QuickSel: Quick Selectivity Learning with Mixture Models

Yongjoo Park\textsuperscript{1,}\textsuperscript{*}, Shucheng Zhong\textsuperscript{2,}\textsuperscript{*}, Barzan Mozafari\textsuperscript{2}
University of Illinois at Urbana-Champaign\textsuperscript{1} University of Michigan\textsuperscript{2}
yongjoo@illinois.edu \{joezhong,mozafari\}@umich.edu

ABSTRACT
Estimating the selectivity of a query is a key step in almost any cost-based query optimizer. Most of today’s databases rely on histograms or samples that are periodically refreshed by re-scanning the data as the underlying data changes. Since frequent scans are costly, these statistics are often stale and lead to poor selectivity estimates. As an alternative to scans, query-driven histograms have been proposed, which refine the histograms based on the actual selectivities of the observed queries. Unfortunately, these approaches are either too costly to use in practice—i.e., require an exponential number of buckets—or quickly lose their advantage as they observe more queries.

In this paper, we propose a selectivity learning framework, called QuickSel, which falls into the query-driven paradigm but does not use histograms. Instead, it builds an internal model of the underlying data, which can be refined significantly faster (e.g., only 1.9 milliseconds for 300 queries). This fast refinement allows QuickSel to continuously learn from each query and yield increasingly more accurate selectivity estimates over time. Unlike query-driven histograms, QuickSel relies on a mixture model and a new optimization algorithm for training its model. Our extensive experiments on two real-world datasets confirm that, given the same target accuracy, QuickSel is \(34.0 \times - 179.4 \times\) faster than state-of-the-art query-driven histograms, including ISOMER and STHoles. Further, given the same space budget, QuickSel is 26.8\%–91.8\% more accurate than periodically-updated histograms and samples, respectively.

\textsuperscript{*}These authors contributed equally to this work.

1 INTRODUCTION
Estimating the selectivity of a query—the fraction of input tuples that satisfy the query’s predicate—is a fundamental component in cost-based query optimization, including both traditional RDBMSs \cite{2, 3, 7, 9, 83} and modern SQL-on-Hadoop engines \cite{41, 88}. The estimated selectivities allow the query optimizer to choose the cheapest access path or query plan \cite{53, 90}.

Today’s databases typically rely on histograms \cite{2, 7, 9} or samples \cite{83} for their selectivity estimation. These structures need to be populated in advance by performing costly table scans. However, as the underlying data changes, they quickly become stale and highly inaccurate. This is why they need to be updated periodically, creating additional costly operations in the database engine (e.g., \texttt{ANALYZE table}).\textsuperscript{1}

To address the shortcoming of scan-based approaches, numerous proposals for query-driven histograms have been introduced, which continuously correct and refine the histograms based on the actual selectivities observed after running each query \cite{11, 12, 19, 52, 66, 75, 86, 93, 96}. There are two approaches to query-driven histograms. The first approach \cite{11, 12, 19, 66}, which we call error-feedback histograms, recursively splits existing buckets (both boundaries and frequencies) for every distinct query observed, such that their error is minimized for the latest query. Since the error-feedback histograms do not minimize the (average) error across multiple queries, their estimates tend to be much less accurate.

To achieve a higher accuracy, the second approach is to compute the bucket frequencies based on the maximum entropy principle \cite{52, 75, 86, 93}. However, this approach

\textsuperscript{1}Some database systems \cite{9} automatically update their statistics when the number of modified tuples exceeds a threshold.
(which is also the state-of-the-art) requires solving an optimization problem, which quickly becomes prohibitive as the number of observed queries (and hence, number of buckets) grows. Unfortunately, one cannot simply prune the buckets in this approach, as it will break the underlying assumptions of their optimization algorithm (called iterative scaling, see §2.3 for details). Therefore, they prune the observed queries instead in order to keep the optimization overhead feasible in practice. However, this also means discarding data that could be used for learning a more accurate distribution.

**Our Goal** We aim to develop a new framework for selectivity estimation that can quickly refine its model after observing each query, thereby producing increasingly more accurate estimates over time. We call this new framework **selectivity learning**. We particularly focus on designing a low-overhead method that can scale to a large number of observed queries without requiring an exponential number of buckets.

**Our Model** To overcome the limitations of query-driven histograms, we use a **mixture model** [17] to capture the unknown distribution of the data. A mixture model is a probabilistic model to approximate an arbitrary probability density function (pdf), say $f(x)$, using a combination of simpler pdfs:

$$f(x) = \sum_{z=1}^{m} h(z) g_z(x) \quad (1)$$

where $g_z(x)$ is the $z$-th simpler pdf and $h(z)$ is its corresponding weight. The subset of the data that follows each of the simpler pdfs is called a **subpopulation**. Since the subpopulations are allowed to overlap with one another, a mixture model is strictly more expressive than histograms. In fact, it is shown that mixture models can achieve a higher accuracy than histograms [24], which is confirmed by our empirical study (§5.5). To the best of our knowledge, we are the first to propose a mixture model for selectivity estimation.²

**Challenges** Using a mixture model for selectivity learning requires finding optimal parameter values for $h(z)$ and $g_z(x)$; however, this optimization (a.k.a. training) is challenging for two reasons.

First, the training process aims to find parameters that maximize the model **quality**, defined as $\int Q(f(x)) \, dx$ for some metric of quality $Q$ (e.g., entropy). However, computing this integral is non-trivial for a mixture model since its subpopulations may overlap in arbitrary ways. That is, the combinations of $m$ subpopulations can create $2^m$ distinct ranges, each with a potentially different value of $f(x)$. As a result, naively computing the quality of a mixture model quickly becomes intractable as the number of observed queries grows.

Second, the outer optimization algorithms are often iterative (e.g., iterative scaling, gradient descent), which means they have to repeatedly evaluate the model quality as they search for optimal parameter values. Thus, even when the model quality can be evaluated relatively efficiently, the overall training/optimization process can be quite costly.

**Our Approach** First, to ensure the efficiency of the model quality evaluation, we propose a new optimization objective. Specifically, we find the parameter values that minimize the $L^2$ distance (or equivalently, mean squared error) between the mixture model and a uniform distribution, rather than maximizing the entropy of the model (as pursued by previous work [52, 74, 75, 86, 93]). As described above, directly computing the quality of a mixture model involves costly integrations over $2^m$ distinct ranges. However, when minimizing the $L^2$ distance, the $2^m$ integrals can be reduced to only $m^2$ multiplications, hence greatly reducing the complexity of the model quality evaluation. Although minimizing the $L^2$ distance is much more efficient than maximizing the entropy, these two objectives are closely related (see our report [82] for a discussion).

