Theoretical Analysis of Learned Database Operations under Distribution Shift through Distribution Learnability

Sepanta Zeighami [1] **Cyrus Shahabi** [2]

Abstract

Use of machine learning to perform database operations, such as indexing, cardinality estimation, and sorting, is shown to provide substantial performance benefits. However, when datasets change and data distribution shifts, empirical results also show performance degradation for learned models, possibly to worse than non-learned alternatives. This, together with a lack of theoretical understanding of learned methods undermines their practical applicability, since there are no guarantees on how well the models will perform after deployment. In this paper, we present the first known theoretical characterization of the performance of learned models in dynamic datasets, for the aforementioned operations. Our results show novel theoretical characteristics achievable by learned models and provide bounds on the performance of the models that characterize their advantages over non-learned methods, showing why and when learned models can outperform the alternatives. Our analysis develops the *distribution learnability* framework and novel theoretical tools which build the foundation for the analysis of learned database operations in the future.

1. Introduction

Given a fixed dataset, learned database operations (machine learning models learned to perform database operations such as indexing, cardinality estimation and sorting) have been shown to outperform non-learned methods, providing speedups and space savings both empirically (Kraska et al., 2018; Kipf et al., 2018; Kristo et al., 2020) and, for the case of indexing, theoretically (Zeighami & Shahabi, 2023; Ferragina et al., 2020). For dynamic datasets (e.g., when new points can be inserted into the dataset), significant empirical benefits are also often observed when using learned methods. However, an important caveat accompanying these results is that, especially when data distribution changes, models' performance may deteriorate after new insertions (Ding et al., 2020; Negi et al., 2023; Wang et al., 2021), possibly to worse than non-learned methods (Wongkham et al., 2022). This, combined with the lack of a theoretical understanding of the behavior of the learned models as datasets change, poses a critical hurdle to their deployment in practice. It is theoretically unclear why and when learned models outperform non-learned methods, and, until this paper, no theoretical work shows any advantage in using the learned methods in dynamic datasets and under distribution shift. The goal of this paper is to theoretically understand the capabilities of learned models for database operations, show why and when they outperform non-learned alternatives and provide theoretical guarantees on their performance.

We specifically study learned solutions for three fundamental database operations: indexing, cardinality estimation and sorting. Our main focus is the study of learned indexing and cardinality estimation in the presence of insertions from a possibly changing data distribution, while we also study learned sorting (in static scenario) to show the broader applicability of our developed theoretical tools. In all cases, a learned model, $\hat{f}(x;\theta)$ is used to replace a specific data operation, $f_{\boldsymbol{D}}(x)$, that takes an input x and calculates a desired answer from the dataset $\boldsymbol{D}$. For cardinality estimation, $f_{\boldsymbol{D}}(x)$ returns the number of points in the database $\boldsymbol{D}$ that match the query x, and for indexing $f_{\boldsymbol{D}}(x)$ returns the true location of x in a sorted array. The model $\hat{f}(x;\theta)$ is trained to approximate $f_{\boldsymbol{D}}(x)$, and an accurate approximation leads to efficiency gains when using the model (e.g., for learned indexing, if $\hat{f}(x;\theta)$ gives an accurate estimate of location of x in a sorted array, a local search around the estimated location efficiently finds the exact location). In the presence of insertions, the ground-truth $f_{\boldsymbol{D}}(x)$ changes as the dataset changes (e.g., the cardinality of some queries increase as new points are inserted). Thus, as more points are inserted (not only due to distribution shift, but exacerbated by it), the accuracy of $\hat{f}(x;\theta)$ worsens. A common solution is to periodically retrain $\hat{f}$ to ensure consistent accuracy. This, however, increases insertion cost when insertions trigger a (computationally expensive) model retraining.

[1]University of California, Berkeley. Work done while at USC's Infolab [2]University of Southern California. Correspondence to: Sepanta Zeighami <zeighami@berkeley.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Learned Operation	Query Complexity	Insertion Complexity	Space Complexity
Indexing	$\mathcal{T}_n^{\mathfrak{X}} \log\log n + \log \delta n$	$\mathcal{T}_n^{\mathfrak{X}} \log\log n + \log \delta n + \mathcal{B}_n^{\mathfrak{X}} \log^2 \log n$	$n \log n^{\dagger}$
CE, d-dim, $\epsilon = \Omega(\sqrt{n})$	$\mathcal{T}_n^{\mathfrak{X}}$	$\max\{\frac{\delta n}{\epsilon}, 1\}\mathcal{B}_n^{\mathfrak{X}}$	$\mathcal{S}_n^{\mathfrak{X}}$
CE, 1-dim	$\mathcal{T}_{\epsilon^2}^{\mathfrak{X}} + \log n$	$\max\{\delta\epsilon, 1\}\mathcal{B}_{\epsilon^2}^{\mathfrak{X}} + \log n$	$\frac{n}{\epsilon^2}\mathcal{S}_{\epsilon^2}^{\mathfrak{X}} + \frac{n}{\epsilon^2}\log n$
Sorting	$\mathcal{T}_n^{\mathfrak{X}} n \log\log n^{\dagger}$		$\mathcal{S}_{\sqrt{n}}^{\mathfrak{X}} + n \log n$
Sorting, appx. known dist.	$\mathcal{T}^{\mathfrak{X}} n$		$\mathcal{S}^{\mathfrak{X}}$

Table 1. Summary of results for data sampled from a distribution learnable class $\mathfrak{X}$ (CE: cardinality estimation, †: for simplicity assuming $\mathcal{S}_n^{\mathfrak{X}}, \mathcal{B}_n^{\mathfrak{X}}$ are at most linear in data size, see Theorem 4.1 and Theorem 4.5 for general cases).

Theoretically, the relationship between accuracy change and new data insertion has not been well understood, leading to a lack of meaningful theoretical guarantees for learned methods in the presence of insertions. The only existing guarantees are by the PGM index (Ferragina & Vinciguerra, 2020), which achieves a worst-case insertion time of $O(\log n)$ with worst-case query time of $O(\log^2 n)$. Although experimental results show PGM often outperforms B-trees in practice (Ferragina & Vinciguerra, 2020; Wongkham et al., 2022), the theoretical guarantees are worse than those of a B-tree (that supports both insertions and queries in $O(\log n)$). Such theoretical guarantees do not meaningfully characterize the index's performance in practice nor show why and when the learned model performs better (or worse) than B-trees.

In this paper, we present the first known theoretical characterization of the performance of learned models for indexing and cardinality estimation in the presence of insertions, painting a thorough picture of why and when they outperform non-learned alternatives for these fundamental database operations. Our analysis develops the notion of *distribution learnability*, a characteristic of data distributions that helps quantify learned database operation's performance for data form such distributions. Using this notion, our results are distribution dependent (as one expects bounds on learned operations should be), without making unnecessary assumptions about data distribution. Our developed theoretical framework builds a foundation for the analysis of learned database operations in the future. To show its broader applicability, we present a theoretical analysis of learned sorting, showing its theoretical characteristics and proving why and when it outperforms non-learned methods.

1.1. Summary of Results

Table 1 summarizes our results in the following setting. Suppose n data points are sampled independently from distributions $\chi_1, ..., \chi_n$, and let the distribution class $\mathfrak{X} = \{\chi_1, ..., \chi_n\}$. The points are inserted one by one into a dataset. $\boldsymbol{D}^i$ denotes the dataset after i insertion. Our goal is to efficiently answer cardinality estimation and indexing queries on $\boldsymbol{D}^i$ accurately for any i, i.e., as new points are being inserted. We denote *distribution shift* by $\delta \in [0, 1]$ (defined based on, and often equal, to total variation distance) where $\delta = 0$ means no distribution shift. Table 1 also contains results for sorting, where the goal is to sort the fixed array $\boldsymbol{D}^n$, and the reported results are the time and space complexity of doing so. For sorting only, we assume the samples are i.i.d. All results are expected complexities, with the expectation over sampling of the data, and the insertion complexity is amortized over n insertions. To obtain our results, we develop a novel theoretical framework, dubbed *distribution learnability*. We provide an informal discussion of the framework before discussing the results.

Distribution Learnability. At a high level, distribution learnability means *we can model a data distribution well*. This notion allows us to state our results in the form "*if we can model a data distribution well, learned database operations will perform well on data from that distribution*". Then, if one indeed proves that "*we can model the data distribution χ well*", our result immediately implies "*learned database operations will perform well on data coming from χ*". Crucially, our Theorem 3.5 shows that purely function approximation results (independent of the application of learned databases) imply distribution learnability, enabling us to utilize function approximation results to show the benefits of learned database operations.

More concretely (but still informally), we say a distribution class $\mathfrak{X}$ is *distribution learnable* with parameters $\mathcal{T}_n^{\mathfrak{X}}$, $\mathcal{S}_n^{\mathfrak{X}}$, $\mathcal{B}_n^{\mathfrak{X}}$, if given a set of observations, $\boldsymbol{D}^n$, from distributions in $\mathfrak{X}$, there exists a learning algorithm that returns an *accurate* model, $\hat{f}$, of the distributions in $\mathfrak{X}$, and that $\hat{f}$ can be evaluated in $\mathcal{T}_n^{\mathfrak{X}}$ operations, and takes space at most $\mathcal{S}_n^{\mathfrak{X}}$ to store. Furthermore, the learning algorithm takes time $n \times \mathcal{B}_n^{\mathfrak{X}}$ to learn $\hat{f}$, where $\mathcal{B}_n^{\mathfrak{X}}$ is the amortized training time. The notion is related to statistical estimation, but we also utilize it to characterize time and space complexity of modeling.

Results in Table 1 are stated for data sampled from any distribution learnable class $\mathfrak{X}$. For illustration, we summarize the results for two specific distribution classes: (1) distributions, $\mathfrak{X}_\rho$, with p.d.f bounded between $0 < \rho_1$ and $\rho_2 < \infty$, and (2) distributions, $\mathfrak{X}_c$, where the data distribution is known and probability of events can be calculated efficiently (e.g., distribution is known to be uniform or piece-wise polynomial). The first case formulates a realistic scenario for the data distribution (experimentally shown by Zeighami & Shahabi (2023)), while the second case presents a best case scenario for the learned models, showing what is possible

in favorable circumstances. Lemma 3.6 shows $\mathfrak{X}_\rho$, (and trivially) $\mathfrak{X}_c$ are distribution learnable, deriving the corresponding values for $\mathcal{T}_n^{\mathfrak{X}}$, $\mathcal{S}_n^{\mathfrak{X}}$ and $\mathcal{B}_n^{\mathfrak{X}}$ (See Table 2 for exact values). Next, we discuss Table 1 for $\mathfrak{X}_\rho$ and $\mathfrak{X}_c$, where we substitute the values of $\mathcal{T}_n^{\mathfrak{X}}$, $\mathcal{S}_n^{\mathfrak{X}}$ and $\mathcal{B}_n^{\mathfrak{X}}$ from Lemma 3.6 for $\mathfrak{X}_\rho$ and $\mathfrak{X}_c$, and discuss the resulting complexities.

Indexing. After substituting the complexities in the first row of Table 1 we obtain that for $\mathfrak{X}_\rho$ and $\mathfrak{X}_c$, query and insertion complexities are $O(\log\log n + \log(\delta n))$. To understand this result, consider the simple scenario with $\delta = 0$, where inserted items are sampled form a fixed distribution, and thus frequent model updates are not necessary. The result shows that a learned model performs insertions and queries in $O(\log\log n)$, showing their superiority over non-learned methods that perform queries and insertions in $O(\log n)$. Nonetheless, when there is a distribution shift, model performance worsens. In the worst-case and when $\delta = 1$, we see no advantage to using learned models over non-learned methods. This is not surprising, since learned models use current observations to make prediction about the future, and if the future distribution is drastically different, one should not be able to gain from using the current observations.

Cardinality Estimation. First, consider the second row in Table. 1, showing performance of learned models for cardinality estimation in high dimensions but when error is at least $\sqrt{n}$. Substituting the complexities in this row, for $\mathfrak{X}_c$, we obtain that learned models perform insertions and queries in $O(1)$ time and space in this setting. This is significant, given that a non-learned method such as sampling (and more broadly ϵ-approximations (Mustafa & Varadarajan, 2017)), even in this accuracy regime, needs space exponential in dimensionality (Wei & Yi, 2018). Nonetheless, modeling in high dimensions is difficult, and consequently this result requires the accuracy to be at least $\sqrt{n}$. Moreover, even for $\epsilon \geq \sqrt{n}$ but for more general distribution class of $\mathfrak{X}_\rho$, our results show that learned methods will also take space exponential in dimensionality (which is broadly needed, even for neural networks (Petersen & Voigtlaender, 2018), without further assumptions). We also mention that $\sqrt{n}$ has a statistical significance (see Sec. 3 for discussion), and appears in our analysis throughout. Second, we show that in 1-dimension (the third row of Table 1), learned models perform cardinality estimation queries effectively, where for $\mathfrak{X}_c$, a learned model can perform queries and insertions in $O(\log n)$ while taking space $O(\frac{n}{\epsilon^2}\log n)$. This result also shows that a learned approach outperforms the non-learned (and worst-case optimal) method discussed in (Wei & Yi, 2018) that takes space $O(\frac{n}{\epsilon}\log \epsilon n)$ to answer queries.

