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 Verdict 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, 17, 28, 30, 62, 80]), 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.*

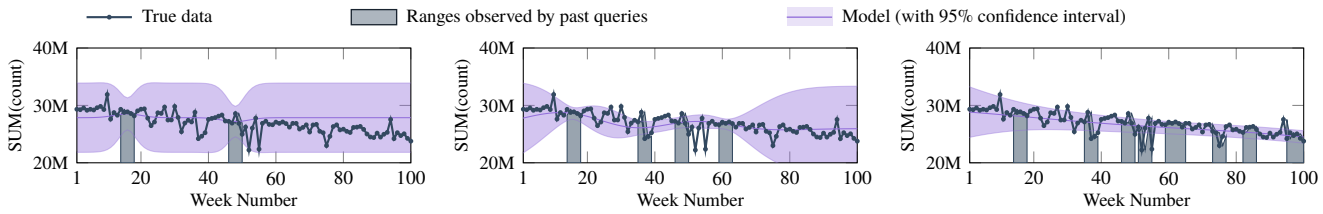
SIGMOD'17, May 14-19, 2017, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3035918.3064013

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 Leaning (ML) whereby past observations are used to improve future predictions [15,16,66]. 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



(a) After 2 Queries **(b)** After 4 Queries **(c)** After 8 Queries

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[1] 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 [26,32,61] or materialized views [8] 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 about 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 [32]) 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 are only beneficial when there is a strict structural compatibility—such as query containment or equality—between past and new queries [27].

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

1. Since materialized views, indexing, and caching are for exact query processing, they are only effective when new queries touch previously accessed ranges. On the contrary, DBL works in AQP settings; thus, DBL can benefit new queries even if they query ranges 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. In contrast, indices and materialized views grow in size as the data grows, while DBL's storage requirement remains *oblivious to the data size* (see Section 8 for a detailed discussion).

Our Approach— Our vision of database learning (DBL) [47] 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* [71]. 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.
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.

[1] In general, DBL does not make any *a prior* 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.

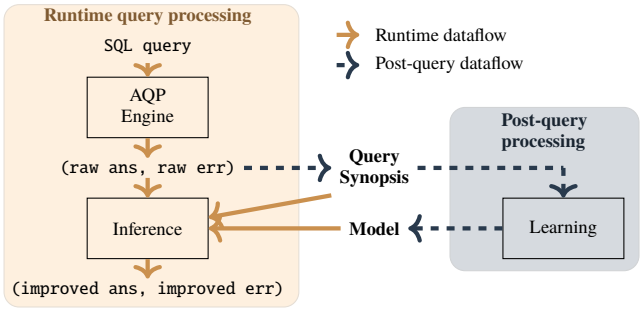


Figure 2: Workflow in Verdict. At query time, the Inference module uses the *Query Synopsis* and the *Model* to improve the query answer and error computed by the underlying AQP engine (i.e., *raw answer/error*) before returning them to the user. Each time a query is processed, the raw answer and error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

The rest of this paper is organized as follows. Section 2 overviews Verdict's workflow, supported query types, and internal query representations. 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, and Section 7 reports our empirical results. Section 8 discusses 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.

We begin by defining *query snippets*, which serve as the basic units of inference in Verdict.

Definition 1. (Query Snippet) A query snippet is a *supported* SQL query whose answer is a *single scalar value*, where supported queries are formally defined in Section 2.2.

Section 2.3 describes how a supported query (whose answer may be a set) is decomposed into possibly multiple query snippets. For simplicity, and without loss of generality, here we assume that every incoming query is a query snippet.

For the i-th query snippet q_i, the AQP engine's answer includes a pair of an approximate answer θ_i and a corresponding expected error β_i. θ_i and β_i are formally defined in Section 3.1, and are produced by most AQP systems [6, 28, 55, 79, 80, 82]. 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: $\{(q_i, \theta_i, \beta_i) \mid i = 1, \ldots, n\}$.

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

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

Table 1: Terminology.

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 θ_{n+1} and a raw error β_{n+1}. Then, Verdict combines this raw answer/error and the previously computed model to *infer* an *improved answer* $\widehat{\theta}_{n+1}$ and an associated expected error $\widehat{\beta}_{n+1}$, 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**. Verdict supports foreign-key joins between a fact table[2] and any number of dimension tables, exploiting the fact that this type of join does not introduce a sampling bias [3]. 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 numeric 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. Note that the underlying AQP engine may affect the cardinality of the result set depending on the `having` clause (e.g., subset/superset error). Verdict simply operates on the result set returned by the AQP engine.

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 [27]. In fact, this

[2]Data warehouses typically record measurements (e.g., sales) into fact tables and normalize commonly appearing attributes (e.g., seller information) into dimension tables [70].

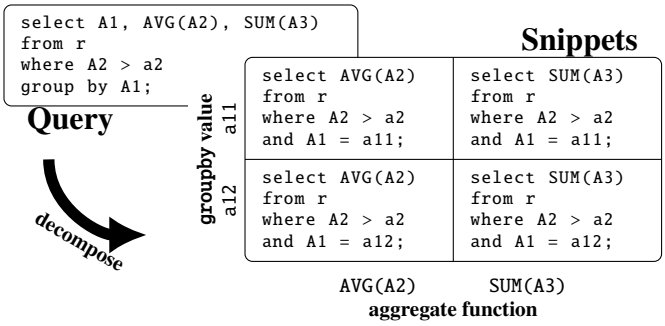


Figure 3: Example of a query's decomposition into multiple snippets.

is the process used by Hive for supporting the nested queries of the TPC-H benchmark [35]. 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. Verdict computes improved answers only for those snippets in order to bound the computational overhead.

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

2.4 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.** $S_i \cap S_j \neq \phi$ implies 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 $S_i \cap S_j \neq \phi$. In practice, most real-life datasets tend to have non-zero inter-tuple covariances, i.e., correlated attribute values (see Appendix D 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.