In addition, we adopt a non-conventional variant of mixture models, called a **uniform mixture model**. While uniform mixture models have been previously explored in limited settings (with only a few subpopulations) [26, 36], we find that they are quite appropriate in our context as they allow for efficient computations of the $L^2$ distance. That is, with this choice, we can evaluate the quality of a model by only using min, max, and multiplication operations (§3.2). Finally, our optimization can be expressed as a standard **quadratic program**, which still requires an iterative procedure.

---

²Our mixture model also differs from **kernel density estimation** [18, 35, 40], which scans the actual data, rather than analyzing observed queries.

---

**Table 1: The differences between query-driven histograms [52, 74, 75, 86, 93] and our method (QuickSel)**

<table>
<thead>
<tr>
<th></th>
<th>Query-driven Histograms</th>
<th>QuickSel (ours)</th>
<th>Our Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>histograms (non-overlapping buckets)</td>
<td>mixture models (overlapping subpopulations)</td>
<td>Employs a new expressive model</td>
</tr>
<tr>
<td>Training</td>
<td>maximum entropy solved by iterative scaling</td>
<td>min difference from a uniform distribution solved analytically</td>
<td>A new optimization objective and its reduction to quadratic programming (solved analytically)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ no exponential growth of complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ fast training and model refinements</td>
</tr>
</tbody>
</table>

---

In addition, we adopt a non-conventional variant of mixture models, called a **uniform mixture model**. While uniform mixture models have been previously explored in limited settings (with only a few subpopulations) [26, 36], we find that they are quite appropriate in our context as they allow for efficient computations of the $L^2$ distance. That is, with this choice, we can evaluate the quality of a model by only using min, max, and multiplication operations (§3.2). Finally, our optimization can be expressed as a standard **quadratic program**, which still requires an iterative procedure.
Therefore, to avoid the costly iterative optimization, we also devise an analytic solution that can be computed more efficiently. Specifically, in addition to the standard reduction (i.e., moving some of the original constraints to the objective clause as penalty terms), we completely relax the positivity constraints for $f(x)$, exploiting the fact that they will be naturally satisfied in the process of approximating the true distribution of the data. With these modifications, we can solve for the solution analytically by setting the gradient of the objective function to zero. This simple transformation speeds up the training by $1.5x-17.2x$. In addition, since our analytic solution requires a constant number of operations, the training time is also consistent across different datasets and workloads.

Using these ideas, we have developed a first prototype of our selectivity learning proposal, called QuickSel, which allows for extremely fast model refinements. As summarized in Table 1, QuickSel differs from—and considerably improves upon—query-driven histograms [11, 52, 66, 75, 86, 93] in terms of both modeling and training (see §7 for a detailed comparison).

**Contributions** We make the following contributions:

1. We propose the first mixture model-based approach to selectivity estimation (§3).
2. For training the mixture model, we design a constrained optimization problem (§4.1).
3. We show that the proposed optimization problem can be reduced (from an exponentially complex one) to a quadratic program and present further optimization strategies for solving it (§4.2).
4. We conduct extensive experiments on two real-world datasets to compare QuickSel’s performance and state-of-the-art selectivity estimation techniques (§5).

## 2 PRELIMINARIES

In this section, we first define relevant notations in §2.1 and then formally define the problem of query-driven selectivity estimation in §2.2. Next, in §2.3, we discuss the drawbacks of previous approaches.

### 2.1 Notations

Table 2 summarizes the notations used throughout this paper.

**Set Notations** $T$ is a relation that consists of $d$ real-valued columns $C_1, \ldots, C_d$.\footnote{Handling integer and categorical columns is discussed in §2.2.} The range of values in $C_i$ is $[l_i, u_i]$ and the cardinality (i.e., row count) of $T$ is $N=|T|$. The tuples in $T$ are denoted by $x_1, \ldots, x_N$, where each $x_i$ is a size-$d$ vector that belongs to $B_0 = [l_1, u_1] \times \cdots \times [l_d, u_d]$. Geometrically, $B_0$ is the area bounded by a hyperrectangle whose bottom-left corner is $(l_1, \ldots, l_d)$ and top-right corner is $(u_1, \ldots, u_d)$. The size of $B_0$ can thus be computed as $|B_0|=(u_1-l_1)\times \cdots \times (u_d-l_d)$.

**Predicates** We use $P_i$ to denote the (selection) predicate of the $i$-th query on $T$. In this paper, a predicate is a conjunction\footnote{See §2.2 for a discussion of disjunctions and negations.} of one or more constraints. Each constraint is a range constraint, which can be one-sided (e.g., $3 \leq C_1$) or two-sided (e.g., $-3 \leq C_1 \leq 10$). This range can be extended to also handle equality constraints on categorical data (see §2.2). Each predicate $P_i$ is represented by a hyperrectangle $B_i$. For example, a constraint $"1 \leq C_1 \leq 3 \text{ AND } 2 \leq C_2"$ is represented by a hyperrectangle $(1,3) \times (2, u2)$, where $u2$ is the upper-bound of $C_2$. We use $P_0$ to denote an empty predicate, i.e., one that selects all tuples.

**Selectivity** The selectivity $s_i$ of $P_i$ is defined as the fraction of the rows of $T$ that satisfy the predicate. That is, $s_i = \frac{1}{N} \sum_{k=1}^N I(x_k \in B_i)$, where $I(\cdot)$ is the indicator function. A pair $(P_i, s_i)$ is referred to as an observed query.\footnote{This pair is also referred to as an assertion by prior work [85].} Without loss of generality, we assume that $n$ queries have been observed for $T$ and seek to estimate $s_{ni}$. Finally, we use $f(x)$ to denote the joint probability density function of tuple $x$ (that has generated tuples of $T$).