Sorting. Substituting complexities in the fourth row of Table 1, we obtain $O(n\log\log n)$ time complexity for $\mathfrak{X}_\rho$, using a method that is a variation of Kristo et al. (2020) that learns to sort through sampling. Our framework applies to this method because its study needs to consider the generalization of a model learned from samples (similar to how models need to generalize to a new dataset after insertions). Moreover, last row of Table 1 shows that, if we (approximately) know the data distribution, and the distribution can be efficiently evaluated and stored, we can sort an array in $O(n)$ ($\mathcal{T}^{\mathfrak{X}}$ is independent of data size), showing benefits of using data distribution to perform database operations.

To conclude, our results in Table 1 are more general than the two distribution classes discussed above. A major contribution of this paper is developing the distribution learnability framework that allows us to orthogonally study the two problems of modeling a data distribution (*the modeling problem*), and how learned models can be used to perform database operations with theoretical guarantees (*the model utilization problem*). Table 1 summarizes our contributions to the latter problem, while our results connecting distribution learnability to function approximation concepts (Theorem 3.5) is our contribution to the former. The rest of this paper discusses our developed framework and results in more detail, but for the sake of space, formal discussion, proofs, and a detailed discussion of the related work are differed to the appendix.

2. Preliminaries

2.1. Problem Setting

Setup. We study performing database operations on a possibly changing d-dimensional dataset. We either consider the setting when n data points are inserted one by one (dynamic setting), or that we are given a fixed set of n data points (static setting). We define $\boldsymbol{D}^i$ as the dataset after i insertions, and the final dataset, $\boldsymbol{D}^n$, is often denoted as $\boldsymbol{D}$. We study indexing, cardinality estimation and sorting operations.

For indexing, the goal is to build an index to store and find items in a 1-dimensional dataset. The index supports insertions and queries. That is, after i insertions, for any i, we can retrieve items from the dataset $\boldsymbol{D}^i$, where the query is either an exact match query or a range query. For cardinality estimation, the dataset is d-dimensional, and we support insertions and axis-parallel queries. That is, after i insertions, for any i, we would like to estimate the number of items in the dataset $\boldsymbol{D}^i$ that match a query q, where q defines an axis-parallel hyper-rectangle. Finally, the goal of sorting is to sort a fixed 1-dimensional array, $\boldsymbol{D}$, of size n. Indexing and sorting always return exact results (i.e., array has to be fully sorted after the operation), while cardinality estimation accepts an error of ϵ for the query answer estimates.

Data Distribution and Distribution Shift. We consider the case that the i-th data point is sampled independently from a distribution χ_i, and denote by $\boldsymbol{D} \sim \chi$ this sampling procedure, where $\chi = \{\chi_1, ..., \chi_n\}$. We say $\boldsymbol{D}$ was sampled from a distribution class $\mathfrak{X}$ if $\chi_i \in \mathfrak{X}\ \forall i$. We use total variation to quantify distribution shift. We say $\boldsymbol{D}$

was sampled from a distribution χ with distribution shift δ, when $\max_{\chi_i,\chi_j\in\chi}\|\chi_i-\chi_j\|_{TV}$, where $\|\chi_i-\chi_j\|_{TV}$ denoted the total variation (TV) distance between χ_i and χ_j. We also define total variation of a distribution set χ as $TV(\chi)=\sup_{\chi_i,\chi_j\in\chi}\|\chi_i-\chi_j\|_{TV}$. TV is a number between 0 and 1 with $\delta=1$ the maximum distribution shift and $\delta=0$ the case with no distribution shift.

Problem Definition. We study the performance of learned models when performing the above data operations. Assume an algorithm takes at most $T_I(\boldsymbol{D})$ operations to perform n insertions from a dataset $\boldsymbol{D}$, at most $T_Q(\boldsymbol{D})$ to perform any query and has $S(\boldsymbol{D})$, space overhead (excluding the space to store the data). We study amortized expected insertion time defined as $\frac{1}{n}\mathbb{E}_{\boldsymbol{D}\sim\chi}T_I(\boldsymbol{D})$, expected query time, $\mathbb{E}_{\boldsymbol{D}\sim\chi}T_Q(\boldsymbol{D})$, and storage space, $\mathbb{E}_{\boldsymbol{D}\sim\chi}S(\boldsymbol{D})$.

2.2. Learned Database Operations

Operation Functions. Let $f_{\boldsymbol{D}}(x)$ be an *operation function*, defined as a function that takes an input x and outputs the answer, calculated from the database $\boldsymbol{D}$, for some desired operation. In this paper, $f_{\boldsymbol{D}}$ is either the cardinality function, $c_{\boldsymbol{D}}(x)$, that takes a query, x, as input and outputs the number of points in $\boldsymbol{D}$ that match x, or the rank function, $r_{\boldsymbol{D}}(x)$ that takes a 1-dimensional query as input and returns the number of elements in $\boldsymbol{D}$ smaller than x. The rank function is used in sorting and indexing, because $r_{\boldsymbol{D}}(x)$ is the index of x if $\boldsymbol{D}$ was stored in a sorted array. We use the notation $f_{\boldsymbol{D}}\in\{r_{\boldsymbol{D}},c_{\boldsymbol{D}}\}$ (or $f\in\{r,c\}$) to refer to both functions, $r_{\boldsymbol{D}}$ and $c_{\boldsymbol{D}}$ (for instance, $f_D\geq 0$ is equivalent to the two independent statements that $r_{\boldsymbol{D}}\geq 0$ and $c_{\boldsymbol{D}}\geq 0$).

We also define *distribution operation function*, f_χ, for an operation f, defined as $f_\chi(\boldsymbol{x})=\mathbb{E}_{\boldsymbol{D}\sim\chi}[f_{\boldsymbol{D}}(\boldsymbol{x})]$, if $\boldsymbol{D}$ is sampled from a distribution χ. Note that *distribution operation function* depend only on the data distribution (and not observed dataset). For instance, $\frac{1}{n}r_\chi$ is the c.d.f of data distribution, χ if $\boldsymbol{D}$ is sampled i.i.d from χ, and similarly $\frac{1}{n}c_\chi(\boldsymbol{x})$ is the probability that a sample from χ falls in an axis-parallel rectangle defined by $\boldsymbol{x}$. We call $\mathbb{E}_{\boldsymbol{D}\sim\chi}[c_{\boldsymbol{D}}(\boldsymbol{x})]$ distribution cardinatliy function.

Learned Database Operations with Insertions. Learned database operations learn a model $\hat{f}$ that approximates $f_{\boldsymbol{D}}$ well, and use the learned model to obtain an estimate of the operation output (for sorting and indexing, a refinement step ensures exact result, through either local binary search or lightweight sorting). However, as new data points are inserted and the dataset changes, the ground truth answers to operations change, thereby increasing the model error. Note that model answers are scaled to current data size (i.e., if $\hat{f}$ was trained on a dataset of size i, and tested on a dataset of size j, we report $\frac{j}{i}\hat{f}$ as answers), but this does not stop the error from increasing. Thus, to guarantee the error is below a threshold, one needs to update the models as the datasets change, which is often done by periodically retraining the models. Model retraining contributes to insertion cost in the database. To keep the insertion cost low, one needs to minimize retraining frequency. Meanwhile, infrequent retraining increases error (and, for indexing, query time). Finding a suitable balance between insertion time, accuracy and query time is a subject in much of our theoretical study.

3. Analysis through Distribution Learnability

Our goal is to ensure that a model $\hat{f}$ trained to perform operations f, $f\in\{r,c\}$, has bounded error. We first discuss a lower bound on the error of models in the presence of insertions, which motivates our analysis framework.

Lower Bound on Model Generalization. Consider a model $\hat{f}$, trained after the i-th insertion and using dataset $\boldsymbol{D}^i$. Assume the model is not retrained after k further insertions so that $\hat{f}$ is used to answer queries for dataset $\boldsymbol{D}^j$, $j=i+k$. The following lemma shows a lower bound on the expected maximum *generalization error* of the model to dataset $\boldsymbol{D}^j$, defined as $\sup_{\boldsymbol{x}}\mathbb{E}_{\boldsymbol{D}\sim\chi}[|\frac{j}{i}\hat{f}(\boldsymbol{x})-f_{\boldsymbol{D}^j}(\boldsymbol{x})]$.

Theorem 3.1. *Consider any model $\hat{f}$ trained after the i-th insertion and on dataset $\boldsymbol{D}^i$. For any integer $j>i+2$ and after performing $k=j-i$ new insertions we have*

$$\sup_{\boldsymbol{x}}\mathbb{E}_{\boldsymbol{D}^j\sim\chi}[|\frac{j}{i}\hat{f}(\boldsymbol{x})-f_{\boldsymbol{D}^j}(\boldsymbol{x})|]\geq\frac{\sqrt{k}}{4},$$

when $\boldsymbol{D}^j$ is i.i.d from any continuous distribution χ.

Theorem 3.1 states that the expected error of a *single fixed* model, no matter how good the model is when it is trained, after k insertions, will increase to $\Omega(\sqrt{k})$ on some input. Consequently, to achieve an error at most ϵ, we have to retrain the model at least every $(4\epsilon)^2$ insertions. For any constant error ϵ, this implies $\frac{n}{(4\epsilon)^2}=O(n)$ model retraining is needed when inserting n data points. Model retraining for many practical choices costs $O(n)$ (to go over the data at least once), so that amortized insertion cost, i.e., insertion *cost per insertion*, must be at least $O(n)$. This is significantly larger than non-learned methods, e.g., for indexing B-trees support insertions in $O(\log n)$.

Nonetheless, the $\sqrt{k}$ barrier (and consequently a heavy insertion cost) can be avoided, as is often done in practice, by *partial* retraining. A common example is arranging a set of models in a tree structure and retraining parts of the tree structure as new data is inserted. This avoids a full retraining every $O(\epsilon^2)$ insertions, but makes smaller necessary adjustments throughout that are cheap to make. Thus, Theorem 3.1 provides a theoretical justification for many practical design choices in existing work (Ding et al., 2020; Zeighami et al., 2023; Galakatos et al., 2019) that partition the space and train multiple models, utilizing data structures built around multiple models to perform operations. We also note that such approaches often come with repartitioning and tree balancing as new data is inserted to adjust the

created partitions after observing new points. Indeed such repartitioning is also necessary in the presence of insertions. Given that for a fixed set of partitions the number of points per partition will grow linearly in data size, Theorem 3.1 can be used to show the error per partition will remain large unless partitions are recreated and adjusted as new points are observed.

The error in Theorem 3.1 is independent of total data size, j, and only depends on k. This is because we make no assumptions on model capacity, and consequently, when the model is trained on the dataset, $\boldsymbol{D}^i$ of size i, the training error can be zero. Thus, error on $\boldsymbol{D}^j$ only depends on how well the trained model on $\boldsymbol{D}^i$ generalizes to $\boldsymbol{D}^j$, which, intuitively, only depends on the difference between $\boldsymbol{D}^i$ and $\boldsymbol{D}^j$. Theorem 3.1 quantifies this difference in terms of k.

Analysis Framework Overview. In light of Theorem 3.1 and existing practical modeling choices that use a set of models to perform an operation, analyzing database operations in the presence of insertion can be divided into two components: (1) how well a model can learn a set of observations (*the modeling problem*), and (2) how a set of models can be used to perform operations in the presence of insertions (*the model utilization problem*). Our framework allows studying the two separately, as discussed next.

3.1. The Modeling Problem

Our analysis is divided into studying the problem of modeling and the problem of model utilization. We introduce the notion of *distribution learnability* to abstract away the modeling problem when studying the utilization problem. Roughly speaking, if a distribution class is *distribution learnable*, we can use observations from the class to model their distribution operation functions *well*. In other words, if a distribution class is distribution learnable, we have a solution to the modeling problem, and thus, we can focus on the model utilization problem. Meanwhile, the modeling problem is reduced to showing distribution learnablity. In this section, we define *distribution learnability* and discuss how we can prove a distribution class is distribution learnable.