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 Appendix B.

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. Verdict's inference process follows the standard machine learning arguments: we can understand in part the true distribution by means of observations, then we apply our understanding to predicting the unobserved. To this end, Verdict applies well-established techniques, such as the principle of maximum entropy and kernel-based estimations, to an AQP setting.

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

Table 2: Mathematical Notations.

To present our approach, we first 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 and 3.4 present the details of the Verdict's approach to solving the problem. Section 3.5 discusses some challenges in applying Verdict's approach.

3.1 Problem Statement

Let r be a relation drawn from some unknown underlying distribution. r can be a join or Cartesian product of multiple tables. Let r's attributes be $A_1, \ldots, A_m$, where $A_1, \ldots, A_l$ are the *dimension attributes* and $A_{l+1}, \ldots, A_m$ are the *measure attributes*. Dimension attributes cannot appear inside aggregate functions while measure attributes can. Dimension attributes can be numeric or categorical, but measure attributes are numeric. 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 θ_i along with an associated expected error β_i. Formally, β_i^2 is the expectation of the squared deviation of θ_i from the (unknown) exact answer $\bar{\theta}_i$ to q_i.[3] β_i and β_j are independent if $i \neq j$.

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., $\texttt{AVG}(A_k)$), where A_k is one of the measure attributes. Our problem is then stated as follows: *given Q_n and (θ_{n+1}, β_{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. The generalization of Verdict under data updates is presented in our extended report [59].

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. Using random variables to represent degrees of belief is a standard approach in Beyesian inference. Specifically, we denote our knowledge of the raw answer and the exact answer to the i-th query snippet by random variables $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_i$, respectively. At this step, the only information available to us regarding $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_i$ is that θ_i is an instance of $\boldsymbol{\theta}_i$; no other assumptions are made.

Next, we represent the relationship between the set of random variables $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$ 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 $f(\boldsymbol{\theta}_1 = \theta'_1, \ldots, \boldsymbol{\theta}_{n+1} = \theta'_{n+1}, \bar{\boldsymbol{\theta}}_{n+1} = \bar{\theta}'_{n+1})$. For brevity, we also use $f(\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1})$ when the meaning is clear from the context. (Recall that θ_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 $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$ takes a particular combination of the values, i.e., $\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1}$. 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 $\bar{\boldsymbol{\theta}}_{n+1}$, by first conditionalizing the joint pdf on the actual observations (i.e., raw answers) from the AQP engine, i.e., $f(\bar{\boldsymbol{\theta}}_{n+1} = \bar{\theta}'_{n+1} \mid \boldsymbol{\theta}_1 = \theta_1, \ldots, \boldsymbol{\theta}_{n+1} = \theta_{n+1})$. 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 $\ddot{\theta}_{n+1}$. Section 3.4 provides more details of this process. Finally, $\ddot{\theta}_{n+1}$ and its associated expected error $\ddot{\beta}_{n+1}$ are returned as Verdict's improved answer and improved error if they pass the model validation (described in Appendix B). 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 $f(\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1})$ that represents its knowledge of the underlying distribution. 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) [13, 71], a simple but powerful statistical tool for determining a pdf of random variables given some statistical information available. 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*:

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

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

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

[3] Here, the expectation is made over θ_i since the value of θ_i depends on samples.

principle. On one hand, the resulting pdf can be computed more efficiently if the provided statistics are simple or few, i.e., simple statistics reduce the computational complexity. On the other hand, the resulting pdf can describe the relationship among the random variables more accurately if richer statistics are provided, i.e., the richer the statistics, the 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 [71].

Lemma 1. Let $\vec{\boldsymbol{\theta}} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})^\intercal$ be a vector of $n+2$ random variables with mean values $\vec{\mu} = (\mu_1, \ldots, \mu_{n+1}, \bar{\mu}_{n+1})^\intercal$ and a $(n+2)\times(n+2)$ covariance matrix Σ 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:

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

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 Appendix B, this model-based answer is chosen as an improved answer if it passes a model validation. Finally, in Section 3.5, we discuss the challenges involved in obtaining $\vec{\mu}$ and Σ, 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 $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$, using a joint pdf. In this section, we exploit this joint pdf to infer the most likely answer to the new snippet. In other words, we aim to find the most likely value for $\bar{\boldsymbol{\theta}}_{n+1}$ (the random variable representing q_{n+1}'s exact answer), given the observed values for $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$, 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 $\ddot{\theta}_{n+1}$ to q_{n+1} can be expressed as:

$$\ddot{\theta}_{n+1} = \operatorname*{Arg\,Max}_{\bar{\theta}'_{n+1}} f(\bar{\theta}'_{n+1} \mid \boldsymbol{\theta}_1 = \theta_1, \ldots, \boldsymbol{\theta}_{n+1} = \theta_{n+1}). \quad (3)$$

That is, $\ddot{\theta}_{n+1}$ is the value at which the conditional pdf has its maximum value. The conditional pdf, $f(\bar{\theta}'_{n+1} \mid \theta_1, \ldots, \theta_{n+1})$, 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 μ_c and variance σ_c^2 [14]:

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

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

where:

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

Since the mean of a normal distribution is the value at which the pdf takes a maximum value, we take μ_c as our model-based answer $\ddot{\theta}_{n+1}$. 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 $\ddot{\theta}_{n+1}$ coincides with the variance σ_c^2 of the conditional pdf. Thus, we take σ_c as our model-based error $\ddot{\beta}_{n+1}$.

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). As a future work, we plan to employ inferential techniques with sub-linear time complexity [42, 75] as well as a more sophisticated eviction policy for past queries.

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

3.5 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 $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$. However, acquiring these statistics is a non-trivial task for two reasons. First, we have only observed one value for each of the random values $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$, namely $\theta_1, \ldots, \theta_{n+1}$. 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 $\bar{\boldsymbol{\theta}}_{n+1}$ (recall that $\bar{\boldsymbol{\theta}}_{n+1}$ represents our knowledge of the exact answer to the new snippet, i.e., $\bar{\theta}_{n+1}$). 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 $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$. In this process, we need to know the means, variances, and covariances of these random variables.