### 2.2 Problem Statement

Next, we formally state the problem:

**Problem 1 (Query-driven Selectivity Estimation)** Consider a set of $n$ observed queries $(P_1, s_1),\ldots, (P_n, s_n)$ for $T$. By definition, we have the following for each $i = 1, \ldots, n$:

$$\int_{x \in B_i} f(x) \, dx = s_i$$
Then, our goal is to build a model of \( f(x) \) that can estimate the selectivity \( s_{n+1} \) of a new predicate \( P_{n+1} \).

Initially, before any query is observed, we can conceptually consider a default query \( (P_0, 1) \), where all tuples are selected and hence, the selectivity is 1 (i.e., no predicates).

**Discrete and Categorical Values**  Problem 1 can be extended to support discrete attributes (e.g., integers, characters, categorical values) and equality constraints on them, as follows. Without loss of generality, suppose that \( C_i \) contains the integers in \( \{1, 2, \ldots, b_i\} \). To apply the solution to Problem 1, it suffices to (conceptually) treat these integers as real values in \( [1, b_i + 1] \) and then convert the original constraints on the integer values into range constraints, as follows. A constraint of the form \( "C_i = k" \) will be converted to a range constraint of the form \( k \leq C_i < k + 1 \). Mathematically, this is equivalent to replacing a probability mass function with a probability density function defined using dirac delta functions. Then, the summation of the original probability mass function can be converted to the integration of the probability density function. String data types (e.g., char, varchar) and their equality constraints can be similarly supported, by conceptually mapping each string into an integer (preserving their order) and applying the conversion described above for the integer data type.

**Supported Queries**  Similar to prior work [11, 19, 52, 66, 74, 75, 86, 93], we support selectivity estimation for predicates with conjunctions, negations, and disjunctions of range and equality constraints on numeric and categorical columns. We currently do not support wildcard constraints (e.g., LIKE ‘*word*’), EXISTS constraints, or ANY constraints. In practice, often a fixed selectivity is used for unsupported predicates, e.g., 3.125% in Oracle [83].

To simplify our presentation, we focus on conjunctive predicates. However, negations and disjunctions can also be easily supported. This is because our algorithm only requires the ability to compute the intersection size of pairs of predicates \( P_i \) and \( P_j \), which can be done by converting \( P_i \land P_j \) into a disjunctive normal form and then using the inclusion-exclusion principle to compute its size.

As in the previous work, we focus our presentation on predicates on a single relation. However, any selectivity estimation technique for a single relation can be applied to estimating selectivity of a join query whenever the predicates on the individual relations are independent of the join conditions [7, 41, 90, 98].

---

\(^a\) A dirac delta function \( \delta(x) \) outputs \( \infty \) if \( x = 0 \) and outputs 0 otherwise while satisfying \( \int \delta(x) \, dx = 1 \).
exponentially as the number of observed queries grows. For example, in our experiment in §5.5, the number of buckets was 22,370 for 100 observed queries, and 318,936 for 300 observed queries. Unfortunately, the number of buckets directly affects the training time. Specifically, using iterative scaling—the optimization algorithm used by all previous work [52, 74, 75, 86, 93]—the cost of each iteration grows linearly with the number of variables (i.e., the number of buckets). This means that the cost of each iteration can grow exponentially with the number of observed queries.

As stated in §1, we address this problem by employing a mixture model, which can express a probability distribution more effectively than query-driven histograms. Specifically, our empirical study in §5.5 shows that—using the same number of parameters—a mixture model achieves considerably more accurate estimates than histograms.

**Limitation 2: Non-trivial Bucket Merge/Pruning** Given that query-driven histograms [75, 93] quickly become infeasible due to their large number of buckets, one might consider merging or pruning the buckets in an effort to reduce their training times. However, merging or pruning the histogram buckets violates the assumption used by their optimization algorithms, i.e., iterative scaling. Specifically, iterative scaling relies on the fact that a bucket is either completely included in a query’s predicate range or completely outside of it. That is, no partial overlap is allowed. However, merging some of the buckets will inevitably cause partial overlaps (between predicate and histogram buckets). For interested readers, we have included a more detailed explanation of why iterative scaling requires this assumption in our technical report [82].

### 3 QUICKSEL: MODEL

This section presents how QuickSel models the population distribution and estimates the selectivity of a new query. QuickSel’s model relies on a probabilistic model called a mixture model. In §3.1, we describe the mixture model employed by QuickSel. §3.2 describes how to estimate the selectivity of a query using the mixture model. §3.3 describes the details of QuickSel’s mixture model construction.

#### 3.1 Uniform Mixture Model

A mixture model is a probabilistic model that expresses a (complex) probability density function of the population as a combination of (simpler) probability density functions (of subpopulations). The population distribution is the one that generates the tuple \( x \) of \( T \). The subpopulations are internally managed by QuickSel to best approximate \( f(x) \).

**Uniform Mixture Model** QuickSel uses a type of mixture model, called the uniform mixture model. The uniform mixture model represents a population distribution \( f(x) \) as a weighted summation of multiple uniform distributions, \( g_z(x) \) for \( z = 1, \ldots, m \). Specifically,

\[
    f(x) = \sum_{z=1}^{m} h(z) g_z(x) = \sum_{z=1}^{m} w_z g_z(x)
\]

where \( h(z) \) is a categorical distribution that determines the weight of the \( z \)-th subpopulation, and \( g_z(x) \) is the probability density function (which is a uniform distribution) for the \( z \)-th subpopulation. The support of \( h(z) \) is the integers ranging from 1 to \( m \); \( h(z) = w_z \). The support for \( g_z(x) \) is represented by a hyperrectangle \( G_z \). Since \( g_z(x) \) is a uniform distribution, \( g_z(x) = 1/|G_z| \) if \( x \in G_z \) and 0 otherwise. The locations of \( G_z \) and the values of \( w_z \) are determined in the training stage (§4). In the remainder of this section (§3), we assume that \( G_z \) and \( w_z \) are given.

**Benefit of Uniform Mixture Model** The uniform mixture model was studied early in the statistics community [26, 36]; however, recently, a more complex model called the Gaussian mixture model has received more attention [17, 84, 110]. The Gaussian mixture model uses a Gaussian distribution for each subpopulation; the smoothness of its probability density function (thus, differentiable) makes the model more appealing when gradients need to be computed. Nevertheless, we intentionally use the uniform mixture model for QuickSel due to its computational benefit in the training process, as we describe below.