3.1.1. Defining Distribution Learnability

A distribution class is distribution learnable if there exists an algorithm that returns a good model of the data distribution given an observed dataset. Formally,

Definition 3.2. A distribution class $\mathfrak{X}$, is said to be *distribution learnable for an operation* f, $f \in \{r, c\}$, with parameters $\mathcal{T}_n^{\mathfrak{X}}$, $\mathcal{S}_n^{\mathfrak{X}}$ and $\mathcal{B}_n^{\mathfrak{X}}$, if for any $\chi \subseteq \mathfrak{X}$, there exists an algorithm that takes a set of observations, $\boldsymbol{D} \sim \chi$, of size n as input and returns a model $\hat{f}$ such that:

- (*Accuracy*) If $\boldsymbol{D}$ is sampled from χ, for some $\chi \subseteq \mathfrak{X}$, we have that
$$\mathbb{P}_{D\sim\chi}[\|f_\chi - \hat{f}\|_\infty \geq \epsilon] \leq \varkappa_1 e^{-\varkappa_2(\frac{\epsilon}{\sqrt{n}}-1)^2},$$
For any $\epsilon \geq \sqrt{n}$ and universal constants $\varkappa_1, \varkappa_2 > 0$;
- (*Inference Complexity*) It takes $\mathcal{T}_n^{\mathfrak{X}}$ number of operations to evaluate $\hat{f}$ and space $\mathcal{S}_n^{\mathfrak{X}}$ to store it; and
- (*Training Complexity*) Each call to the algorithm costs $\mathcal{B}_n^{\mathfrak{X}}$ amortized number of operations.

That a distribution class is distribution learnable for operation f means that observations from the distribution class can be used to model the expected value of f to a desired accuracy, and that distribution dependent parameters $\mathcal{T}_n^{\mathfrak{X}}$, $\mathcal{S}_n^{\mathfrak{X}}$ and $\mathcal{B}_n^{\mathfrak{X}}$, characterize the computational complexity of the modeling (amortized number of operations is total number of operations divided by n, so $n\mathcal{B}_n^{\mathfrak{X}}$ is total number of operations). We make two remarks regarding the definition.

Remark 3.3. The accuracy requirement for distribution learnability is defined so that, with high probability, the model error is at most $O(\sqrt{n})$. This is due to Theorem 3.1, which shows the expected generalization error, after n insertions, will be $\Omega(\sqrt{n})$ as dataset changes because of insertions and irrespective of modeling accuracy. Thus, having modeling error lower than $O(\sqrt{n})$ will not improve the generalization error, but will increase inference complexity (larger models will be needed to improve accuracy). Meanwhile, due to the inherent $\Omega(\sqrt{n})$ error, an extra modeling error of $\sqrt{n}$ only increases generalization error by a constant factor, thus not changing any of our results asymptotically.

We also note that, given that Def. 3.2 allows modeling errors to occur, using distribution learnability to return exact results (i.e., for indexing and sorting) requires designing fallback strategies to ensure correctness even in the presence of modeling error. This is done in our results in Sec. 4, where model outputs are adjusted by some lightweight non-learned method (e.g., exponential search for indexing and merge sort for sorting) to ensure correctness. On the other hand, for cardinality estimation, where the goal is to obtain fast estimates and errors can be tolerated in practice, our results in Sec. 4 use the modeling error formulation in Def. 3.2 to present methods that answers queries with guaranteed bounds on error.

Remark 3.4. Distribution learnability for the distribution class, $\mathfrak{X}$ is defined so that we can characterize the computational complexity of modeling data from $\mathfrak{X}$. Such characterization is important because different modeling choices are beneficial for different distributions. For instance, a linear model may be sufficient to model data from uniform distribution but not for a Gaussian distribution. The definition allows us to distinguish between simple distribution classes where we can create models that are fast to evaluate (e.g., linear models for uniform distribution), from more complex distribution classes that may need more complex models with higher runtime and space complexity (e.g., neural networks for complex distributions). This is done through parameters $\mathcal{T}_n^{\mathfrak{X}}$, $\mathcal{S}_n^{\mathfrak{X}}$ and $\mathcal{B}_n^{\mathfrak{X}}$.

Distribution class	$\mathcal{T}_{\mathbf{n}}^{\mathfrak{X}}$	$\mathcal{S}_{\mathbf{n}}^{\mathfrak{X}}$	$\mathbf{n}\mathcal{B}_{\mathbf{n}}^{\mathfrak{X}}$
$\mathfrak{X}_\rho$	1	$\rho\sqrt{n}\log n$	$\rho\sqrt{n}\log n$
$\mathfrak{X}_l$	$\log l$	$l\log n$	$n\log n$
$\mathfrak{X}_c$	1	1	1

Table 2. Asymptotic complexities of some distribution learnable classes for rank function defined in Lemma 3.6

3.1.2. Proving Distribution Learnability

For a distribution class, $\mathfrak{X}$, to be distribution learnable for an operation f, we need to be able to model distribution operation functions in that class using some model class $\mathcal{F}$. Intuitively, $\mathcal{F}$ needs to have *enough representation power* to model distributions in $\mathfrak{X}$, and we need to be able to effectively optimize over $\mathcal{F}$ to find the good representations given an input (i.e., $\mathcal{F}$ is *optimizable*). The following theorem shows that if these two properties are true, then the distribution class is indeed distribution learnable. For the sake of space, we only state our results here informally (formal statement is in Sec. C), as a formal statement requires making *enough representation power* and *opitimizability* concrete, which diverts from our main discussion.

Theorem 3.5 (Informal)**.** *Let $\mathfrak{X}$ be a distribution class whose operation functions belong to some function class $\mathcal{G}$. That is, $f_\chi \in \mathcal{G}$ for all $\chi \in \mathfrak{X}$, for an operation f. Assume another function class, $\mathcal{F}$, has enough representation power to represent $\mathcal{G}$, and is optimizable. Then, $\mathfrak{X}$ is distribution learnable for operation f.*

Theorem 3.5 can be broadly used to translate function approximation results to distribution learnability. For instance, Taylor's theorem shows that infinitely differentiable functions can be approximated by polynomials to arbitrary accuracy (i.e., polynomials have enough representation power to represent infinitely differentiable functions), and the exchange algorithm (Powell, 1981) shows that we can find the best polynomial approximating a function (i.e., shows optimizability for polynomials). These together with Theorem 3.5 imply that distributions with infinitely differentiable operation functions are distribution learnable. Nonetheless, the time complexity of function approximation is important when deciding what function class to choose for modeling purposes in database applications. For instance, the exchange algorithm, although converges, can take too long to find polynomials that model functions with a desired accuracy (Powell, 1981). Our next result uses Theorem 3.5 to show distribution learnability using piecewise linear and piecewise constant models that show better time/space complexity. We first discuss learnability for rank operations.

Lemma 3.6. *Let $\mathfrak{X}_\rho$ be the set of distributions with p.d.f bounded by ρ, $\mathfrak{X}_l$ the set of distributions with piecewise linear c.d.f with at most l pieces and $\mathfrak{X}_c$ a distribution the c.d.f of which can be stored and evaluated in constant time. $\mathfrak{X}_\rho$, $\mathfrak{X}_l$, $\mathfrak{X}_c$ are distribution learnable for rank operation with parameters shown in Table 2.*

Lemma 3.6 presents results for multiple distribution classes. $\mathfrak{X}_\rho$ formulates a realistic scenario (experimentally shown by (Zeighami & Shahabi, 2023)). $\mathfrak{X}_c$ shows the ideal scenario for learned models, where the data distribution is easy to model, and is included to show a best-case scenario for our results when using learned models. Piece-wise linear models have been used for the purpose of indexing (Ferragina & Vinciguerra, 2020; Galakatos et al., 2019), and $\mathfrak{X}_l$ is included to study their theoretical properties for the distribution class where they are well suited. Next, consider distribution learnability for cardinality operation.

Lemma 3.7. *Let $\mathfrak{X}_\rho$ be the set of distributions for which the distribution cardinatliy function has gradient bounded by ρ, and let $\mathfrak{X}_c$ be a countable set of c distributions for which distribution cardinatliy function can be stored and evaluated in constant time. $\mathfrak{X}_\rho$ and $\mathfrak{X}_c$ are distribution learnable for cardinality estimation where the same parameters as Table 2 hold for $\mathfrak{X}_c$. For $\mathfrak{X}_\rho$, we have $\mathcal{B}_n^{\mathfrak{X}}$ and $\mathcal{S}_n^{\mathfrak{X}}$ as $O(\sqrt{2d}(\rho\sqrt{n})^{2d}\log n)$, while $\mathcal{T}_n^{\mathfrak{X}} = O(1)$.*

As before, we have included $\mathfrak{X}_c$ to show a best-case scenario for learned models. Nonetheless, cardinality estimation is a problem in high dimensions where modeling is difficult. The exponential behavior in Lemma 3.7 for $\mathfrak{X}_\rho$ is required for different modeling choices, including neural networks (Petersen & Voigtlaender, 2018; Yarotsky, 2018). To reduce complexity, stricter assumptions on data distribution are often justified. For example, attributes may be correlated and only fall in a small part of the space. A common assumption using neural networks is that data is supported on a low dimensional manifold (Pope et al., 2021), which together with results showing that neural networks can approximate data on low dimensional manifolds well (Chen et al., 2019), yields that neural networks can avoid space complexity exponential in dimensionality. This is an active area of research orthogonal to our work, and our results show how learned database operations can benefit from such approximation theoretic results as they become available.

3.2. The Model Utilization Problem

Our results in Sec. 4 thoroughly discuss how learned models can perform different database operations for distribution learnable classes. Here, we provide a brief overview of the general methodology and state required definitions.

Typical methods used in practice to perform database operations partition the domain and model different parts of the domain separately. Each partition can be denoted by a set R of the space it covers. The model in each partition can be seen as a model of the conditional distribution of the data, where the original data distribution is conditioned on the set R. As such, to effectively model the data in a partition, we need to be able to model the conditional distribution for the partition. This means not only the original data distribution, but also the conditional data distributions need to be distri-

bution learnable. We formalize our notion of conditional distribution to be able to formalize this statement.

Let $\mathcal{R}$ be a set s.t. $\mathcal{R} \subset 2^{\mathcal{D}}$ (i.e., $\mathcal{R}$ is a set of subsets of the data domain). Then, for any $R \in \mathcal{R}$ with $\mathbb{P}_{X\sim\chi}(X \in R) > 0$, we define $\chi|R$ as the data distribution with c.d.f $F_{\chi|R}(\boldsymbol{x}) = \mathbb{P}_{X\sim\chi}(X \leq \boldsymbol{x}|X \in R)$. In this paper, unless otherwise stated, $\mathcal{R}$ is the set of axis-parallel rectangles, where $R = (\boldsymbol{r}_{min}, \boldsymbol{r}_{max})$ with $\boldsymbol{r}_{min}, \boldsymbol{r}_{max} \in [0, 1]^d$ define two corners of the hyper rectangle. We define the *normalized* conditional distribution, $\bar{\chi|R}$, as the distribution with c.d.f $F_{\bar{\chi|R}}(\boldsymbol{x}) = F_{\chi|R}((\boldsymbol{r}_{max} - \boldsymbol{r}_{min})\boldsymbol{x} + \boldsymbol{r}_{min})$. The normalization scales the domain of the conditional distribution back to [0, 1]^d, and helps standardize our modeling discussion. We define the closure of a distribution class $\mathfrak{X}$, denoted by $\bar{\mathfrak{X}}$, as the set $\{\bar{\chi|R}, \forall\chi \in \mathfrak{X}, R \in \mathcal{R}\}$. That is, $\bar{\mathfrak{X}}$ contains not only $\mathfrak{X}$ but all the other distributions obtained by distributions $\chi \in \mathfrak{X}$ conditioned under sets $R \in \mathcal{R}$. Often, we need the distribution class $\bar{\mathfrak{X}}$, and not only $\mathfrak{X}$, to be distribution learnable. $\bar{\mathfrak{X}}$ and $\mathfrak{X}$ can be (but not necessarily are) the same set. An example is the uniform distribution, where conditioning the distribution over any interval yields another uniform distribution over the interval.

4. Results

4.1. Indexing Dynamic Data

We show the following result for dynamic indexing.

Theorem 4.1. *Suppose $\boldsymbol{D} \sim \chi$ for $\chi \subseteq \mathfrak{X}$ for some distribution class $\mathfrak{X}$ with $TV(\chi) \leq \delta$, and that $\bar{\mathfrak{X}}$ is distribution learnable. There exists a learned index into which the n data points of $\boldsymbol{D}$ can be inserted in $O(\mathcal{T}_n^{\mathfrak{X}} \log\log n + \log\delta\sqrt{n} + \mathcal{B}_n^{\mathfrak{X}} \log^2 \log n)$ expected amortized time, that can be queried in $O(\mathcal{T}_n^{\mathfrak{X}} \log\log n + \log\delta\sqrt{n})$ expected time and takes space $O(n \log n + \sum_{i=0}^{\log\log n} n^{1-2^{-i}} \mathcal{S}_{n^{2^{-i}}}^{\mathfrak{X}})$.*

The term $\mathcal{T}_n^{\mathfrak{X}} \log\log n$ is due to making $\log\log n$ calls to the distribution model, and $\mathcal{B}_n^{\mathfrak{X}}$ roughly refers to the need to rebuild a model every n insertions. For example, without distribution shift (i.e., $\delta = 0$), one can answer queries and perform insertions with $O(\log\log n)$ model calls, while every n insertions incurs extra $\mathcal{B}_n^{\mathfrak{X}}$ cost for model rebuilding.