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

Thus, in the rest of this section, we focus on obtaining the variances and covariances of these random variables, which are the elements of the $(n+2) \times (n+2)$ covariance matrix Σ in Lemma 1 (thus, we can obtain the elements of the column vector $\vec{k}_{n+1}$ and the variance $\bar{\kappa}^2$ as well). Note that, due to the independence between expected errors, we have:

$$\begin{aligned} \text{cov}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) &= \text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) + \delta(i,j) \cdot \beta_i^2 \\ \text{cov}(\boldsymbol{\theta}_i, \bar{\boldsymbol{\theta}}_j) &= \text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) \end{aligned} \tag{6}$$

where $\delta(i,j)$ returns 1 if $i = j$ and 0 otherwise. Thus, computing $\text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ is sufficient for obtaining Σ.

Computing $\text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ 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.

In this work, we further extend the above observation. That is, if we are able to compute the covariance between the average revenues at an infinitesimal time t and another infinitesimal time t', we will be able to compute the covariance between (i) the average revenue of 2014–2015 and (ii) the average revenue of 2015–2016, by integrating the covariances between the revenues at infinitesimal times over appropriate ranges. Here, the covariance between the average revenues at two infinitesimal times t and t' is defined in terms of the underlying data distribution that has generated the relation r, where the past query answers help us discover the most-likely underlying distribution. The rest of this section formalizes 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.

4.1 Covariance Decomposition

To compute the variances and covariances between query snippet answers (i.e., $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$), 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 $\boldsymbol{t} = (a_1, \ldots, a_l)$ be a vector of length l comprised of the values for r's dimension attributes $A_1, \ldots, A_l$. To help simplify the mathematical descriptions in this section, we assume that all dimension attributes are numeric (not categorical), and the selection predicates in queries may contain range constraints on some of those dimension attributes. Handling categorical attributes is a straightforward extension of this process (see [59]).

We define a continuous function $\nu_g(\boldsymbol{t})$ for every aggregate function g (e.g., `AVG(`A_k`)`, `FREQ(*)`) such that, when integrated, it produces answers to query snippets. That is (omitting possible normalization and weight terms for simplicity):

$$\bar{\boldsymbol{\theta}}_i = \int_{\boldsymbol{t} \in F_i} \nu_g(\boldsymbol{t})\, d\boldsymbol{t} \tag{7}$$

Formally, F_i is a subset of the Cartesian product of the domains of the dimension attributes, $A_1, \ldots, A_l$, such that $\boldsymbol{t} \in F_i$ satisfies the selection predicates of q_i. Let $(s_{i,k}, e_{i,k})$ be the range constraint for A_k specified in q_i. We set the range to $(\min(A_k),\ \max(A_k))$ if no constraint is specified for A_k. Verdict simply represents F_i as the product of those l per-attribute ranges. Thus, the above Equation (7) can be expanded as:

$$\bar{\boldsymbol{\theta}}_i = \int_{s_{i,l}}^{e_{i,l}} \cdots \int_{s_{i,1}}^{e_{i,1}} \nu_g(\boldsymbol{t})\, da_1 \cdots da_l$$

For brevity, we use the single integral representation using F_i unless the explicit expression is needed.

Using Equation (7) and the linearity of covariance, we can decompose $\text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ into:

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

As a result, the covariance between query answers can be broken into an integration of the covariances between tuple-level function values, which we call *inter-tuple covariances*.

To use Equation (8), we must be able to compute the inter-tuple covariance terms. However, computing these inter-tuple covariances is challenging, as we only have a single observation for each $\nu_g(\boldsymbol{t})$. Moreover, even if we had a way to compute the inter-tuple covariance for arbitrary $\boldsymbol{t}$ and $\boldsymbol{t}'$, the exact computation of Equation (8) would still require an infinite number of inter-tuple covariance computations, which would be infeasible. In the next section, we present an efficient alternative for estimating these inter-tuple covariances.

4.2 Analytic Inter-tuple Covariances

To efficiently estimate the inter-tuple covariances, and thereby compute Equation (8), we propose using *analytical covariance functions*, a well-known technique in statistical literature for approximating covariances [14]. In particular, Verdict uses squared exponential covariance functions, which is capable of approximating any continuous target function arbitrarily closely as the number of observations (here, query answers) increases [46].[4] Although the underlying distribution may not be a continuous function, it is sufficient for us to obtain $\nu_g(\boldsymbol{t})$ such that, when integrated (as in Equation (7)), produces the same values as the integrations of the underlying distribution.[5] In our setting, the squared exponential covariance function $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ is defined as:

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

Here, $l_{g,k}$ for k=1 . . . l and σ_g^2 are tunable *correlation parameters* to be learned from past queries and their answers (Appendix A).

Intuitively, when $\boldsymbol{t}$ and $\boldsymbol{t}'$ are similar, i.e., $(a_k - a_k')^2$ is small for most A_k, then $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ returns a larger value (closer to σ_g^2), indicating that the expected values of g for $\boldsymbol{t}$ and $\boldsymbol{t}'$ are highly correlated.

With the analytic covariance function above, the $\text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ terms involving inter-tuple covariances can in turn be computed analytically. Note that Equation (9) involves the multiplication of

[4] This property of the universal kernels is asymptotic (i.e., as the number of observations goes to infinity).