As will be presented in §4.2, QuickSel’s training involves the computations of the intersection size between two subpopulations, for which the essential operation is evaluating the following integral:

\[
    \int g_{z_1}(x) g_{z_2}(x) \, dx
\]

Evaluating the above expression for multivariate Gaussian distributions, e.g., \( g_{z_1}(x) = \exp(-x^\top \Sigma_1^{-1} x) / \sqrt{(2\pi)^d |\Sigma_1|} \), requires numerical approximations [31, 50], which are either slow or inaccurate. In contrast, the intersection size between two hyperrectangles can be exactly computed by simple min, max, and multiplication operations.

#### 3.2 Selectivity Estimation with UMM

For the uniform mixture model, computing the selectivity of a predicate \( P_i \) is straightforward:

\[\text{There are other variants of mixture models [15, 78].}\]
Predicate ranges

Generates points using predicate ranges

Workload-aware points

Creates ranges that cover the points

(a) Case 1: Highly-overlapping query workloads

Predicate ranges

Generates points using predicate ranges

Workload-aware points

Creates ranges that cover the points

(b) Case 2: Scattered query workloads

Figure 2: QuickSel’s subpopulation creation. Due to the property of mixture model (i.e., subpopulations may overlap with one another), creating subpopulations is straightforward for diverse query workloads.

\[
\int_{B_i} f(x) \, dx = \int_{B_i} \sum_{z=1}^{m} w_z g_z(x) \, dx = \sum_{z=1}^{m} w_z \int_{B_i} g_z(x) \, dx
\]

\[
= \sum_{z=1}^{m} w_z \int \frac{1}{|G_z|} I(x \in G_z \cap B_i) \, dx = \sum_{z=1}^{m} w_z \frac{|G_z \cap B_i|}{|G_z|}
\]

Recall that both \(G_z\) and \(B_i\) are represented by hyperrectangles. Thus, their intersection is also a hyperrectangle, and computing its size is straightforward.

3.3 Subpopulations from Observed Queries

We describe QuickSel’s approach to determining the boundaries of \(G_z\) for \(z = 1, \ldots, m\). Note that determining \(G_z\) is orthogonal to the model training process, which we describe in §4; thus, even if one devises an alternative approach to creating \(G_z\), our fast training method is still applicable.

QuickSel creates \(m\) hyperrectangular ranges\(^9\) (for the supports of its subpopulations) in a way that satisfies the following simple criterion: if more predicates involve a point \(x\), use a larger number of subpopulations for \(x\). Unlike query-driven histograms, QuickSel can easily pursue this goal by exploiting the property of a mixture model: the supports of subpopulations may overlap with one another.

In short, QuickSel generates multiple points (using predicates) that represent the query workloads and create hyperrectangles that can sufficiently cover those points. Specifically, we propose two approaches for this: a sampling-based one and a clustering-based one. The sampling-based approach is faster; the clustering-based approach is more accurate. Each of these is described in more detail below.

**Sampling-based** This approach performs the following operations for creating \(G_z\) for \(z = 1, \ldots, m\).

1. Within each predicate range, generate multiple random points \(r\). Generating a large number of random points increases the consistency; however, QuickSel limits the number to 10 since having more than 10 points did not improve accuracy in our preliminary study.

2. Use simple random sampling to reduce the number of points to \(m\), which serves as the centers of \(G_z\) for \(z = 1, \ldots, m\).

3. The length of the \(i\)-th dimension of \(G_z\) is set to twice the average of the distances (in the same \(i\)-th dimension) to the 10 nearest-neighbor centers.

**Clustering-based** The second approach relies on a clustering algorithm for generating hyperrectangles:

1. Do the same as the sampling-based approach.

2. Cluster \(r\) into \(m\) groups. (We used K-means++)

3. For each of \(m\) groups, we create the smallest hyperrectangle \(G_z\) that covers all the points belonging to the group.

Note that since each \(r\) belongs to a cluster and we have created a hyperrectangle that fully covers each cluster, the union

\(9\)The number \(m\) of subpopulations is set to \(\min(4 \cdot n, 4000)\), by default.
of the hyperrectangles covers all \( r \). Our experiments primarily use the sampling-based approach due to its efficiency, but we also compare them empirically in §5.7.

The following section describes how to assign the weights (i.e., \( h(x) = w_z \)) of these subpopulations.

4 QUICKSEL: MODEL TRAINING

This section describes how to compute the weights \( w_z \) of QuickSel’s subpopulations. For training its model, QuickSel finds the model that maximizes uniformity while being consistent with the observed queries. In §4.1, we formulate an optimization problem based on this criteria. Next, §4.2 presents how to solve the problem efficiently.

4.1 Training as Optimization

This section formulates an optimization problem for QuickSel’s training. Let \( g_0(x) \) be the uniform distribution with support \( B_0 \); that is, \( g_0(x) = 1/|B_0| \) if \( x \in B_0 \) and 0 otherwise. QuickSel aims to find the model \( f(x) \), such that the difference between \( f(x) \) and \( g_0(x) \) is minimized while being consistent with the observed queries.

There are many metrics that can measure the distance between two probability density functions \( f(x) \) and \( g_0(x) \), such as the earth mover’s distance [89], Kullback-Leibler divergence [62], the mean squared error (MSE), the Hellinger distance, and more. Among them, QuickSel uses MSE (which is equivalent to \( L2 \) distance between two distributions) since it enables the reduction of our originally formulated optimization problem (presented shortly: Problem 2) to a quadratic programming problem, which can be solved efficiently by many off-the-shelf optimization libraries [1, 4, 5, 14]. Also, minimizing MSE between \( f(x) \) and \( g_0(x) \) is closely related to maximizing the entropy of \( f(x) \) [52, 75, 86, 93]. See §6 for the explanation of this relationship.

MSE between \( f(x) \) and \( g_0(x) \) is defined as follows:

\[
\text{MSE}(f(x), g_0(x)) = \int (f(x) - g_0(x))^2 \, dx
\]

Recall that the support for \( g_0(x) \) is \( B_0 \). Thus, QuickSel obtains the optimal weights by solving the following problem.

**Problem 2 (QuickSel’s Training)** QuickSel obtains the optimal parameter \( w \) for its model by solving:

\[
\begin{align*}
\text{arg min}_{w} & \int_{x \in B_0} \left( f(x) - \frac{1}{|B_0|} \right)^2 \, dx \quad (3) \\
\text{such that} & \int_{B_i} f(x) \, dx = s_i \quad \text{for } i = 1, \ldots, n \quad (4) \\
& f(x) \geq 0 \quad (5)
\end{align*}
\]

Here, (5) ensures \( f(x) \) is a proper probability density function.