Distribution shift increases both insertion and query time by $O(\log\delta\sqrt{n})$. In the worst case, having $\delta = 1$, we recover the traditional $O(\log n)$ insertion and query time. That is, our results show no gain from modeling when distribution shift is too severe. This is as expected. If data distribution changes too much, one cannot use the current knowledge of data distribution to locate future elements. By systematically handling the distribution shift, we show that a learned method can provide robustness in such scenarios.

The data structure that achieves the bound is a tree structure with a distribution model used in each node to find the node's child to traverse given a query or insertion. The structure can be thought of as a special case of Alex (Ding et al., 2020), with specific tree height, fanout and split mechanism to ensure the desired gaurantees. All elements are stored at leaf nodes, and the traversal to the leaf nodes is similar to B-trees but using learned models to choose the child. Using Lemma 3.6 we can specialize Theorem 4.1 for specific distribution classes.

Corollary 4.2. *Let $\mathfrak{X}_{\rho_1,\rho_2}$ be the class of distributions with bounded p.d.f. That is, for all $\chi \in \mathfrak{X}_{\rho_1,\rho_2}$ and denoting by g_χ the p.d.f of χ, we have $0 < \rho_1 \leq g_\chi(x) \leq \rho_2 < \infty$, $\forall x$. Suppose $\boldsymbol{D} \sim \chi$ for $\chi \subseteq \mathfrak{X}$ for some distribution class $\mathfrak{X} \subseteq \mathfrak{X}_{\rho_1,\rho_2}$ with $TV(\chi) \leq \delta$. There exists a learned index that supports insertions in $O(\log\log n + \log\delta\sqrt{n})$ expected amortized time, queries in $O(\log\log n + \log\delta\sqrt{n})$ expected time and takes space $O(\frac{\rho_1}{\rho_2} n \log n)$.*

Corollary 4.2 shows a learned index that performs insertions and answers queries in $O(\log\log n + \log(\delta n))$, while non-learned methods take $O(\log n)$. Thus, when distribution shift is not severe, a learned method can outperform non-learned methods, while large distribution shift ($\delta = 1$) leads to same bounds as non-learned methods. Corollary 4.1 strictly generalizes results in (Zeighami & Shahabi, 2023) to the setting with insertions and data distribution change.

We note that one can consider the data structure used in the proof of Theorem 4.1 (and consequently Corollary 4.2) as a method for switching between learned and non-learned indexes when the distribution shift grows. Recall that proof of Theorem 4.1 uses a tree of learned models as an index. When there is no distribution shift, this tree is traversed only using learned models. However, when the distribution shift is large, the tree structure needs to be adjusted frequently (i.e., nodes are split frequently), and these adjustments are stored in a non-learned data structure. Thus, when the distribution shift is large, the tree traversal becomes more reliant on non-learned data structures.

4.2. Cardinality Estimation

For cardinality estimation, designing learned models that answer queries with arbitrary accuracy is more challenging due to the high dimensionality of the problem. The curse of dimensionality is a well-understood phenomenon for non-learned methods, leading to approaches that take space exponential in dimensionality (Cormode et al., 2012; Wei & Yi, 2018). We first show that this is not the case when using learned models if an error of $\Omega(\sqrt{n})$ is tolerable.

Theorem 4.3. *Suppose $\boldsymbol{D} \sim \chi$ for $\chi \subseteq \mathfrak{X}$ for a distribution learnable class $\mathfrak{X}$ with $TV(\chi) \leq \delta$. There exists a learned cardinality estimator that answers queries with expected error ϵ for $\epsilon = \Omega(\sqrt{n})$ supports insertions in $O(\max\{\frac{\delta\sqrt{n}}{\epsilon}, 1\}\mathcal{B}_n^{\mathfrak{X}})$, queries in $O(\mathcal{T}_n^{\mathfrak{X}})$ and takes space $O(\mathcal{S}_n^{\mathfrak{X}})$.*

Theorem 4.3 states that we can use a distribution model to answer queries for any expected error $\Omega(\sqrt{n})$. Consequently, when we can effectively model a data distribu-

tion, we can answer queries to accuracy at least $\sqrt{n}$ without having an exponential space blowup. Comparing this with random sampling, and more broadly ϵ-approximations, that need at least $\sqrt{n}\log^{d-1}(\sqrt{n})$ data samples to answer queries with accuracy $\sqrt{n}$ (Wei & Yi, 2018; Matoušek & Nikolov, 2015), we show a clear advantage to learned models over such non-learned methods in this accuracy regime.

Theorem 4.3 uses a single distribution model that is periodically retrained with insertion. The frequency of retraining depends on distribution shift. If $\delta \leq \frac{1}{\sqrt{n}}$, the error caused by distribution shift is on a similar scale as error due to randomness. Thus, the distribution shift does not significantly affect insertion time. On the other hand, in the worst case when $\delta = 1$, we need to retrain the model every $\frac{1}{\epsilon}$ insertions, which can be significant depending on retraining cost.

Error of $\Omega(\sqrt{n})$ is not necessarily too large. Indeed, expected query answer for a fixed query with probability p is $n \times p$, so error, relative to the expected query answer is $O(\frac{\sqrt{n}}{np}) = O(\frac{1}{\sqrt{n}})$ and goes to zero as data size increases. Nonetheless, one may wish to answer queries more accurately. Below, we discuss how to achieve this in one dimension. Appendix D.6.3, presents Lemma D.1 that shows how ideas in one dimension can be extended to high dimensions, but nevertheless, only achieves space complexity exponential in dimensionality, similar to non-learned methods.

Arbitrary Accuracy in One Dimension. In one dimension, we show the following is possible using learned models.

Theorem 4.4. *Suppose $\boldsymbol{D} \sim \chi$ for $\chi \subseteq \mathfrak{X}$ for a distribution class $\mathfrak{X}$ with $TV(\bar{\chi}) \leq \delta$, and that $\mathfrak{X}$ is distribution learnable. There exists a learned cardinality estimator that answers queries with expected error ϵ for any $\epsilon > 0$ supports insertions in $O(\max\{delta\epsilon, \epsilon^2\}\mathcal{B}^{\mathfrak{X}}_{\epsilon^2}) + \log n)$, queries in $O(\log n + \mathcal{T}^{\mathfrak{X}}_{\epsilon^2})$ and takes space $O(\frac{n}{\epsilon^2}\mathcal{S}^{\mathfrak{X}}_{\epsilon^2} + \frac{n}{\epsilon^2}\log n)$.*

Theorem 4.4 shows that we can effectively answer queries to any accuracy in one dimension using learned models. Importantly, the result shows that if the data distribution can be modeled space-efficiently (e.g., whenever $\mathcal{S}^{\mathfrak{X}}_{\epsilon^2} \leq \log n$), then a learned approach outperforms non-learned (and worst-case optimal) method discussed in (Wei & Yi, 2018) that takes space $O(\frac{n}{\epsilon}\log n)$ to answer queries with accuracy ϵ.

The learned model that achieves the bound in Theorem 4.4 uses a combination of materialized answers and model estimates to answer queries. Given that a model can be at best accurate to $\sqrt{n}$ if the dataset contains n points, the algorithm divides up the data domain into $\frac{n}{\epsilon^2}$ intervals, each containing ϵ^2 points, so that a model for each interval will have accuracy ϵ. Meanwhile, the algorithm materializes query answers that span multiple intervals so that errors do not accumulate when answering such queries. The materialization is done through a B-tree like structure, where each node stores the exact number of points inserted into it. Because, in our construction, we build several models each for a subset of the space, it is not enough that the total variation between the distributions is bounded, but also that the total variation after conditioning is bounded ($\bar{\chi}$ is the closure of χ under conditioning defined in Sec. 3.2).

4.3. Sorting

Sorting involves only a fixed array, while the operations we discussed so far consider a dataset that changes due to insertions. Our discussion here shows that the distribution learnability framework can be beneficial for analyzing learned database operations beyond insertions.

To see why our framework applies to learned sorting, first recall the existing learned sorting algorithm of Kristo et al. (2020), which sorts an array by first sampling a subset of the array, learning a model to predict the correct location using the sample (the sample is sorted by an existing algorithm for the purpose of training), and then using the learned model to predict the item locations in the original array. Here, similar to the case of learned operations with insertions, the problem isn't (only) how well we can learn a model, but also how well a model learned from a sample of the array will generalize to the complete array. Thus, we need to both study a modeling problem and a model utilization problem, and we can do so using distribution learnability. Finally, since a function that sorts an array is the rank function, our discussion on distribution learnability for the rank operation already covers the modeling problem.

Before stating our results, we also note that one can sort a fixed array by iteratively inserting its element into a learned index. Thus, Theorem 4.1 has already provided a method for sorting an array using machine learning. Our result below presents another learned method for sorting an array. This is analogous to how both B-trees and merge sort can be used to sort a fixed array. The result below can be seen as a means of extending merge sort with machine learning.

Theorem 4.5. *Suppose an array consists of n points sampled i.i.d. from a distribution learnable class $\mathfrak{X}$. There exists a learned sorting method that sorts the array in $O(\mathcal{T}^{\mathfrak{X}}_{\sqrt{n}} n \log\log n + \sqrt{n}\mathcal{B}^{\mathfrak{X}}_{\sqrt{n}} + \sum_{i=0}^{\log\log n} n^{1-\frac{1}{2}(\frac{4}{5})^i}\mathcal{B}^{\mathfrak{X}}_{n^{\frac{1}{2}(\frac{4}{5})^i}})$ and space $O(\mathcal{S}^{\mathfrak{X}}_{\sqrt{n}} + n\log n)$*

We note that $\sqrt{n}\mathcal{B}^{\mathfrak{X}}_{\sqrt{n}} + \sum_{i=0}^{\log\log n} n^{1-\frac{1}{2}(\frac{4}{5})^i}\mathcal{B}^{\mathfrak{X}}_{n^{\frac{1}{2}(\frac{4}{5})^i}})$ is $O(n)$ if model training takes linear in data size, so that the runtime is $O(\mathcal{T}^{\mathfrak{X}}_{\sqrt{n}} n \log\log n)$. This is independent of training time, $\mathcal{B}^{\mathfrak{X}}_n$, because, due to sampling, training is done on much smaller arrays than the original data. Thus, Theorem 4.5 provides a time complexity for sorting similar to Theorem 4.1, both showing that for efficient modeling choices, one can sort an array in $O(n\log\log n)$ model calls.

The algorithm that achieves the bound in Theorem 4.5 is similar to Kristo et al. (2020), which first samples a subset of the original array, uses it to build a distribution model,

and uses the model to sort the original array. Due to modeling errors, the resulting attempt using the model will only be a partially sorted array. Unlike Kristo et al. (2020) that uses insertion sort to fully sort the partially sorted array, we use a merge sort like approach to recursively sort the array. This is because, to reduce the asymptotic complexity below $O(n \log n)$, the sample needs to be of size $o(n)$. However, the generalization error of a model trained on a sample of size $o(n)$ would be too large to allow insertion sort to be effective. Partitioning the partially sorted array and recursively sorting each portion allows us to sort the array while performing $O(\mathcal{T}^{\mathfrak{X}}_{\sqrt{n}} n \log \log n)$ operations.

Finally, the lower bound of Theorem 3.1 does not apply to sorting, and a natural question is if it is possible to do better than $O(n \log \log n)$. The following result shows that under stronger assumptions on data distribution, this is possible.

Theorem 4.6. *Suppose an array consists of n points sampled i.i.d. from a distribution χ, and assume we have a model $\hat{r}$ s.t., $\|\hat{r} - r_\chi\|_\infty \leq \epsilon_0$, that can be evaluated in $\mathcal{T}^\chi$ and takes space $\mathcal{S}^\chi$. There exists an algorithm that sorts the array in $O(\mathcal{T}^\chi n \log \epsilon_0)$ taking space $O(\mathcal{S}^\chi + n)$.*

The theorem shows if we know the data distribution very accurately, then we can sort the data very efficiently. $\mathcal{T}^\chi$ can be $O(1)$, e.g., if the data c.d.f was a polynomial, so that we can sort an array in $O(n)$. This is because the data distribution provides a good indicator of the location of the item in the sorted array. The algorithm that achieves this can be seen as a special case of merge sort, where instead of dividing the array into 2, we divide the array, using $\hat{r}$, to up to n groups, and recursively sort each.

The difference between Theorems 4.6 and 4.5 is how accurate of a model of data distribution we have access to. Theorem 4.5 effectively assumes that data distribution can only be modeled to accuracy $O(\sqrt{n})$, which is too large to allow fixing model errors in sorting with a single pass over the array. On the other hand, Theorem 4.6 assumes the model is correct to within a constant accuracy. As a result, a single iteration over the partially sorted array fixes any potential inversions and yields the $O(n)$ complexity. Nonetheless, knowing the data distribution to a constant accuracy can be impractical, because it requires further knowledge about the data distribution beyond merely observing its samples.