[5] The existence of such a continuous function is implied by the kernel density estimation technique [74].

l terms, each of which containing variables related to a single attribute. As a result, plugging Equation (9) into Equation (8) yields:

$$\mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) = \sigma_g^2 \prod_{k=1}^{l} \int_{s_{i,k}}^{e_{i,k}} \int_{s_{j,k}}^{e_{j,k}} \exp\left(-\frac{(a_k - a'_k)^2}{l_{g,k}^2}\right) da'_k a_k \tag{10}$$

The order of integrals are interchangeable, since the terms including no integration variables can be regarded as constants (and thus can be factored out of the integrals). Note that the double-integral of an exponential function can also be computed analytically (see [59]); thus, Verdict can efficiently compute $\mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ in $O(l)$ times by directly computing the integrals of inter-tuple covariances, without explicitly computing individual inter-tuple covariances. Finally, we can compose the $(n+2) \times (n+2)$ matrix Σ in Lemma 1 using Equation (6).

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

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

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 $(\widehat{\theta}_{n+1}, \widehat{\beta}_{n+1})$ is set either to Verdict's model-based answer/error, i.e., $(\ddot{\theta}_{n+1}, \ddot{\beta}_{n+1})$, or to the AQP system'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 $\widehat{\beta}_{n+1} \leq \beta_{n+1}$, and hence it is enough to show that $\ddot{\beta}_{n+1} \leq \beta_{n+1}$.

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

We first define notations. Let Σ be the covariance matrix of the vector of random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$; $\vec{k}_n$ be a column vector of length n whose i-th element is the $(i, n+1)$-th entry of Σ; Σ_n be an $n \times n$ submatrix of Σ that consists of Σ's first n rows/columns; $\bar{\kappa}^2$ be a scalar value at the $(n+2, n+2)$-th entry of Σ; and $\vec{\theta}_n$ be a column vector $(\theta_1, \ldots, \theta_n)^\intercal$.

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

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

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 Σ and $(i, n+2)$-th element of Σ have the same values for $i = 1, \ldots, n$. They have the same values because the covariance between $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{n+1}$ and the covariance between $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_{n+1}$ are same for $i = 1, \ldots, n$ due to Equation (6).

Using the formula of block matrix inversion [34], we can obtain the following alternative forms of Equations (4) and (5) (here, we assume zero means to simplify the expressions):

$$\gamma^2 = \bar{\kappa}^2 - \vec{k}_n^\intercal \Sigma_n^{-1} \vec{k}_n, \qquad \theta = \vec{k}_n^\intercal \Sigma_n^{-1} \vec{\theta}_n \tag{11}$$

$$\ddot{\theta}_{n+1} = \frac{\beta_{n+1}^2 \cdot \theta + \gamma^2 \cdot \theta_{n+1}}{\beta_{n+1}^2 + \gamma^2}, \qquad \ddot{\beta}_{n+1}^2 = \frac{\beta_{n+1}^2 \cdot \gamma^2}{\beta_{n+1}^2 + \gamma^2} \tag{12}$$

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 $\gamma^2 \to \infty$. □

Lemma 2. The time complexity of Verdict's inference is $O(N^{max} \cdot l \cdot n^2)$ The space complexity of Verdict is $O(n \cdot N^{max} + n^2)$, where $n \cdot N^{max}$ is the size of the query snippets and n^2 is the size of the precomputed covariance matrix.

Proof. It is enough to prove that the computations of a model-based answer and a model-based error can be performed in $O(n^2)$ time, where n is the number of past query snippets. Note that this is clear from Equations (11) and (12), because the computation of Σ_n^{-1} involves only the past query snippets. For computing γ^2, multiplying $\vec{k}_n$, a precomputed Σ_n^{-1}, and $\vec{k}_n$ takes $O(n^2)$ time. Similarly for θ in Equation (11) □

These results imply that the domain sizes of dimension attributes do not affect Verdict's computational overhead. This is because Verdict analytically computes the covariances between pairs of query answers without individually computing inter-tuple covariances (Section 4.2).

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** [28, 55, 79, 80]: 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, 17, 20, 30, 38, 62, 82]: 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 ϵ is the time needed by Verdict for inferring the improved answer and improved error. Thanks to the efficiency of Verdict's inference, ϵ 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 ϵ 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 Appendix C.2.

Some AQP engines also support error-bound queries but do not offer an online aggregation interface [7, 51, 64]. 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.

Note that Verdict's inference mechanism is not affected by the specific AQP engine used underneath, as long as the conditions in Section 3 hold, namely the error estimate β^2 is the expectation of the squared deviation of the approximate answer from the exact answer. However, the AQP engine's runtime overhead (e.g., query parsing and planning) may affect Verdict's overall benefit in relative terms. For example, if the query parsing amount to 90% of the overall query processing time, even if Verdict completely eliminates the need for processing any data, the relative speedup will only be 1.0/0.9 = 1.11×. However, Verdict is designed for data-intensive scenarios where disk or network I/O is a sizable portion of the overall query processing time.

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×) compared to a sample-based AQP solution.
- Given the same processing time, Verdict reduced 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 also have supplementary experiments in Appendix C. Appendix C.1 shows the benefits of model-based inference in comparison to a strawman approach, which simply caches all past query answers. Appendix C.2 demonstrates Verdict's benefit for time-bound AQP engines.

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 [10] (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. NoLearn estimates its errors and computes confidence intervals using closed-forms (based on the central limit theorem). Error estimation based on the central limit theorem has been one of the most popular approaches in online aggregation systems [28, 39, 79, 80] and other AQP engines [3, 6, 17].
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 computes an improved answer and error using our proposed approach. Naturally, Verdict incurs a (negligible) runtime overhead, due to supported query check, query decomposition, and computation of improved answers; however, Verdict 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 [68] 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.