4.2 Efficient Optimization

We first describe the challenges in solving Problem 2. Then, we describe how to overcome the challenges.

**Challenge** Solving Problem 2 in a naïve way is computationally intractable. For example, consider a mixture model consisting of (only) two subpopulations represented by \( G_1 \) and \( G_2 \), respectively. Then, \( \int_{x \in B_0} (f(x) - g_0(x))^2 \, dx \) is:

\[
\int_{x \in G_1 \cap G_2} \left( \frac{w_1 + w_2}{|G_1 \cap G_2|} - g_0(x) \right)^2 \, dx + \int_{x \in G_1 \cap \neg G_2} (\cdots)^2 \, dx \\
+ \int_{x \in \neg G_1 \cap G_2} (\cdots)^2 \, dx + \int_{x \in \neg G_1 \cap \neg G_2} \left( \frac{0}{|\neg G_1 \cap \neg G_2|} - g_0(x) \right)^2 \, dx
\]

Observe that with this approach, we need four separate integrations only for two subpopulations. In general, the number of integrations is \( O(2^m) \), which is \( O(2^n) \). Thus, this direct approach is computationally intractable.

**Conversion One: Quadratic Programming** Problem 2 can be solved efficiently by exploiting the property of the distance metric of our choice (i.e., MSE) and the fact that we use uniform distributions for subpopulations (i.e., UMM). The following theorem presents the efficient approach.

**Theorem 1** The optimization problem in Problem 2 can be solved by the following quadratic optimization:

\[
\text{arg min}_{w} \quad w^T Q w \quad \text{such that} \quad Aw = s, \quad w \succeq 0
\]

where \((Q)_{ij} = |G_i \cap G_j| / (|G_i||G_j|)\), and \((A)_{ij} = |B_i \cap G_j| / |G_j|\). The bendy inequality sign \( (\succeq) \) means that every element of the vector on the left-hand side is equal to or larger than the corresponding element of the vector on the right-hand side.

**Proof.** This theorem can be shown by substituting the definition of QuickSel’s model (Equation (2)) into the probability density function \( f(x) \) in Equation (3). Note that minimizing \( (f(x) - g_0(x))^2 \) is equivalent to minimizing \( (f(x) - 2g_0(x))^2 \), which is also equivalent to minimizing \( (f(x))^2 \) since \( g_0(x) \) is constant over \( B_0 \) and \( \int f(x) \, dx = 1 \).

The integration of \( (f(x))^2 \) over \( B_0 \) can be converted to a matrix multiplication, as shown below:

\[
\int (f(x))^2 \, dx = \int \left[ \sum_{i=1}^{m} w_z I(x \in G_z) \right] \, dx \\
= \int \sum_{i=1}^{m} \sum_{j=1}^{m} w_i w_j I(x \in G_i) I(x \in G_j) \, dx
\]
which can be simplified to
\[
\sum_{i=1}^{m} \sum_{j=1}^{m} \frac{w_i w_j}{|G_i \cap G_j|} |G_i \cap G_j|
\]
\[
= \begin{bmatrix} w_1 \gamma_{[G_1 \cap G_1]} & \ldots & w_1 \gamma_{[G_1 \cap G_m]} \\ w_2 \gamma_{[G_2 \cap G_1]} & \ldots & w_2 \gamma_{[G_2 \cap G_m]} \\ \vdots & \vdots & \vdots \\ w_m \gamma_{[G_m \cap G_1]} & \ldots & w_m \gamma_{[G_m \cap G_m]} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}
\]
\[
= w^\top Q w
\]

Second, we express the equality constraints in an alternative form. Note that the left-hand side of each equality constraint, i.e., \( \int_{B_i} f(x) \, dx \), can be expressed as:
\[
\int_{B_i} f(x) \, dx = \int_{B_i} \sum_{j=1}^{m} \frac{w_j}{|G_j|} I(x \in G_j) \, dx
\]
\[
= \sum_{j=1}^{m} \frac{w_j}{|G_j|} \int_{B_i} I(x \in G_j) \, dx = \sum_{j=1}^{m} \frac{w_j}{|G_j|} |B_i \cap G_j|
\]
\[
= \begin{bmatrix} |B_i \cap G_1| & \ldots & |B_i \cap G_m| \\ |B_2 \cap G_1| & \ldots & |B_2 \cap G_m| \\ \vdots & \vdots & \vdots \\ |B_m \cap G_1| & \ldots & |B_m \cap G_m| \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}
\]
\[
= s
\]

Then, the equality constraints, i.e., \( \int_{B_i} f(x) \, dx = s_i \) for \( i = 1, \ldots, n \), can be expressed as follows:
\[
\begin{bmatrix} |B_1 \cap G_1| & \ldots & |B_1 \cap G_m| \\ |B_2 \cap G_1| & \ldots & |B_2 \cap G_m| \\ \vdots & \vdots & \vdots \\ |B_n \cap G_1| & \ldots & |B_n \cap G_m| \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = s
\]

Finally, \( w^\top 1 = 1 \) if and only if \( \int f(x) \, dx = 1 \), and \( w \succeq 0 \) for arbitrary \( G \) if and only if \( \int f(x) \, dx \geq 0 \).

The implication of the above theorem is significant: we could reduce the problem of \( O(2^n) \) complexity to the problem of only \( O(n^2) \) complexity.

Conversion Two: Moving Constraints The quadratic programming problem in Theorem 1 can be solved efficiently by most off-the-shelf optimization libraries; however, we can solve the problem even faster by converting the problem to an alternative form. We first present the alternative problem, then discuss it.

Problem 3 (QuickSel’s QP) QuickSel solves this problem alternative to the quadratic programming problem in Theorem 1:

\[
\arg \min_w \ell(w) = w^\top Q w + \lambda \|Aw - s\|^2
\]

where \( \lambda \) is a large real value (QuickSel uses \( \lambda = 10^6 \)).

