16 datasets listed above.



**Figure 14:** Inter-tuple Covariances for 16 real-life UCI datasets.

Figure 14 shows the results of our analysis. The figure reports the percentage of different levels of correlations (or equivalently, normalized inter-tuple covariances) between adjacent attributes. One can observe that there existed strong correlations in the datasets we analyzed. Remember that the users of Verdict do not need to provide any information regarding the inter-tuple covariances; Verdict automatically detects them as described in Section 4.3 relying on the past snippet answers stored in the query synopsis.