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

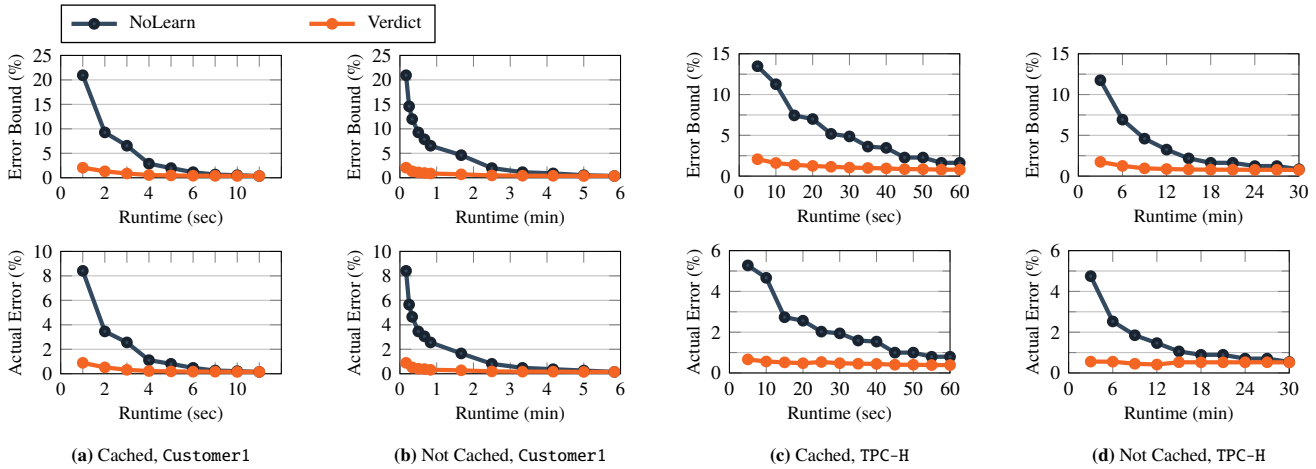
Table 3: Generality of Verdict. Verdict supports a large fraction of real-world and benchmark queries.

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 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



(a) Cached, `Customer1` **(b)** Not Cached, `Customer1` **(c)** Cached, `TPC-H` **(d)** Not Cached, `TPC-H`

Figure 4: 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.

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 kept the queries and their answers in its query synopsis. After processing those queries, Verdict (i) pre-computed the matrix inversions and (ii) learned the correlation parameters. The matrix inversions took 1.6 seconds in total; the correlation parameter learning took 23.7 seconds for `TPC-H` and 8.04 seconds for `Customer1`. The learning process was relatively faster for `Customer1` since most of the queries included `COUNT(*)` for which each attribute did not require a separate learning. This offline training time for both workloads was comparable to the time needed for running only a single approximate query (Table 4).

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 4 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.

	Cached?	Error Bound	Time Taken NoLearn	Time Taken Verdict	Speedup
`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 NoLearn	Achieved Error Bound Verdict	Error Reduction
`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.

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 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

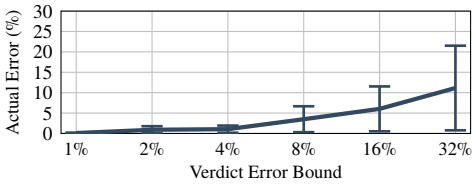


Figure 5: The comparison between Verdict's error bound at 95% confidence and the actual error distribution (5th, 50th, and 95th percentiles are reported for actual error distributions).

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 C.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 5 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.

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 memory footprint of the query synopsis was 5.79MB for `TPC-H` and 18.5MB for `Customer1` workload (23.2KB per-query for `TPC-H` and 15.8KB per-query for `Customer1`). This included past queries in parsed forms, model parameters, covariance matrices, and the inverses of those covariance matrices. The size of query synopsis was small because Verdict 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.)

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.

7.6 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.

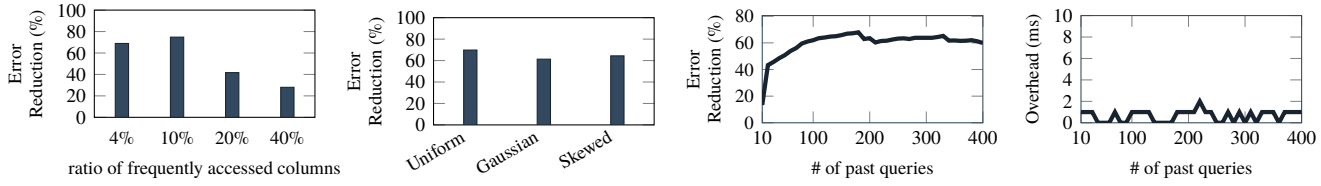
First, we studied the impact of having queries with a more diverse set of columns in their selection predicates. We produced a table of 50 columns and 5M rows, where 10% of the columns were categorical. The domains of the numeric columns were the real values between 0 and 10, and the domains of the categorical columns were the integers between 0 and 100.

Also, we 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 6(a) shows that as the proportion 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 6(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 6(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 6(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



(a) Workload Diversity **(b)** Data distribution **(c)** Learning Behavior **(d)** Overhead

Figure 6: 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.

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 6(d)).

Recall that the domain size of the attributes does not affect Verdict's computational cost since only the lower and upper bounds of range constraints are needed for covariance computations (Section 4.2).

8. RELATED WORK

Approximate Query Processing— There has been substantial work on sampling-based approximate query processing [3, 4, 6, 11, 17, 19, 22, 24, 29, 39, 45, 52, 53, 63, 69]. Some of these systems differ in their sample generation strategies (see [50] and the references within). For instance, STRAT [17] and AQUA [4] create a single stratified sample, while BlinkDB creates samples based on *column sets*. Online Aggregation (OLA) [18, 28, 55, 77] continuously refines its answers during query execution. Others have focused on obtaining faster or more reliable error estimates [5, 78]. 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 [9, 60], probabilistic data [25, 54], and RDF data [31, 72], and searching in high-dimensional space [57]. However, our focus in this paper is on general (SQL-based) AQP engines.