In formulating Problem 3, two types of conversions are performed: (1) the consistency with the observed queries (i.e., \( Aw = s \)) is moved into the optimization objective as a penalty term, and (2) the positivity of \( f(x) \) is not explicitly specified (by \( w \succeq 0 \)). These two types of conversions have little impact on the solution for two reasons. First, to guarantee the consistency, a large penalty (i.e., \( \lambda = 10^6 \)) is used. Second, the mixture model \( f(x) \) is bound to approximate the true distribution, which is always non-negative. We empirically examine the advantage of solving Problem 3 (instead of solving the problem in Theorem 1 directly) in §5.7.

The solution \( w^* \) to Problem 3 can be obtained in a straightforward way by setting its gradients of the objective (with respect to \( w \)) equal to 0:
\[
\frac{\partial \ell(w^*)}{\partial w} = 2 Q w^* + 2 \lambda A^\top (A w^* - s) = 0
\]
\[
\Rightarrow w^* = (Q + \lambda A^\top A)^{-1} \lambda A s
\]

Observe that \( w^* \) is expressed in a closed form; thus, we can obtain \( w^* \) analytically instead of using iterative procedures typically required for general quadratic programming.

5 EXPERIMENT

In this section, we empirically study QuickSel. In summary, our results show the following:

1. End-to-end comparison against other query-driven methods: QuickSel was significantly faster (34.0×–179.4×) for the same accuracy—and produced much more accurate estimates (26.8%–91.8% lower error) for the same time limit—than previous query-driven methods. (§5.2)
2. Comparison against periodic database scans: For the same storage size, QuickSel’s selectivity estimates were 77.7% and 91.3% more accurate than scan-based histograms and sampling, respectively. (§5.3)
3. Impact on PostgreSQL performance: Using QuickSel for PostgreSQL makes the system 2.25× faster (median) than the default. (§5.4)
4. Effectiveness of QuickSel’s mixture model: QuickSel’s model produced considerably more accurate estimates than histograms given the same number of parameters. (§5.5)
5. Robustness to workload shifts: QuickSel’s accuracy quickly recovers after sudden workload shifts. (§5.6)
6. Optimization efficiency: QuickSel’s optimization method (Problem 3) was 1.5×–17.2× faster than solving the standard quadratic programming. (§5.7)

5.1 Experimental Setup

Methods Our experiments compare QuickSel to six other selectivity estimation methods.
**Query-driven Methods:**

1. STHoles [19]: This method creates histogram buckets by partitioning existing buckets (as in Figure 1). The frequency of an existing bucket is distributed uniformly among the newly created buckets.
2. ISOMER [93]: This method applies STHoles for histogram bucket creations, but it computes the optimal frequencies of the buckets by finding the maximum entropy distribution. Among existing query-driven methods, ISOMER produced the highest accuracy in our experiments.
3. ISOMER+QP: This method combines ISOMER’s approach for creating histogram buckets and QuickSel’s quadratic programming (Problem 3) for computing the optimal bucket frequencies.
4. QueryModel [13]: This method computes the selectivity estimate by a weighted average of the selectivities of observed queries. The weights are determined based on the similarity of the new query and each of the queries observed in the past.

**Scan-based Methods:**

5. AutoHist: This method creates an equiwidth multidimensional histogram by scanning the data. It also updates its histogram whenever more than 20% of the data changes (this is the default setting with SQL Server’s AUTO_UPDATE_STATISTICS option [8]).
6. AutoSample: This method relies on a uniform random sample of data to estimate selectivities. Similar to AutoHist, AutoSample updates its sample whenever more than 10% of the data changes.

We have implemented all methods in Java.

**Datasets and Query Sets**

We use two real datasets and one synthetic dataset in our experiments, as follows:

1. DMV: This dataset contains the vehicle registration records of New York State [95]. It contains 11,944,194 rows. Here, the queries ask for the number of valid registrations for vehicles produced within a certain date range. Answering these queries involves predicates on three attributes: model_year, registration_date, and expiration_date.
2. Instacart: This dataset contains the sales records of an online grocery store [94]. We use their orders table, which contains 3.4 million sales records. Here, the queries ask for the reorder frequency for orders made during different hours of the day. Answering these queries involves predicates on two attributes: order_hour_of_day and days_since_prior. (In §5.3, we use more attributes (up to ten).)
3. Gaussian: We also generated a synthetic dataset using a bivariate dimensional normal distribution. We varied this dataset to study our method under workload shifts, different degrees of correlation between the attributes, and more. Here, the queries count the number of points that lie within a randomly generated rectangle.

For each dataset, we measured the estimation quality using 100 test queries not used for training. The ranges for selection predicates (in queries) were generated randomly within a feasible region; the ranges of different queries may or may not overlap.

**Environment**

All our experiments were performed on m5.4xlarge EC2 instances, with 16-core Intel Xeon 2.5GHz and 64 GB of memory running Ubuntu 16.04.

**Metrics**

We use the root mean square (RMS) error:

\[
\text{RMS error} = \left( \frac{1}{t} \sum_{i=1}^{t} (\text{true}_{sel} - \text{est}_{sel})^2 \right)^{1/2}
\]

where \( t \) is the number of test queries. We report the RMS errors in percentage (by treating both \( \text{true}_{sel} \) and \( \text{est}_{sel} \) as percentages).

When reporting training time, we include the time required for refining a model using an additional observed query, which itself includes the time to store the query and run the necessary optimization routines.

### 5.2 Selectivity Estimation Quality

In this section, we compare the end-to-end selectivity estimation quality of QuickSel versus query-driven histograms. Specifically, we gradually increased the number of observed queries provided to each method from 10 to 1,000. For each number of observed queries, we measured the estimation error and training time of each method using 100 test queries.

These results are reported in Figure 3. Given the same number of observed queries, QuickSel’s training was significantly faster (Figures 3a and 3d), while still achieving comparable estimation errors (Figures 3b and 3e). We also studied the relationship between errors and training times in Figures 3c and 3f, confirming QuickSel’s superior efficiency (STHoles, ISOMER+QP, and QueryModel are omitted in these figures due to their poor performance). In summary, QuickSel was able to quickly learn from a large number of observed queries (i.e., shorter training time) and produce highly accurate models.

### 5.3 Comparison to Scan-based Methods

We also compared QuickSel to two automatically-updating scan-based methods, AutoHist and AutoSample, which incorporate SQL Server’s automatic updating rule into equiwidth multidimensional histograms and samples, respectively. Since both methods incur an up-front cost for obtaining their statistics, they should produce relatively more accurate estimates initially (before seeing new queries). In
contrast, the accuracy of QuickSel’s estimates should quickly improve as new queries are observed.