5. Conclusion and Future Work

We have presented a thorough theoretical analysis of learned indexing and cardinality estimation in the presence of insertions from a possibly changing data distribution. Our results characterize learned models' performance, and show when and why they can outperform their non-learned counterparts. We have developed the distribution learnability analysis framework that provides a systematic tool for analyzing learned database operations. Our results enhance our understanding of learned database operations and provide the much-needed theoretical guarantees on their performance for robust practical deployment.

We believe our theoretical tools will pave the way for a broader theoretical understanding of various learned methods. Nonetheless, there are several aspects that require further research to allow for a broader applicability of our results to real-world databases. First, we have established distribution learnability for distributions with bounded p.d.f and piecewise-linear c.d.f, but demonstrating distribution learnability for a broader range of real-world data distributions is needed to cover a more comprehensive set of real-world data distributions. There are two aspects that require further research. There is a need to identify and formally characterize the distribution classes from which real-world datasets originate. This task is challenging, especially for high-dimensional data. For instance, in the case of images, it is commonly believed that they lie on a low-dimensional manifold. It is essential to validate if such assumptions also hold true for real-world tabular datasets. Moreover, we need to establish distribution learnability for such distribution classes using appropriate modeling choices. The best modeling choice may vary depending on the distribution class, necessitating further research to determine the most effective modeling choices for specific distribution classes. Other future work includes incorporating deletions and updates, where we believe statistical tools developed here can be utilized, but formalizing the notion of data distribution in the presence of deletions/updates, and the relationship between insertions and deletions/updates require further research. Finally, considering query distribution, and analyzing other database operations are other future directions.

Acknowledgements

This research has been funded by NSF grant IIS-2128661 and NIH grant 5R01LM014026. Opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any sponsors, such as NSF and NIH.

Impact Statement

This paper presents work whose goal is to theoretically understand the use of Machine Learning in database systems. Developing such theoretical results can improve trust and usability of ML in database systems, leading to a more widespread adoption in practice. This can help reduce cost, lower energy consumption, and improve usability for database systems. There can be various societal consequences for this, including the creation of more data-driven applications.

References

Agarwala, A., Das, A., Juba, B., Panigrahy, R., Sharan, V., Wang, X., and Zhang, Q. One network fits all? modular versus monolithic task formulations in neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uz5uw6gM0m.

Andersson, A. and Mattsson, C. Dynamic interpolation search in o (log log n) time. In *Automata, Languages and Programming: 20th International Colloquium, ICALP 93 Lund, Sweden, July 5–9, 1993 Proceedings 20*, pp. 15–27. Springer, 1993.

Berend, D. and Kontorovich, A. A sharp estimate of the binomial mean absolute deviation with applications. *Statistics & Probability Letters*, 83(4):1254–1259, 2013.

Chen, M., Jiang, H., Liao, W., and Zhao, T. Efficient approximation of deep relu networks for functions on low dimensional manifolds. *Advances in neural information processing systems*, 32, 2019.

Cormode, G., Garofalakis, M., Haas, P. J., and Jermaine, C. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends Databases*, 4(1–3): 1–294, January 2012. ISSN 1931-7883. doi: 10.1561/1900000004. URL https://doi.org/10.1561/1900000004.

Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., Zhang, H., Chandramouli, B., Gehrke, J., Kossmann, D., et al. Alex: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 969–984, 2020.

Ferragina, P. and Vinciguerra, G. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment*, 13 (8):1162–1175, 2020.

Ferragina, P., Lillo, F., and Vinciguerra, G. Why are learned indexes so effective? In *International Conference on Machine Learning*, pp. 3123–3132. PMLR, 2020.

Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R., and Kraska, T. Fiting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data*, pp. 1189–1206, 2019.

Hilprecht, B., Schmidt, A., Kulessa, M., Molina, A., Kersting, K., and Binnig, C. Deepdb: Learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13 (7), 2019.

Hu, X., Liu, Y., Xiu, H., Agarwal, P. K., Panigrahi, D., Roy, S., and Yang, J. Selectivity functions of range queries are learnable. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pp. 959–972, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517896. URL https://doi.org/10.1145/3514221.3517896.

Janson, S. On the probability that a binomial variable is at most its expectation. *Statistics & Probability Letters*, 171:109020, 2021.

Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P., and Kemper, A. Learned cardinalities: Estimating correlated joins with deep learning. *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research*, 2018.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 489–504, 2018.

Kristo, A., Vaidya, K., Çetintemel, U., Misra, S., and Kraska, T. The case for a learned sorting algorithm. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1001–1016, 2020.

Lu, Y., Kandula, S., König, A. C., and Chaudhuri, S. Pre-training summarization models of structured datasets for cardinality estimation. *Proceedings of the VLDB Endowment*, 15(3):414–426, 2021.

Ma, Q. and Triantafillou, P. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*, pp. 1553–1570, 2019.

Matoušek, J. and Nikolov, A. Combinatorial discrepancy for boxes via the gamma_2 norm. In *31st International Symposium on Computational Geometry (SoCG 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Mehlhorn, K. and Tsakalidis, A. Dynamic interpolation search. *Journal of the ACM (JACM)*, 40(3):621–634, 1993.

Mustafa, N. H. and Varadarajan, K. R. Epsilon-approximations and epsilon-nets. *Chapter 47 in Handbook of Discrete and Computational Geometry, 3rd edition*, 2017.

Negi, P., Marcus, R., Kipf, A., Mao, H., Tatbul, N., Kraska, T., and Alizadeh, M. Flow-loss: Learning cardinality estimates that matter. *Proc. VLDB Endow.*, 14 (11):2019–2032, jul 2021. ISSN 2150-8097. doi: 10.14778/3476249.3476259. URL https://doi.org/10.14778/3476249.3476259.

Negi, P., Wu, Z., Kipf, A., Tatbul, N., Marcus, R., Madden, S., Kraska, T., and Alizadeh, M. Robust query driven cardinality estimation under changing workloads. *Proceedings of the VLDB Endowment*, 16(6):1520–1533, 2023.

O'Rourke, J. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 24 (9):574–578, 1981.

Perl, Y. and Reingold, E. M. Understanding the complexity of interpolation search. *Information Processing Letters*, 6(6):219–222, 1977.

Perl, Y., Itai, A., and Avni, H. Interpolation search—a log log n search. *Communications of the ACM*, 21(7): 550–553, 1978.

Petersen, P. and Voigtlaender, F. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

Peterson, W. W. Addressing for random-access storage. *IBM journal of Research and Development*, 1(2):130–146, 1957.

Pope, P., Zhu, C., Abdelkader, A., Goldblum, M., and Goldstein, T. The intrinsic dimension of images and its impact on learning. *arXiv preprint arXiv:2104.08894*, 2021.

Powell, M. J. D. *Approximation theory and methods*. Cambridge university press, 1981.

Shorack, G. R. and Wellner, J. A. *Empirical processes with applications to statistics*. SIAM, 1986.

Wang, X., Qu, C., Wu, W., Wang, J., and Zhou, Q. Are we ready for learned cardinality estimation? *Proceedings of the VLDB Endowment*, 14(9):1640–1654, 2021.

Wasan, M. T. Parametric estimation. *McGraw-Hill;*, 1970.

Wei, Z. and Yi, K. Tight space bounds for two-dimensional approximate range counting. *ACM Transactions on Algorithms (TALG)*, 14(2):1–17, 2018.

Wongkham, C., Lu, B., Liu, C., Zhong, Z., Lo, E., and Wang, T. Are updatable learned indexes ready? *Proceedings of the VLDB Endowment*, 15(11):3004–3017, 2022.

Wu, P. and Cong, G. A unified deep model of learning from both data and queries for cardinality estimation. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 2009–2022, 2021.

Yang, Z., Liang, E., Kamsetty, A., Wu, C., Duan, Y., Chen, X., Abbeel, P., Hellerstein, J. M., Krishnan, S., and Stoica, I. Deep unsupervised cardinality estimation. *Proceedings of the VLDB Endowment*, 13(3):279–292, 2019.

Yang, Z., Kamsetty, A., Luan, S., Liang, E., Duan, Y., Chen, X., and Stoica, I. Neurocard: one cardinality estimator for all tables. *Proceedings of the VLDB Endowment*, 14 (1):61–73, 2020.

Yao, A. C. and Yao, F. F. The complexity of searching an ordered random table. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pp. 173–177. IEEE Computer Society, 1976.

Yarotsky, D. Optimal approximation of continuous functions by very deep relu networks. In *Conference on learning theory*, pp. 639–649. PMLR, 2018.

Zeighami, S. and Shahabi, C. On distribution dependent sub-logarithmic query time of learned indexing. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 40669–40680. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/zeighami23a.html.

Zeighami, S. and Shahabi, C. Towards establishing guaranteed error for learned database operations. In *The Twelfth International Conference on Learning Representations*, 2024.

Zeighami, S., Shahabi, C., and Sharan, V. Neurosketch: Fast and approximate evaluation of range aggregate queries with neural networks. *Proceedings of the ACM on Management of Data*, 1(1):1–26, 2023.

A. Related Work

A large and growing body of work has focused on using machine learning to speed up database operations, among them, learned indexing (Galakatos et al., 2019; Kraska et al., 2018; Ferragina & Vinciguerra, 2020; Ding et al., 2020), learned cardinality estimation (Kipf et al., 2018; Wu & Cong, 2021; Hu et al., 2022; Yang et al., 2019; 2020; Lu et al., 2021; Negi et al., 2021) and learned sorting (Kristo et al., 2020). Most existing work focus on improving modeling choices, with various modeling choices such as neural networks (Zeighami et al., 2023; Kipf et al., 2018; Kraska et al., 2018), piece-wise linear approximation (Ferragina & Vinciguerra, 2020), sum-product networks (Hilprecht et al., 2019) and density estimators (Ma & Triantafillou, 2019). Existing results show significant empirical benefits in static datasets, while performance often deteriorates in dynamic datasets and in the presence of distribution shift (Wang et al., 2021; Wongkham et al., 2022). Our theoretical results help explain such observations and provide a theoretical framework for analysis of the operations under different modeling choices.

On the theory side, no existing study meaningfully characterizes performance of learned models in the dynamic setting or studies learned sorting. In the static setting, (Zeighami & Shahabi, 2023; Ferragina et al., 2020) study query time of learned indexing. Ferragina et al. (2020) shows learned models can provide constant factor improvements under an assumption on the distribution of the gap between observations, and Zeighami & Shahabi (2023) shows a learned model can answer queries in $O(\log\log n)$ query time if the p.d.f of data distribution is non-zero and bounded. Our results strictly generalize the latter to the dynamic setting, in the presence of insertions from a possibly changing distribution, and also show that more generally, $O(\mathcal{T}_n^{\mathfrak{X}}\log\log n+\log\delta n)$ query time is possible for any distribution learnable class $\mathfrak{X}$. Moreover, Zeighami et al. (2023) presents a special case of our Theoerem 4.3 for cardinality estimation on static datasets for distributions where the operation distribution function is Lipschitz continuous. Our result strictly generalizes Zeighami et al. (2023) to the dynamic setting with distribution change and any distribution learnable class $\mathfrak{X}$. Orthogonal to our work, Zeighami & Shahabi (2024) study lower bounds on the model size needed to perform various database operations with a desired accuracy and (Hu et al., 2022; Agarwala et al., 2021) study the number of training samples needed to achieve a desired accuracy for different database operations.

Finally, we draw a broader connection between our work, learned indexing and interpolation search. A large body of early work focused on interpolation search Peterson (1957); Perl et al. (1978); Perl & Reingold (1977); Yao & Yao (1976); Mehlhorn & Tsakalidis (1993), proposed by (Peterson, 1957) which uses linear interpolation to estimate the location of a query in an array. It has been shown that this search algorithm achieves $O(\log\log n)$ query time on uniformly distributed arrays (Yao & Yao, 1976; Perl & Reingold, 1977), with extensions to cover *smooth* distribution classes and dynamic data in (Andersson & Mattsson, 1993; Mehlhorn & Tsakalidis, 1993). Indeed, interpolation search can be seen as an early example of a model-based search, where linear models are used to estimate item locations. Using the terminology introduced in this paper and given that uniform distribution is distribution learnable using linear models (c.d.f of the uniform distribution is a linear function), the $O(\log\log n)$ query time can be seen as a special case of our results. Overall, interpolation search can be seen as a special case of learned indexing, where learned indexing allows for more complex data-driven modeling choices that can be useful for a broader class of data distributions.