Adaptive Indexing, View Selection— Adaptive Indexing and database cracking [32,33] has been proposed for a column-store database as a means of incrementally updating indices as part of query processing; then, it can speed up future queries that access previously indexed ranges. While the adaptive indexing is an effective mechanism for exact analytic query processing in column-store databases, answering queries that require accessing multiple columns (e.g., selection predicates on multiple columns) is still a challenging task: column-store databases have to join relevant columns to reconstruct tuples. Although Idreos et al. [33] pre-join some subsets of columns, the number of column combinations still grows exponentially as the total number of columns in a table increases. Verdict can easily handle queries with multiple columns due to its analytic inference. Materialized views are another means of speeding up future aggregate queries [21, 27, 37, 49]. Verdict also speed up aggregate queries, but Verdict does not require strict query containments as in materialized views.

Maximum Entropy Principle— In the database community, the principle of maximum entropy (ME) has been previously used for determining the most surprising piece of information in a data exploration context [67], and for constructing histograms based on cardinality assertions [40]. Verdict uses ME differently than these previous approaches; they assign a unique variable to each non-overlapping area to represent the number of tuples belonging to that area. This approach poses two challenges when applied to an AQP context. First, it requires a slow iterative numeric solver for its inference. Thus, using this approach for DBL may eliminate any potential speedup. Second, introducing a unique variable for every non-overlapping area can be impractical as it requires $O(2^n)$ variables for n past queries. Finally, the previous approach cannot express inter-tuple covariances in the underlying data. In contrast, Verdict's approach handles arbitrarily overlapping ranges in multi-dimensional space with $O(n)$ variables (and $O(n^2)$ space), and its inference can be performed analytically.

9. CONCLUSION AND FUTURE WORK

In this paper, we presented database learning (DBL), 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 DBL, (ii) the development of *active database learning* [48, 56], 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 [12, 36, 41, 58, 73, 76].

10. ACKNOWLEDGEMENT

This work was in part funded by NSF awards 1544844, 1629397, and 1553169. The authors are grateful to Aditya Modi, Morgan Lovay, and Shannon Riggins for their feedback on this manuscript.

11. 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. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 2012.
[8] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.

[9] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.

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

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

[12] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. 2015.

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

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

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

[16] 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.

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

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

[19] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

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

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

[22] W. Fan, F. Geerts, Y. Cao, T. Deng, and P. Lu. Querying big data by accessing small data. In *PODS*, 2015.

[23] D. Freedman, R. Pisani, and R. Purves. *Statistics*. 2007.

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

[25] W. Gatterbauer and D. Suciu. Approximate lifted inference with probabilistic databases. *PVLDB*, 2015.

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

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

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

[29] K. Hose, D. Klan, and K.-U. Sattler. Distributed data summaries for approximate query processing in pdms. In *IDEAS*, 2006.

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

[31] H. Huang, C. Liu, and X. Zhou. Approximating query answering on rdf databases. *WWW*, 2012.

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

[33] S. Idreos, M. L. Kersten, and S. Manegold. Self-organizing tuple reconstruction in column-stores. In *SIGMOD*, 2009.

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

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

[36] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Interactive data exploration with smart drill-down. In *ICDE*, 2016.

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

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

[39] 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.

[40] R. Kaushik, C. Ré, and D. Suciu. General database statistics using entropy maximization. In *DBPL*, 2009.

[41] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 2015.

[42] N. Lawrence, M. Seeger, R. Herbrich, et al. Fast sparse gaussian process methods: The informative vector machine. *NIPS*, 2003.

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

[44] M. Lovric. *International Encyclopedia of Statistical Science*. Springer, 2011.

[45] A. Meliou, C. Guestrin, and J. M. Hellerstein. Approximating sensor network queries using in-network summaries. In *IPSN*, 2009.

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

[47] B. Mozafari. Verdict: A system for stochastic query planning. In *CIDR, Biennial Conference on Innovative Data Systems*, 2015.

[48] B. Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*, 2017.

[49] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. CliffGuard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

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

[51] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. Snappydata: A unified cluster for streaming, transactions, and interactive analytics. In *CIDR*, 2017.

[52] B. Mozafari and C. Zaniolo. Optimal load shedding with aggregates and mining queries. In *ICDE*, 2010.

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

[54] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.

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

[56] Y. Park. Active database learning. In *CIDR*, 2017.

[57] Y. Park, M. Cafarella, and B. Mozafari. Neighbor-sensitive hashing. *PVLDB*, 2015.

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

[59] Y. Park, A. S. Tajik, M. Cafarella, and B. Mozafari. Database learning: Toward a database that becomes smarter every time. https://arxiv.org/abs/1703.05468.

[60] K. S. Perera, M. Hahmann, W. Lehner, T. B. Pedersen, and C. Thomsen. Efficient approximate olap querying over time series. In *IDEAS*, 2016.

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

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

[63] N. Potti and J. M. Patel. Daq: a new paradigm for approximate query processing. *PVLDB*, 2015.

[64] J. Ramnarayan, B. Mozafari, S. Menon, S. Wale, N. Kumar, H. Bhanawat, S. Chakraborty, Y. Mahajan, R. Mishra, and K. Bachhav. Snappydata: A hybrid transactional analytical store built on spark. In *SIGMOD*, 2016.

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

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

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

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

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

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

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

[72] A. Souihli and P. Senellart. Optimizing approximations of dnf query lineage in probabilistic xml. In *ICDE*, 2013.

[73] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 2015.

[74] L. Wasserman. *All of Nonparametric Statistics*. Springer, 2006.

[75] C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, 2000.

[76] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. *PVLDB*, 2014.

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

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

[79] 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.

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

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

[82] 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

Appendix is organized as follows. Appendices A and B provide Verdict's learning and model validation processes, respectively. Appendix C includes additional experiments for demonstrating the effectiveness and accuracy of Verdict. Appendix D studies the prevalence of non-zero inter-tuple correlations in real-world datasets.

A. PARAMETER LEARNING

This section describes Verdict's correlation parameter learning. In Appendix A.1, we present a mathematical description of the process, and in Appendix A.2, we study its effectiveness with experiments.