To verify this empirically, we first generated a Gaussian dataset (1 million tuples) with correlation 0. We then inserted 200K new tuples generated from a distribution with a different correlation after processing 200 queries, and repeated this process. In other words, after processing the first 100 queries, we inserted new data with correlation 0.1; after processing the next 100 queries, we inserted new data with correlation 0.2; and continued this process until a total of 1000 queries were processed. We performed this process for each method under comparison. QuickSel adjusted its model each time after observing 100 queries. AutoHist and AutoSample updated their statistics after each batch of data insertion. QuickSel and AutoHist both used 100 parameters (# of subpopulations for the mixture model and # of buckets for histograms); AutoSample used a sample of 100 tuples.

Figure 4a shows the error of each method. As expected, AutoHist produced more accurate estimates initially. However, as more queries were processed, the error of QuickSel drastically decreased. In contrast, the errors of AutoSample and AutoHist did not improve with more queries, as they only depend on the frequency at which a new scan (or sampling) is performed. After processing only 100 queries (i.e., initial update), QuickSel produced more accurate estimates than both AutoHist and AutoSample. On average (including the first 100 queries), QuickSel was 71.4% and 89.8% more accurate than AutoHist and AutoSample, respectively. This is consistent with the previously reported observations that query-driven methods yield better accuracy than scan-based ones [19]. (The reason why query-driven proposals have not been widely adopted to date is due to their prohibitive cost; see §7.2).

In addition, Figure 4b compares the update times of the three methods. By avoiding scans, QuickSel’s query-driven updates were 525× and 243× faster than AutoHist and AutoSample, respectively.

Finally, we studied how the performance of those methods changed as we increased the data dimension (i.e., the number of attributes appearing in selection predicates). First, using the Instacart dataset, we designed each query to target a
random subset of dimensions (N/2) as increasing the dimension N from 2 to 10. In all test cases (Figure 4c left), QuickSel’s accuracy was consistent, showing its ability to scale to high-dimensional data. Also in this experiment, QuickSel performed significantly better than, or comparably to, histograms and sampling. We could also obtain a similar result using the Gaussian dataset (Figure 4c right). This consistent performance across different data dimensions is primarily due to how QuickSel is designed; that is, its estimation only depends on how much queries overlap with one another.

### 5.4 Impact on Query Performance

This section examines QuickSel’s impact on query performance. That is, we test if QuickSel’s more accurate selectivity estimates can lead to improved query performance for actual database systems (i.e., shorter latency).

To measure the actual query latencies, we used PostgreSQL ver. 10 with a third-party extension, called pg_hint_plan [6]. Using this extension, we enforced our own estimates (for PostgreSQL’s query optimization) in place of the default ones. We compared PostgreSQL Default (i.e., no hint) and QuickSel—to measure the latencies of the following join query in processing the Instacart dataset:

```sql
select count(*)
from S inner join T on S.tid = T.tid
inner join U on T.uid = U.uid
where (range_filter_on_T)
and (range_filter_on_S);
```

where the joins keys for the tables S, T, and U were in the PK-FK relationship, as described by the schema (of Instacart).

Figure 5 shows the speedups QuickSel could achieve in comparison to PostgreSQL Default. Note that QuickSel does not improve any underlying I/O or computation speed; its speedups are purely from helping PostgreSQL’s query optimizer choose a more optimal plan based on improved selectivity estimates. Even so, QuickSel could bring a 2.25× median speedup, with a 3.47× max speedup. In the worst case, PostgreSQL with QuickSel was almost identical to PostgreSQL Default (i.e., 0.98× speedup).

### 5.5 QuickSel’s Model Effectiveness

In this section, we compare the effectiveness of QuickSel’s model to that of models used in the previous work. Specifically, the effectiveness is assessed by (1) how the model size—its number of parameters—grows as the number of observed queries grows, and (2) how quickly its error decreases as its number of parameters grows.

Figure 6a reports the relationship between the number of observed queries and the number of model parameters. As discussed in §2.3, the number of buckets (hence, parameters)
of ISOMER increased quickly as the number of observed queries grew. STHoles was able to keep the number of its parameters small due to its bucket merging technique; however, this had a negative impact on its accuracy. Here, QuickSel used the least number of model parameters. For instance, when 100 queries were observed for DMV, QuickSel had 10× fewer parameters than STHoles and 56× fewer parameters than ISOMER.

We also studied the relationship between the number of model parameters and the error. The lower the error (for the same number of model parameters), the more effective the model. Figure 6b shows the result. Given the same number of model parameters, QuickSel produced significantly more accurate estimates. Equivalently, QuickSel produced the same quality estimates with much fewer model parameters.

5.6 Robustness to Workload Shifts

In this section, we test QuickSel’s performance under significant workload shifts. That is, after observing a certain number of queries (i.e., 100 queries) around a certain region of data, the query workload suddenly jumps to a novel region. This pattern repeats several times.

Figure 7 shows the result. Here, we could observe the following pattern. QuickSel’s error increased significantly right after each jump (i.e., at query sequence #100 and at #200), producing 1.5×-3.6× higher RMS errors compared to histograms. However, QuickSel’s error dropped quickly, achieving 12×-378× lower RMS errors than histograms. This was possible due to QuickSel’s faster adaptation.
### Table 3: Comparison of selectivity estimation methods