B. Formalized Setup and Operations

We are interested in performing database operations on a possibly changing dataset. We assume data records are d-dimensional points in the range $[0,1]$ (otherwise, the data domain can be scaled and shifted to this range). We either consider the setting when n data points are inserted one by one into the dataset, or that we are given a fixed set of n data points. We refer to the former as the dynamic setting and the latter as the static setting. We define $\boldsymbol{D}^i$ as the dataset $\boldsymbol{D}^i\in[0,1]^{i\times d}$, i.e., a dataset consisting of i records inserted so far and in d dimensions with each attribute in the range [0, 1], where i and d are integers greater than or equal to 1. $\boldsymbol{D}^n$ is the dataset after the last insertion, and is often denoted as $\boldsymbol{D}$. $\boldsymbol{D}^{i:j}$ denotes the dataset of points inserted after the i-th insertion until the j-th (i.e., $\boldsymbol{D}^j\setminus\boldsymbol{D}^i$). We use $\boldsymbol{D_i}$ to refer to the i-th record of a dataset (which is a d-dimensional vector) and $D_{i,j}$ to refer to the j-th element of $\boldsymbol{D_i}$. If $d=1$ (i.e., $\boldsymbol{D}$ is 1-dimensional), then D_i is the i-th element of $\boldsymbol{D}$ (and is not a vector). We study the following database operations.

Indexing. The goal is to use an index to store and find items in a 1-dimensional dataset. The index supports insertions and queries. n items are inserted into the index one by one. After inserting k items, for any $1\leq k\leq n$, we would like to retrieve items from the dataset based on a query $q\in[0,1]$. The query is either an exact match query or a range query. An exact match query returns the point in the database that exactly matches the query q (or `NULL` if there is none) while a range query $[q,q']$ returns all the elements in the dataset that fall in the range $[q,q']$, for $q,q'\in[0,1]$.

Cardinality Estimation. Used often for query optimization, the goal is to find how many records in the dataset match a range query, where the query specifies lower and upper bound conditions on the values of each attribute. Specifically, the query predicate $\boldsymbol{q}=(c_1,...,c_d,r_1,...,r_d)$, specifics the condition that the i-th attribute is in the interval $[c_i,c_i+r_i]$, for $c_i,r_i\in[0,1]$. Data records can be inserted into the data set one by one. After the insertion of k-th item, for any $1\leq k\leq n$, we would like to obtain an estimate of the cardinality of query $\boldsymbol{q}$. We expect that the answers are within error ϵ of the true answers. That is, if $c(\boldsymbol{q})$ is the true cardinality of q and $\hat{c}(\boldsymbol{q})$ is an estimate, we expect $|c(\boldsymbol{q})-\hat{c}(\boldsymbol{q})|\leq\epsilon$. This guarantee has to hold throughout, and as new elements are inserted in the dataset.

Sorting. The goal is to sort a fixed array of size n. That is, we are given a one-dimensional array, $\boldsymbol{D}$, and the goal is to return an array, $\boldsymbol{D}'$, which has the same elements as $\boldsymbol{D}$ but ordered so that $D'_i\leq D'_{i+1}$. Unlike indexing and cardinality estimation, sorting assumes a fixed given array that needs to be sorted. Although indexing can often be used to sort an array (e.g., inserting elements one by one into a binary tree sorts a fixed array), we study the problem of sorting more broadly and explore other learned solutions to the problem beyond indexing (e.g., analogous to how merge sort can also be used to sort an array).

C. Distribution Learnability Through Function Approximation

We first formalize representation power and optimizablity, and then present a formal statement for Theorem 3.5.

Representation Power. Consider using a function class $\mathcal{F}$ to approximate another function class $\mathcal{G}$ (e.g., neural networks to approximate real-valued functions). Consider some hyperparameter, ϑ, that controls the representation power and inference complexity in $\mathcal{F}$, and denote by $\mathcal{F}_\vartheta$ is the subset of $\mathcal{F}$ with hyperparameter ϑ. For instance, ϑ can be the number of learnable parameters of a neural network, the maximum degree of a polynomial, or the number of pieces in a piecewise approximation. In all such cases, larger ϑ implies better representation power but also higher inference time and/or space complexity. Assume we have access to a *representation complexity* function $\alpha_{\mathcal{F}\to\mathcal{G}}(\epsilon)$, that given a maximum error ϵ returns the smallest value of ϑ such that for any $g \in \mathcal{G}$ there exists an $f \in \mathcal{F}_\vartheta$ with $\|f - g\|_\infty$. The function $\alpha_{\mathcal{F}\to\mathcal{G}}(\epsilon)$ determines the required model complexity of $\mathcal{F}$, in terms of ϑ, to represent all elements of $\mathcal{G}$ with error at most ϵ. For instance, such a function for neural networks approximating real-valued functions will show the minimum number of neural network parameters needed to approximate all real-valued functions to error at most ϵ with a neural network. We say that a function class, $\mathcal{F}$ has the *representation power* to model $\mathcal{G}$ if there exists a representation complexity function $\alpha_{\mathcal{F}\to\mathcal{G}}(\epsilon)$ for all $\epsilon > 0$. Finally, let $\tau_{\mathcal{F}\to\mathcal{G}}(\epsilon)$ and $\sigma_{\mathcal{F}\to\mathcal{G}}(\epsilon)$ respectively be the maximum time and space complexity of performing a model forward pass for functions in $\mathcal{F}_\vartheta$ for $\vartheta = \alpha_{\mathcal{F}\to\mathcal{G}}(\epsilon)$.

Optimizability. We say a function class $\mathcal{F}$ is optimizable with an algorithm $\mathcal{A}$ if given any function h and a hyperparameter value ϑ, $\mathcal{A}(h, \vartheta)$ returns an approximately optimal representation of h in $\mathcal{F}_\theta$. Formally, for $\hat{h} = \mathcal{A}(h, \vartheta)$ and if $h^* = \arg\min_{\hat{h}\in\mathcal{F}_\theta} \|h - \hat{h}\|$, $\|h^* - h\|_\infty \geq \varkappa \|\hat{h} - h\|_\infty$ for a constant $\varkappa \leq 1$. Let $\beta(\vartheta)$ be the maximum time complexity of $\mathcal{A}$.

We note that although optimizability as defined broadly above is sufficient to show distribution learnability, it is not necessary. Here, we discuss two qualifications to the definition that make proving optimizability simpler, specifically for database operations. First, it is only necessary to have optimizability for $h \in f_{\boldsymbol{D}}$ for all possible $\boldsymbol{D}$ and for a desired operation function f (since we will only use $\mathcal{A}$ to model operation functions). This can simplify the optimizability requirement depending on the operation function considered. For example, when showing opimizability for rank operations, we only need an $\mathcal{A}$ that returns approximately optimal estimates for input functions that are non-decreasing (since all rank functions are non-decreasing). Second, when $\mathcal{A}$ is used on $h = f_{\boldsymbol{D}^n}$, we can allow additive error of $O(\frac{1}{\sqrt{n}})$. That is, we only need to show $\frac{1}{\varkappa}\|h^* - h\|_\infty + \frac{\varkappa'}{\sqrt{n}} \geq \|\hat{h} - h\|_\infty$ for $\varkappa' \geq 0$ and $\varkappa \leq 1$ universal constants.

Theorem C.1. *Assume a function class, $\mathcal{F}$, is optimizable with an algorithm $\mathcal{A}$, and that $\mathcal{F}$ has enough representation power to represent $\mathcal{G}$. Let $\mathfrak{X}$ be a distribution class with $f_\chi \in \mathcal{G}$ for all $\chi \in \mathfrak{X}$. Then, $\mathfrak{X}$ is distribution learnable with $\mathcal{T}_n^{\mathfrak{X}} = \tau_{\mathcal{F}\to\mathcal{G}}(\frac{1}{\sqrt{n}})$, $\mathcal{S}_n^{\mathfrak{X}} = \sigma_{\mathcal{F}\to\mathcal{G}}(\frac{1}{\sqrt{n}})$, and $\mathcal{B}_n^{\mathfrak{X}} = \beta(\alpha_{\mathcal{F}\to\mathcal{G}}(\frac{1}{\sqrt{n}}))$.*

D. Proofs

The high-level idea behind most of our theoretical results is to use the relationship between query answers and distribution properties. Overall, many statistical tools have been developed that relate the properties of an observed dataset to the data distribution (e.g., studying the relationship between sample mean and distribution mean). In statistics, such tools have been used to describe the population using observed samples. Our proofs often use such tools to do the opposite, that is, use the properties of the data distribution to describe observed samples. Indeed, that is the intuition behind learned database operations, that if the data distribution can be efficiently modeled, then it can be used to answer queries about the observed samples (i.e., the database) efficiently. Our proposed distribution learnability framework allows us to state this more formally. It allows us to assume that we can indeed model the data distribution efficiently. Then, the analysis can focus on utilizing statistical tools to characterize the relationship between the observed sample and the data distribution. Having access to an accurate model of the data distribution, we use existing statistical tools to analyze its error. However, a main challenge in the case of learned database operations is to balance accuracy and efficiency. Thus, our theoretical study includes designing data structures and algorithms that can utilize modeling capacities while performing operations as efficiently as possible.

D.1. Proof of Theorem 3.1

We would like to bound $\mathbb{E}_{\boldsymbol{D}\sim\chi}[\|\hat{f} - f_{\boldsymbol{D}^j}\|]$. Note that both $\hat{f}$ and $f_{\boldsymbol{D}^j}$ are random variable (since $\hat{f}$ depends on $f_{\boldsymbol{D}^i}$). First, consider

$$f_{\boldsymbol{D}^j}(q) = \frac{1}{j}\sum_{k\in[j]} I_{\boldsymbol{D}_k \in q} = \frac{i}{j} f_{\boldsymbol{D}^i}(q) + \frac{j-i}{j} f_{\boldsymbol{D}^{i:j}}(q),$$

So that the error is

$$\mathbb{E}_{\boldsymbol{D}\sim\chi}[\|\hat{f} - \frac{i}{j} f_{\boldsymbol{D}^i}(q) + \frac{j-i}{j} f_{\boldsymbol{D}^{i:j}}(q)\|]$$
$$= \mathbb{E}_{\boldsymbol{D}^i\sim\chi}[\mathbb{E}_{\boldsymbol{D}^{i:j}\sim\chi}[\|\hat{f} - \frac{i}{j} f_{\boldsymbol{D}^i}(q) - \frac{j-i}{j} f_{\boldsymbol{D}^{i:j}}(q)\| | \boldsymbol{D}^i]].$$

Now consider $\mathbb{E}_{\boldsymbol{D}^{i:j}\sim\chi}[\|\hat{f} - \frac{i}{j} f_{\boldsymbol{D}^i}(q) - \frac{j-i}{j} f_{\boldsymbol{D}^{i:j}}(q)\| | \boldsymbol{D}^i]$.

Given $\boldsymbol{D}^i$, $\hat{f} - \frac{i}{j} f_{\boldsymbol{D}^i}(q)$ is a fixed quantity. Furthermore, recall that $\arg\min_c \mathbb{E}[|X - c|] = \text{Med}(X)$ for any random variable X, where $\text{Med}(X)$ is a median of X (Wasan, 1970). Therefore, for any query,

$$\mathbb{E}_{\boldsymbol{D}^{i:j}\sim\chi}[\|\hat{f} - \frac{i}{j} f_{\boldsymbol{D}^i}(q) - \frac{j-i}{j} f_{\boldsymbol{D}^{i:j}}(q)\| | \boldsymbol{D}^i] \geq \frac{j-i}{j} \mathbb{E}_{\boldsymbol{D}^{i:j}\sim\chi}[\text{Med}(f_{\boldsymbol{D}^{i:j}}(q)) - f_{\boldsymbol{D}^{i:j}}(q)].$$

Observe that $f_{\boldsymbol{D}^{i:j}}(q) \sim Binomial(j-i, \mathbb{P}_{p\sim\chi}(I_{p\in q}))$ and consider any query such that $(j-i)\mathbb{P}_{p\sim\chi}(I_{p\in q})$ is an integer, which exists as long as the c.d.f of the distribution is continuous. For such queries, we have $\text{Med}(X) = (j-i)\mathbb{P}_{p\sim\chi}(I_{p\in q})$ since mean and median of binomial distributions where $(j-i)\mathbb{P}_{p\sim\chi}(I_{p\in q})$ is an integer are equal (Janson, 2021). Let $p_q = \mathbb{P}_{p\sim\chi}(I_{p\in q})$. Using the bound on the binomial mean absolute deviation in Berend & Kontorovich (2013), we have, when $j - i \geq 2$ and for any query s.t. $\frac{1}{(j-i)} \leq p_q \leq 1 - \frac{1}{j-i}$,

$$\frac{\sqrt{(j-i)p_q(1-p_q)}}{\sqrt{2}} \leq \mathbb{E}_{\boldsymbol{D}^{i:j}\sim\chi}[|(j-i)p_q - f_{\boldsymbol{D}^{i:j}}(q)|].$$

Moreover, setting $p_q = \frac{\lfloor \frac{j-i}{2} \rfloor}{j-i}$, we have

$$\frac{\sqrt{(j-i)p_q(1-p_q)}}{\sqrt{2}} \geq \frac{\sqrt{j-i}}{4}.$$

□

D.2. Proof of Theorem 3.5 (formally Theorem C.1)

First, we use the algorithm $\mathcal{A}$ (due to optimizability) to construct the algorithm in definition 3.2 as $\hat{f} = \frac{1}{n}\mathfrak{A}(f_{\boldsymbol{D}}, \alpha_{\mathcal{F}\to\mathcal{G}}(\frac{1}{\sqrt{n}}))$ for $f \in \{r, c\}$ given an input dataset, $\boldsymbol{D}$, of size n. Let $\vartheta = \alpha_{\mathcal{F}\to\mathcal{G}}(\frac{1}{\epsilon})$.