A.1 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:

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

where Σ_n is an $n \times n$ submatrix of Σ consisting of Σ's first n rows and columns, i.e., (co)variances between pairs of past query answers, and $\propto$ indicates that the two values are proportional, Therefore, without any preference over parameter values, determining the most likely correlation parameters (which determine Σ_n) given past queries amounts to finding the values for $l_{g,1}, \ldots, l_{g,l}, \sigma_g^2$ that maximize the below log-likelihood function:

$$\begin{aligned} &\log \Pr(\vec{\theta}_{\text{past}} \mid \Sigma_n) = \log\ f(\vec{\theta}_{\text{past}}) \\ &= -\frac{1}{2}\vec{\theta}_{\text{past}}^{\intercal}\Sigma_n^{-1}\vec{\theta}_{\text{past}} - \frac{1}{2}\log|\Sigma_n| - \frac{n}{2}\log 2\pi \end{aligned} \tag{13}$$

where $f(\vec{\theta}_{\text{past}})$ is the joint pdf from Equation (2).

Verdict finds the optimal values for $l_{g,1}, \ldots, l_{g,l}$ by solving the above optimization problem with a numerical solver, while it estimates the value for σ_g^2 analytically from past query answers (see [59]). Concretely, the current implementation of Verdict uses the gradient-descent-based (quasi-newton) nonlinear programming solver provided by Matlab's `fminuncon()` function, without providing explicit gradients. Although our current approach is typically slower than using closed-form solutions or than using the solver with an explicit gradient (and a Hessian), 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. We plan to improve the efficiency of this offline training by using explicit gradient expressions.

Since Equation (13) is not a convex function, the solver of our choice only returns a locally optimal point. A conventional strategy to handle this issue is to obtain multiple locally optimal points by solving the same problem with multiple random starting points, and to take the one with the highest log-likelihood value as a final answer. Still, this approach does not guarantee the correctness of the model. In contrast, Verdict's strategy is to find a locally-optimal point that can capture potentially large inter-tuple covariances, and to validate the correctness of the resulting model against a model-free answer (Appendix B). We demonstrate empirically in the following section that this strategy is effective for finding parameter values that are close to true values. Verdict's model validation process in Appendix B provides robustness against the models that may differ from the true distribution. Verdict uses $l_{g,k} = (\max(A_k) - \min(A_k))$ for the starting point of the optimization problem.

Lastly, our use of approximate answers as the constraints for the ME principle is properly accounted for by including additive error terms in their (co)variances (Equation (6)).

A.2 Accuracy of Parameter Learning

In this section, we demonstrate our empirical study on the effectiveness of Verdict's correlation parameter estimation process. For this, we used the synthetic datasets generated from pre-determined 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.

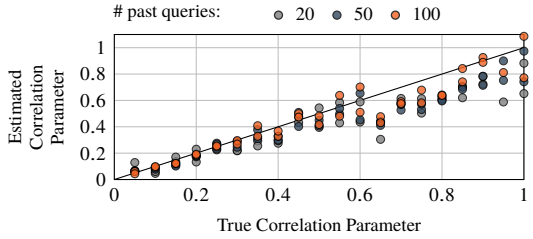


Figure 7: Correlation Param Learning

Figure 7 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.

B. MODEL VALIDATION

Verdict aims to provide correct error bounds even when its model differs significantly from the true data. In Appendix B.1, we describes its process, and in Appendix B.2, we empirically demonstrate its effectiveness.

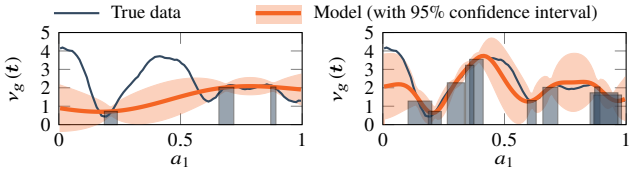
B.1 Model Validation Procedure

Verdict'd model validation rejects 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. Verdict's model validation process addresses two situations: (i) negative estimates for `FREQ(*)`, and (ii) an unlikely large discrepancy between a model-based answer and a raw answer.

Negative estimates for `FREQ(*)`— To obtain the prior distribution of the random variables, Verdict uses the most-likely distribution (based on the maximum entropy principle (Lemma 1)), given the means, variances, and covariances of query answers. Although this makes the inference analytically computable, the lack of explicit non-negative constraints on the query answers may produce negative estimates on `FREQ(*)`. Verdict handles this situation with a simple check; that is, Verdict rejects its model-based answer if $\ddot{\theta}_{n+1} < 0$ for `FREQ(*)`, and uses the raw answer instead. Even if $\ddot{\theta}_{n+1} \geq 0$, the lower bound of the confidence interval is set to zero if the value is less than zero.

Unlikely model-based answer— Verdict's model learned from empirical observations may be different from the true distribution. Figure 8(a) illustrates such an example. 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 8(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 8(a) by validating its model-based answers against the *model-free* answers obtained from the AQP engine. Specifically, we



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

Figure 8: An example of (a) overly optimistic confidence intervals due to incorrect estimation of the underlying distributon, and (b) its resolution with more queries processed. Verdict relies on a model validation to avoid the situation as in (a).

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 θ_{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. This process is akin to hypothesis testing in statistics literatures [23].

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 B.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 $(\ddot{\theta}_{n+1} - t,\ \ddot{\theta}_{n+1} + t)$ with probability δ_v (0.99 by default) if $\ddot{\theta}_{n+1}$ were the exact answer. We call the $(\ddot{\theta}_{n+1} - t,\ \ddot{\theta}_{n+1} + t)$ range the likely region. To compute t, we must find the value closest to $\ddot{\theta}_{n+1}$ that satisfies the following expression:

$$\Pr\left(|X - \ddot{\theta}_{n+1}| < t\right) \geq \delta_v \tag{14}$$

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 $\ddot{\theta}_{n+1}$. The AQP engine's answer can be treated as a random variable since it may differ depending on the random samples used. The probability in Equation (14) can be easily computed using either the central limit theorem or the Chebyshev's inequality [44]. Once the value of t is computed, Verdict rejects its model if θ_{n+1} falls outside the likely region $(\ddot{\theta}_{n+1} - t,\ \ddot{\theta}_{n+1} + t)$.