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model</th>
<th>Method</th>
<th>Key Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Based on Database Scans</strong></td>
<td>Histograms</td>
<td>Multi-dim Hist [27, 42, 70]</td>
<td>Introduces multidimensional histograms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Muralikrishna [79]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Van Gelder [106]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GOH [44]</td>
<td>Estimates join selectivity with histograms for important domains</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thaper [101]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To [102]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sampling</td>
<td>Lipton [68]</td>
<td>Introduces adaptive sampling for high accuracy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Haas [37]</td>
<td>Uses sampling for join selectivity estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Riondato [87]</td>
<td>Guarantees accuracy relying on the VC-dimension of queries</td>
</tr>
<tr>
<td></td>
<td>ML</td>
<td>KDE [34, 35, 40]</td>
<td>Applies kernel density estimation to selectivity estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PGM [32, 92, 104]</td>
<td>Uses probabilistic graphical models for selectivity estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neural Net [56, 69]</td>
<td>Trains a neural network for selectivity estimation</td>
</tr>
<tr>
<td><strong>Based on Observed Queries</strong></td>
<td>Error-feedback Histograms</td>
<td>ST-histogram [11]</td>
<td>Refines the bucket frequencies based on the errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STHistograms [19]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STHoles [19]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SASH [66]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>QueryModel [13]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max-Entropy Histograms (accurate but slow)</td>
<td>ISOMER [74, 75, 93]</td>
<td>Finds a maximum entropy distribution consistent with observed queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kaushik et al. [52]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ré et al. [85, 86]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mixture Model (fast &amp; accurate)</td>
<td>QuickSel (Ours)</td>
<td>Employs a mixture model for selectivity estimation; develops an efficient training algorithm for the new model</td>
</tr>
</tbody>
</table>

7 RELATED WORK

There is extensive work on selectivity estimation due to its importance for query optimization. In this section, we review both scan-based (§7.1) and query-driven methods (§7.2). QuickSel belongs to the latter category. We have summarized the related work in Table 3.

#### 7.1 Database Scan-based Estimation

As explained in §1, we use the term scan-based methods to refer to techniques that directly inspect the data (or part of it) for collecting their statistics. These approaches differ from query-based methods which rely only on the actual selectivities of the observed queries.

**Scan-based Histograms** These approaches approximate the joint distribution by periodically scanning the data. There has been much work on how to efficiently express the joint distribution of multidimensional data [23, 25, 27, 33–35, 37, 39, 40, 42–44, 49, 51, 64, 68, 70, 77, 79, 87, 101, 102, 106]. There is also some work on histograms for special types of data, such as XML [10, 16, 107, 109], spatial data [48, 57–59, 72, 80, 97, 99, 100, 108, 112, 113], graph [29], string [45–47, 76]; or for privacy [38, 63, 65].

**Sampling** Sampling-based methods rely on a sample of data for estimating its joint distribution [37, 68, 87]. However, drawing a new random sample requires a table-scan or random retrieval of tuples, both of which are costly operations and hence, are only performed periodically.
Machine Learning Models  KDE is a technique that translates randomly sampled data points into a distribution [91]. In the context of selectivity estimation, KDE has been used as an alternative to histograms [34, 35, 40]. Like mixture models (MM), KDE also expresses a probability density function as a summation of some basis functions. However, KDE and MM are fundamentally different. KDE relies on independent and identically distributed samples, and hence lends itself to scan-based selectivity estimation. In contrast, MM does not require any sampling and can thus be used in query-driven selectivity estimation (where sampling is not practical). Similarly, probabilistic graphical models [32, 92, 104], neural networks [56, 69], and tree-based ensembles [28] have been used for selectivity estimation. Unlike histograms, these approaches can capture column correlations more succinctly. However, applicability of these models for query-driven selectivity estimation has not been explored and remains unclear.

More recently, sketching [20] and probe executions [103] have been proposed, which differ from ours in that they build their models directly using the data (not query results). Similar to histograms, using the data requires either periodic updates or higher query processing overhead. QuickSel avoids both of these shortcomings with its query-driven MM.

7.2 Query-driven Estimation
Query-driven techniques create their histogram buckets adaptively according to the queries they observe in the workload. These can be further categorized into two techniques based on how they compute their bucket frequencies: error-feedback histograms and max-entropy histograms.

Error-feedback Histograms  Error-feedback histograms [11, 13, 19, 54, 55, 66] adjust bucket frequencies in consideration of the errors made by old bucket frequencies. They differ in how they create histogram buckets according to the observed queries. For example, STHoles [19] splits existing buckets with the predicate range of the new query. SASH [66] uses a space-efficient multidimensional histogram, called MHIST [27], but determines its bucket frequencies with an error-feedback mechanism. QueryModel [13] treats the observed queries themselves as conceptual histogram buckets and determines the distances among those buckets based on the similarities among the queries’ predicates.

Max-Entropy Histograms  Max-entropy histograms [52, 74, 75, 86, 93] find a maximum entropy distribution consistent with the observed queries. Unfortunately, these methods generally suffer from the exponential growth in their number of buckets as the number of observed queries grows (as discussed in §2). QuickSel avoids this problem by relying on mixture models.

Fitting Parametric Functions  Adaptive selectivity estimation [22] fits a parametric function (e.g., linear, polynomial) to the observed queries. This approach is more applicable when we know the data distribution a priori, which is not assumed by QuickSel.

Self-tuning Databases  Query-driven histograms have also been studied in the context of self-tuning databases [60, 71, 73]. IBM’s LEO [96] corrects errors in any stage of query execution based on the observed queries. Microsoft’s AutoAdmin [12, 21] focuses on automatic physical design, self-tuning histograms, and monitoring infrastructure. Part of this effort is ST-histogram [11] and STHoles [19] (see Table 3). DBL [81] and IDEA [30] exploit the answers to past queries for more accurate approximate query processing. QueryBot 5000 [71] forecasts the future queries, whereas OtterTune [105] and index [61] use machine learning for automatic physical design and building secondary indices, respectively.

8 CONCLUSION AND FUTURE WORK
The prohibitive cost of query-driven selectivity estimation techniques has greatly limited their adoption by DBMS vendors, which for the most part still rely on scan-based histograms and samples that are periodically updated and are otherwise stale. In this paper, we proposed a new framework, called selectivity learning or QuickSel, which learns from every query to continuously refine its internal model of the underlying data, and therefore produce increasingly more accurate selectivity estimates over time. QuickSel differs from previous query-driven selectivity estimation techniques by (i) not using histograms and (ii) enabling extremely fast refinements using its mixture model. We formally showed that the training cost of our mixture model can be reduced from exponential to only quadratic complexity (Theorem 1).

Supporting Complex Joins  When modeling the selectivity of join queries, even state-of-the-art modes [28, 56, 111] take a relatively simple approach: conceptually prejoining corresponding tables and constructing a joint probability distribution over each join pattern. We plan to similarly extend our current formulation of QuickSel to model the selectivity of general joins.

9 ACKNOWLEDGEMENT
This material is based upon work supported by the National Science Foundation under Grant No. 1629397 and the Michigan Institute for Data Science (MIDAS) PODS. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
REFERENCES