Note that since $\mathcal{F}$ has enough representation power to represent $\mathcal{G}$, and since by assumption $\{\chi \in \mathfrak{X}, f_\chi\} \subseteq \mathcal{G}$, we have that, for any χ there exists $\hat{f}_\chi \in \mathcal{F}_\vartheta$ s.t. $\|\hat{f}_\chi - f_\chi\|_\infty \leq \frac{1}{\sqrt{n}}$. Furthermore, since we approximately optimally find $\hat{f}$, we have $|\hat{f}(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| \leq \frac{1}{\varkappa}|\hat{f}_\chi(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| + \frac{\varkappa'}{\sqrt{n}}$. Now, to analyze accuracy of $\hat{f}$, observe that, for any input x we have

$$\begin{aligned}
|\hat{f}(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| &\leq \frac{1}{\varkappa}|\hat{f}_\chi(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| + \frac{\varkappa'}{\sqrt{n}} \\
&\leq \frac{1}{\varkappa}|\hat{f}_\chi(x) - f_\chi(x)| + \\
&\qquad \frac{1}{\varkappa}|f_\chi(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| + \frac{\varkappa'}{\sqrt{n}} \\
&\leq \frac{1}{\varkappa\sqrt{n}} + \frac{1}{\varkappa}|f_\chi(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| + \frac{\varkappa'}{\sqrt{n}}.
\end{aligned}$$

We also have

$$|\hat{f}(x) - f_\chi(x)| \leq |\hat{f}(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)| + |f_\chi(x) - \frac{1}{n} f_{\boldsymbol{D}}(x)|,$$

So that ,

$$n|\hat{f}(x) - f_\chi(x)| \leq \frac{\sqrt{n}}{\varkappa} + \frac{2}{\varkappa}|nf_\chi(x) - f_{\boldsymbol{D}}(x)| + \sqrt{n}\varkappa'.$$

By Hoeffeding's inequality, we have

$$\mathbb{P}(|nf_\chi(x) - f_{\boldsymbol{D}}(x)| \geq \epsilon') \leq e^{-2(\frac{\epsilon'}{\sqrt{n}})^2}, \tag{1}$$

So that

$$\mathbb{P}(n|f_\chi(x) - \hat{f}(x)| \geq \frac{2}{\varkappa}\epsilon' + \sqrt{n}(\frac{1}{\varkappa} + \varkappa')) \leq e^{-2(\frac{\epsilon'}{\sqrt{n}})^2}, \tag{2}$$

And therefore, for some universal constant $\varkappa_2$ and $\epsilon = \Omega(\sqrt{n})$,

$$\mathbb{P}(n|f_\chi(x) - \hat{f}(x)| \geq \epsilon) \leq e^{-\varkappa_2(\frac{\epsilon}{\sqrt{n}} - 1)^2}. \tag{3}$$

D.3. Proof of Lemma 3.6

For each distribution class, we show optimizability and representation power of some function class $\mathcal{F}$ that can be used to model the distribution class, which combined with Theorem C.1 shows the desired result for both Lemmas 3.6 and 3.7. Then, for each class, we discuss modeling complexities.

Distribution learnability for $\mathfrak{X}_\rho$. Let $\mathcal{F}$ be the class of piecewise constant functions with uniformly spaced pieces and let $\mathcal{G}$ be the class of real-valued differentiable functions $[0,1]^d \to \mathbb{R}$ with gradient bounded by ρ. Consider the number of pieces to use for approximation as a hyperparameter.

Optimizability. Given the number of pieces, the function that creates the minimum infinity norm is to place a constant at the mid-point of maximum and minimum values in each interval. That is, for an interval $I \subseteq [0,1]^d$, the constant approximating g over with the lowest infinity norm I is $\frac{1}{2}(\min_{x\in I} g(x) + \max_{x\in I} f(x))$. Note that this function has error at most $\max_{x\in I} g(x) - \frac{1}{2}(\min_{x\in I} g(x) + \max_{x\in I} f(x)) = \frac{1}{2}(\max_{x\in I} f(x) - \min_{x\in I} g(x))$. For efficiency purposes, instead of the optimal solution, we let the constant for the piece responsible for I be $g(p)$ for some $p \in I$. Note that for all $x \in I$ $|g(p) - g(x)| \leq |\max_{x\in I} g(x) - \min_{x\in I} g(x)|$, so that this construction gives us a $\frac{1}{2}$-approximation of the optimal solution.

Representation Power. Define $\alpha_{\mathcal{F}\to\mathcal{G}}(\epsilon) = \frac{\sqrt{d}\rho}{\epsilon}$. We show that for any $g \in \mathcal{G}$ and any $\epsilon > 0$, there is a function $\hat{f} \in \mathcal{F}_{\alpha(\epsilon)}$ s.t. $\|\hat{f} - g\|_\infty \leq \epsilon$. This function is the optimal solution as constructed above. To see why the error is at most ϵ, consider a partition over I with j-th dimension $[p_{j,i}, p_{j,i+1}]$, where $p_{j,i+1} - p_{j,i} = \frac{\epsilon}{\rho}$, and let x_1 and x_2 be the two points in I that, respectively, achieve the minimum and maximum of g in I. For any point in $x \in I$, our function approximator answer $\hat{f}(x) = \frac{1}{2}(g(x_1) + g(x_2))$. We have

that

$$\begin{aligned}|\hat{f}(x) - g(x)| &= |\frac{1}{2}(g(x_1) + g(x_2)) - g(x)| \\ &\leq \max\{g(x_2) - g(x), g(x) - g(x_1)\} \\ &\leq \|g'(x)\|_2 \|x_2 - x\|_2 \\ &\leq \rho(\sqrt{d}\frac{\epsilon}{\sqrt{d}\rho}) = \epsilon.\end{aligned}$$

Model Complexity. The inference time, $\mathcal{T}_n^{\mathfrak{X}}$, is constant independent of the number of pieces used. The space complexity is the number of pieces multiplied by the space to store each constant. Given that g consists of integers between 0 to n, $\mathcal{S}_n^{\mathfrak{X}}$ can be stored in $O(\sqrt{d}(\rho\sqrt{n})^d \log n)$. Finally, for rank operation, the algorithm that outputs the function optimizer makes $\rho\sqrt{n}$ calls to g, so that building the function approximator can be done in $O(\rho\sqrt{n}\log n)$, assuming the data is sorted (so that each call to g takes $O(\log n)$). For cardinality estimation there are $O(\sqrt{d}(\rho\sqrt{n})^d \log n)$ calls to the cardinality function, where each call in the worst case takes $O(n)$ (this can optimized by building high dimensional indexes). Thus, in this case, $\mathcal{B}_n^{\mathfrak{X}} = O(\sqrt{d}(\rho\sqrt{n})^d \log n)$

Distribution learnability for $\mathfrak{X}_l$. Let $\mathcal{F}$ and $\mathcal{G}$ be the class of piecewise linear functions with at most l pieces (not necessarily uniformly spaced pieces). Trivially, $\mathcal{F}$ has enough representation power to represent $\mathcal{G}$, thus, it remains to show optimizability and model complexity.

Optimizability. The PLA algorithm, $\mathbf{P}(\epsilon)$ by (O'Rourke, 1981), used in PGM index (Ferragina & Vinciguerra, 2020), is able to find the piecewise linear solution with the smallest number of pieces given an error ϵ. Here, we want to achieve the opposite, i.e., given a number of pieces find piecewise linear approximation with smallest error. Note that ϵ is in the range 0 to 1, and we can do a binary search on the values of ϵ, for each calling $\mathbf{P}(\epsilon)$ until we find the smallest ϵ where $|\mathbf{P}(\epsilon)| \leq l$. Note that since suboptimality of $O(\frac{1}{\sqrt{n}})$ in ϵ is allowed, wee can discretize $[0, 1]$ to $\sqrt{n}$ groups, and only do binary search over this discrete set, which takes $O(\log(\sqrt{n}))$ calls to $\mathbf{P}(\epsilon)$, and each call takes $O(n)$ operations (Ferragina & Vinciguerra, 2020) on a sorted array, so that $\mathcal{F}$ is optimizable to with the algorithm running in $O(n \log n)$

Model Complexity. The learning time $n\mathcal{B}_n^{\mathfrak{X}} = O(n \log n)$ is discussed above. The algorithm always returns l pieces which can be evaluated in $\mathcal{T}_n^{\mathfrak{X}} = O(\log l)$ time. Note the each linear piece can be adjusted to cover an interval starting and ending at points in the dataset (so the interval can be stored as pointers to corresponding dataset item). Moreover, the beginning and end of each line can be adjusted to be an integer (since the rank function only returns integers), similar to (Ferragina & Vinciguerra, 2020), so that the lines can be stored in $\mathcal{S}_n^{\mathfrak{X}} = O(l \log n)$.

Algorithm 1 Dynamic Learned Index Query

Require: New element to be inserted in tree rooted at N
Ensure: Balanced data structure

```
procedure QUERY(q, N)
    if N is a leaf node then
        return BINARYSEARCH(N.content)
    î ← N.f̂(p)
    i ← EXPSEARCH(p, î, N.content)
    j ← BINARYSEARCH(p, N.children[i])
    return QUERY(q, N.children[i][j])
```

Algorithm 2 Dynamic Learned Index Insertions

Require: New element, p to be inserted in tree rooted at N
Ensure: Index with p inserted

```
procedure INSERT(p, N)
    N.counter++
    if N.children is NULL then
        INSERTCONTENT(p, N.content)
        return
    î ← N.f̂(p)
    i ← EXPSEARCH(p, î, N.content)
    j ← BINARYSEARCH(p, N.children[i])
    INSERT(p, N.children[i][j])
    if N.counter = N.max_points then
        A ← the sorted array in the index rooted at N
        if N has no parent then
            return REBUILD(A)
        P ← parent of N
        i_p ← index of N in P.children
        Remove N from P.children[i_p]
        N_1 ← REBUILD(A[: N.max_points/2])
        N_2 ← REBUILD(A[N.max_points/2 :])
        Insert N_1 and N_2 in P.children[i_p]
```

Distribution Learnability for $\mathfrak{X}_c$. Trivially, the class $\mathcal{F}$ containing the distribution operation function has enough approximation power for $\mathfrak{X}_c$ and is optimizable with $\mathcal{S}_n^{\mathfrak{X}}, \mathcal{T}_n^{\mathfrak{X}}, \mathcal{B}_n^{\mathfrak{X}}$ all $O(1)$.

D.4. Proofs for Learned Indexing

D.4.1. Index Operations

The index supports two operations, QUERY and INSERT, which are presented in Algs. 1 and 2. Recall that $\mathcal{A}$ is an algorithm defined in Definition 3.2 and exists due to distribution learnability of $\mathfrak{X}$. The index builds a tree structure similar to (Ding et al., 2020), with each node containing a model, and a set of children. An overview of the tree architecture is shown in Fig. 1. Each node can be seen to cover a subarray of the original indexed array. If the size of the covered subarray is k elements, then the node will have $\sqrt{k}$ children, where the subarray of size k is equally divided between the children (so each child covers $\sqrt{k}$ elements).

Algorithm 3 Procedure for for Rebuilding Root

Require: A sorted array, A
Ensure: A learned index rooted at a new node N

procedure REBUILD(A)
  $N \leftarrow$ new node
  $k \leftarrow |A|$
  N.`counter` $\leftarrow k$
  if $k \leq \varkappa$ **then**
    N.`content` $\leftarrow A$
    return N
  N.`max_points` $\leftarrow 2k$
  $N.\hat{f} = \mathcal{A}(A)$
  N.`content` $\leftarrow A[:: \sqrt{k}]$
  for i in $\sqrt{k}$ **do**
    $N_c \leftarrow$ REBUILD($A[ik : (i+1)k]$)
    N.`children`.$append(N_c)$
  return N

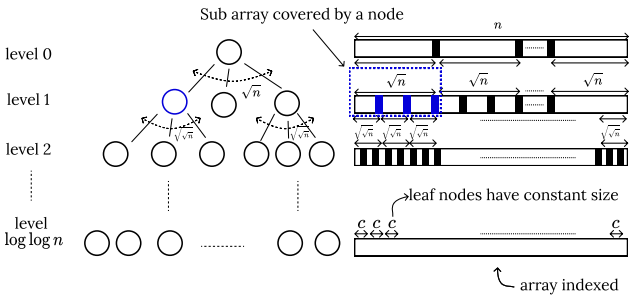

Figure 1. Structure of the learned dynamic index

The root node covers the entire n elements of the array, and therefore has $\sqrt{n}$ children. As can be seen, the number of children of the nodes decreases as we go down the tree. A node won't have any children if its covered subarray is smaller than some constant c. Moreover, shown as black elements in the figure, each parent node stores the minimum value of the subarray covered by each of its children in an array called the node's `content`. Thus, the node's `content` can be used to traverse the tree. When a node's model predicts which child the node should travel, the node first checks with its `content` to make sure it is the correct node. This is done by doing an exponential search on the node's `content`.