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

B.2 Empirical Study on Model Validation

This section presents our study of the effect of Verdict's model validation described in Appendix B.1. 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 9 reports the experiment results from when Verdict was tested without a model validation step and with a model validation step, respectively. In the figure, the values on the X-axis are artificial correlation parameter scales, i.e., the product of the true

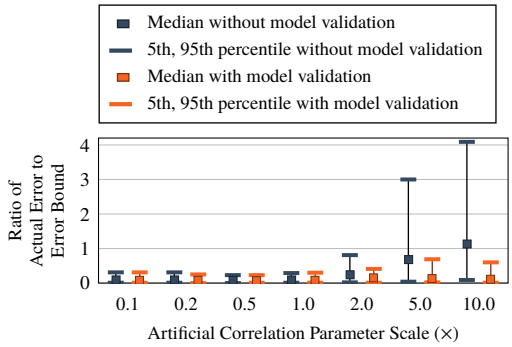


Figure 9: Effect of model validation. For Verdict's error bounds to be correct, the 95th percentile should be below 1.0. One can find that, with Verdict's model validation, the improved answers and the improved errors were probabilistically correct even when largely incorrect correlation parameters were used.

correlation parameters and each of those scales are set in Verdict's model. For instance, if a true correlation parameter was 5.0, and the "artificial correlation parameter scale" was 0.2, Verdict's model was set to 1.0 for the correlation parameter. Thus, the values of the correlation parameters in Verdict's model were set to the true correlation parameters, when the "artificial correlation parameter scale" was 1.0. Since the Y-axis reports the ratio of the actual error to Verdict's error bound, Verdict's error bound was correct when the value on the Y-axis was 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 system.

C. ADDITIONAL EXPERIMENTS

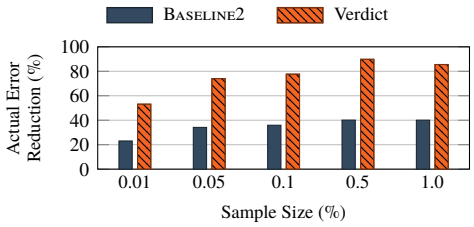
This section contains additional experiments we have not included in the main part of our paper. First, we study the impact of two factors that affect the performance benefits of Verdict (Appendix C.1). Second, we show that Verdict can achieve error reductions over time-bound AQP engines (Appendix C.2).

C.1 Verdict vs. Simple Answer Caching

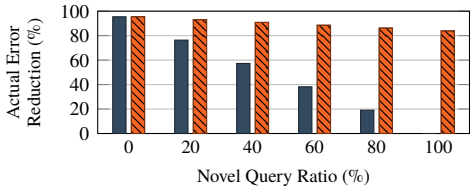
To study the benefits of Verdict's model-based inference, we consider another system Baseline2, and make comparisons between Verdict and Baseline2, using the TPC-H dataset. Baseline2 is similar to NoLearn but returns a cached answer if the new query is identical to one of the past ones. When there are multiple instances of the same query, Baseline2 caches the one with the lowest expected error.

Figure 10(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 10(b) compares Verdict and Baseline2 by changing the ratio of novel queries in the workload. Understandably, both Verdict



(a) Sample Sizes for Past Queries



(b) Query Workload Composition

Figure 10: (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.

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.

C.2 Error Reductions for Time-Bound AQP Engines

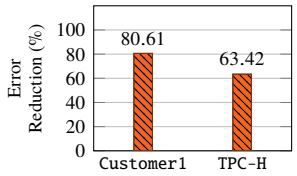
Recall that, in Section 7, we demonstrated Verdict's speedup and error reductions over an online aggregation system. In this section, we show Verdict's error reductions over a time-bound AQP system. First, we describe our experiment setting. Next, we present our experiment results.

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

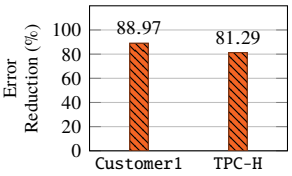
1. NoLearn: This system runs queries on *samples* of the original tables to obtain fast but approximate query answers and their associated estimated errors. This is the same approach taken by existing AQP engines, such as [2, 6, 17, 62, 65, 81]. Specifically, NoLearn maintains uniform random samples created offline (10% of the original tables), and it 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 using 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 we set the time-bounds as 5.0 seconds for both `Customer1` and `TPC-H` datasets loaded from SSD. Figure 11 reports Verdict's error reductions over NoLearn for each of four different combinations of a dataset and cache setting.



(a) Cached



(b) Not Cached

Figure 11: Average error reduction by Verdict (compared to NoLearn) for the same time budget.

In Figure 11, one can observe that Verdict achieved large error reductions (63%–86%) over NoLearn. These results indicate that the users of Verdict can obtain much more precise answers compared to the users of NoLearn within the same time-bounds.

D. PREVALENCE OF INTER-TUPLE COVARIANCES IN REAL-WORLD

In this section, we demonstrate the existence of the inter-tuple covariances in many real-world datasets by analyzing well-known datasets from the UCI repository [43]. 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.

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.

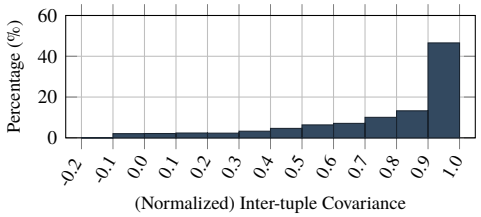


Figure 12: *Inter-tuple Covariances* for 16 real-life UCI datasets.

Figure 12 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 Appendix A, relying on the past snippet answers stored in the query synopsis.