During insertions, each node keeps a counter of the number of points inserted through it. If a node is at level i, then at most $k^{2^{-i}}$ elements are allowed in the node for $i > 0$, where k is the size of the dataset at the time of construction of the current root node (root node's are periodically rebuilt). If the number of insertions reaches $k^{2^{-i}}$, the node splits. When a node splits the subarray it covers is split into two, and an entirely new subtree is built for each half of the subarray, rebuilding all models. To avoid splits affecting parent nodes, as Fig. 2 shows, the newly created node is appended to the list of children (we'll discuss how exactly this is done later).

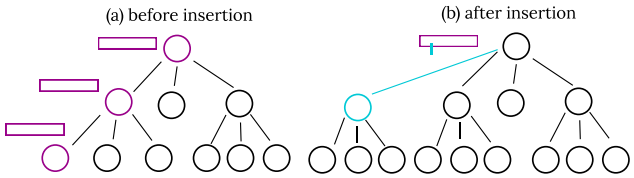

Figure 2. Insertion Causing a split in index

Finally, the root is rebuilt every time its size doubles.

To support the splitting discussed above, the children are arranged in a two dimensional array N.`children`. We refer to children pointed to in N.`children`$[i]$ as children in the i-th child slot. Each child slot contains a sorted list of at least one, but a variable number, of children, where the list is kept sorted using binary trees. To find which node to traverse, we first find the correct node slot with the help of the learned model, and then use the binary tree in the node slot to find the correct child. Leaf nodes have `content` storing the data. The index keeps a counter at each node, and periodically rebuilds the tree rooted at a node. Thus, the two operations are performed as follows.

Query. Performing queries on an index with root node N is similar to performing queries with a B-tree, where nodes are recursively traversed until reaching a leaf node. The only difference is how we decide which child to search. This is done by, for each non-leaf node, first asking a model to estimate which child to search. Then, the model estimate is corrected by performing a local exponential search. Since a child might have been split, the search then uses the binary tree at the correct node slot to find the correct child.

Insertions. Insertions first traverses the tree similar to the queries, with the extra addition that a counter in each node is incremented if a new element is inserted in that node. If the counter of a node passes the maximum size of the node, the node is split into two, and the parent meta data for the corresponding node slot is updated. The only exception is the root node, which does not split, but triggers a full rebuild of the entire tree.

D.4.2. QUERY TIME

Queries are performed by recursively searching each node, where a single node per level is queried. Consider the number of operations performed at the i-th level. Each level performs a model inference, exponential search on the node's content and a binary search on the node's extension. We consider each separately.

Model Inference. A model at the i-th level is built on at most $n^{\frac{1}{2^i}}$ elements. Thus, the inference time is $O(\mathcal{T}^{\mathfrak{X}}_{n^{2^{-i}}})$.

Exponential Search. The time complexity of the exponential search step depends on the accuracy of the model estimate. We show that, on expectation, the time complexity

is constant.

Note that neither the content of each node, nor its model gets modified by insertions, unless a node is rebuilt. Thus, we only need to show the statement for right after the node construction. Assume the node is constructed for some array A with $|A| = k$ for some integer k.

Assume the i-th element of A is a sample originally obtained from the distribution χ_i, let $R = [l, u]$ define the range the elements A have fallen into passed to the algorithm. The elements of A are independent samples from conditional distributions $\chi_1 \mid R, ..., \chi_k \mid R$, respectively. Let $\chi_R = \{\chi_1 \mid R, ..., \chi_k \mid R\}$.

Applying the extension of the DKW bound to independent but non-identical random variables (Shorack & Wellner, 1986, Chapter 25.1) we have that

$$\mathbb{P}(\|kr_{\chi_R} - r_A\|_\infty \geq \sqrt{\frac{k}{2}}\epsilon) \leq 2e^{-\epsilon^2+1}. \tag{4}$$

Furthermore, for $\hat{r}$, the model of $r_{\chi_R} = \frac{1}{k}\sum_{i=1}^{k} r_{\chi_i|R}$, obtained from $\mathcal{A}$ and by the accuracy requirement of distribution learnability, we have for any $\epsilon \geq \sqrt{\varkappa_2}(\frac{1}{\sqrt{k}} - 1)$

$$\mathbb{P}(\|kr_{\chi_R} - \hat{r}\|_\infty \geq \sqrt{k}(\frac{\epsilon}{\sqrt{\varkappa_2}} + 1)) \leq \varkappa_1 e^{-\epsilon^2}. \tag{5}$$

By union bound on Ineq. 4 and 5, and the triangle inequality, for any $\epsilon \geq \sqrt{k}$, we have

$$\mathbb{P}(\|\hat{r} - r_A\|_\infty \geq \epsilon) \leq \varkappa_1' e^{-\varkappa_2'(\frac{\epsilon}{\sqrt{k}}-1)^2}, \tag{6}$$

where $\varkappa_1' = 2e + \varkappa_1$ and $\varkappa_2' = \frac{2\varkappa_2}{(\sqrt{2}+\sqrt{\varkappa_2})^2}$.

Now let $N(q)$ be the number of operations by performed by exponential search for a query q. Recall that the content of the node is $A_c = A[::\sqrt{k}]$ and that we use $\hat{r}_{A_c} = \lceil \frac{1}{\sqrt{k}}\hat{r}(q) \rceil$ as the start location to start searching for q in A_c with exponential search. We have the true location of q in A_c is $r_{A_c}(q) = \lceil \frac{1}{\sqrt{k}} r_A(q) \rceil$. Observe that, for any q

$$\|r_A - \hat{r}\|_\infty > (\hat{r}_{A_c}(q) - r_{A_c}(q)| - 2)\sqrt{k}, \tag{7}$$

and that it is easy to see that for exponential search we have

$$|\hat{r}_{A_c}(q) - r_{A_c}(q)| \geq 2^{\frac{N(q)}{2}-1}, \tag{8}$$

Combining which we get, for any query q,

$$\|r_A - \hat{r}\|_\infty > (2^{\frac{N(q)}{2}-1} - 2)\sqrt{k}.$$

So that, for any i, $N(q) \geq i$ implies that $\|r_A - \hat{r}\|_\infty > (|2^{\frac{i}{2}-1} - 2)\sqrt{k}$ Thus, we have

$$\begin{aligned}\mathbb{E}_{A\sim\chi}[N(q)] &= \sum_{i=1}^{2\log k} \mathbb{P}_{A\sim\chi}(N(q) \geq i) \\ &\leq \sum_{i=1}^{2\log k} \mathbb{P}_{A\sim\chi}(\|r_A - \hat{r}\|_\infty > (|2^{\frac{i}{2}-1} - 2)\sqrt{k}) \\ &\leq 5 + \varkappa_1' \sum_{i=6}^{2\log k} e^{-\varkappa_2'(2^{\frac{i}{2}-1}-3)^2} = O(1)\end{aligned}$$

Thus, the expected time searching performing exponential search is $O(1)$.

Searching Node's Extension. Next, we study the expected time for searching the additional list added to the nodes. Let L be the size of the list. Note that the lists are created for all the nodes except the root. Moreover, a non-root node at level i has capacity $cn^{\frac{1}{2^i}}$, it will split every $cn^{\frac{1}{2^i}}$ insertions into the parent, with each split adding an element to the parent's extension list. Furthermore, the node gets rebuilt after every $k = n^{\frac{1}{2^{i-1}}}$ insertions into its parent. Thus, if k_N elements out of $n^{\frac{1}{2^{i-1}}}$ get inserted into a node slot N, the number of splits for that slot will be $\lfloor \frac{k_N}{cn^{\frac{1}{2^i}}} \rfloor$.

Assume the k new insertions into the parent, N_p of N since the last rebuild of the parent were from r.v.s with distribution $\chi_1, ..., \chi_k$. Given that they fall in N_p, their conditional distribution is $\chi_1 \mid R, ..., \chi_k \mid R$ for R defining an interval for which the node N_P was built. Let $\chi_R' = \{\chi_1 \mid R, ..., \chi_k \mid R\}$. Furthermore, let χ_R be the original distribution the model of in the parent node was created based on. Let A' be the set of k insertions and let A be the set of points based on which the parent of N was built. The number of insertions out of the k new insertions into the j-th node slot is $r_{A'}(N_j) - r_{A'}(N_{j-1})$. To study this quantity, recall that the j-th slot was created so that

$$r_A(N_j) - r_A(N_{j-1}) = n^{\frac{1}{2^i}}. \tag{9}$$

Thus, we first relate r_A and r_A'.

The elements of A and A' are independent so that applying the extension of the DKW bound to independent but non-identical random variables (Shorack & Wellner, 1986, Chapter 25.1) for both A and A' we have

$$\mathbb{P}(\|kr_{\chi_R'} - r_{A'}\|_\infty \geq \sqrt{\frac{k}{2}}\epsilon) \leq 2e^{-\epsilon^2+1}, \tag{10}$$

$$\mathbb{P}(\|kr_{\chi_R} - r_A\|_\infty \geq \sqrt{\frac{k}{2}}\epsilon) \leq 2e^{-\epsilon^2+1}. \tag{11}$$

Moreover, assume TV($\chi_i \mid R, \chi_j \mid R$)$\leq \delta$ for all i, j, we have that $\|r_\chi - r_{\chi'}\|_\infty \leq \delta$. Combining this with Ineq. 10

and 11 and using the triangle inequality, we have

$$\mathbb{P}(\|r_{A'} - r_A\|_\infty \geq \sqrt{2k}\epsilon + \delta k) \leq 4e^{-\epsilon^2+1}. \quad (12)$$

Finally, combining Ineq. 12 with Eq. 9, and recalling that $k = n^{2^{-i+1}}$ implies that

$$\mathbb{P}(r_{A'}(N_j) - r_{A'}(N_{j-1}) \geq n^{2^{-i}}(1 + 2\sqrt{2}\epsilon) + 2\delta n^{2^{-i+1}}) \leq 4e^{-\epsilon^2+1}.$$

And therefore

$$\mathbb{P}(\lfloor \frac{r_{A'}(N_j) - r_{A'}(N_{j-1})}{n^{2^{-i}}} \rfloor \geq 2\sqrt{2}\epsilon + 2\delta n^{2^{-i}}) \leq 4e^{-\epsilon^2+1},$$

Where $\lfloor \frac{r_{A'}(N_j) - r_{A'}(N_{j-1})}{n^{2^{-i}}} \rfloor$ is the number of splits of the j-th node slot. Let S_j denote this random variable. We have

$$\begin{aligned}\mathbb{E}[S_j] &= \sum_{i=0}^{k} \mathbb{P}(S_j \geq i) \\ &\leq 2\delta n^{2^{-i}} + \sum_{i=0}^{k-2\delta n^{2^{-i}}} \mathbb{P}(S_j \geq i + 2\delta n^{2^{-i}}) \\ &\leq 2\delta n^{2^{-i}} + \sum_{i=0}^{k-2\delta n^{2^{-i}}} 4e^{-\frac{i^2}{8}+1} = O(\delta n^{2^{-i}})\end{aligned}$$

Finally, we are interested in $\mathbb{E}[\log(S_j)] \leq \log(\mathbb{E}[S_j]) = O(\log(n^{\frac{1}{2^i}}\delta))$.

Total Query Time. Thus, the expected time to search a node at the i-th level to find its children is $O(\mathcal{T}^{\mathfrak{X}}_{n^{2^{-i}}} + \log(n^{-2^i}\delta))$. Thus, the total time to search the tree is $O(\sum_{i=1}^{\log\log n} \mathcal{T}^{\mathfrak{X}}_{n^{2^{-i}}} + \log(n^{-2^i}\delta_i))$. We can bound this as $O(\mathcal{T}^{\mathfrak{X}}_n \log\log n + \log(n\bar{\delta}))$, where $\bar{\delta} = \min\{\delta, \delta_c^{\log\log n}\}$.

D.4.3. Insertion Time

Note that insertion time is equal to query time plus the total cost of rebuilds. Next, we calculate the cost of rebuilds.

Let $T(N)$ be the amortized cost of inserting N elements into a tree that currently has N elements and was just rebuilt at its root, so that $NT(N)$ will be the total insertion cost for the N elements. Note that the amortized cost of all n insertions starting from a tree with one 1 element is at most $\frac{1}{n}\sum_{i=0}^{\log n} \frac{n}{2^i} T(\frac{n}{2^i}) \leq T(n) \sum \frac{1}{2^i} = O(T(n))$.

Thus, we only need to study $T(n)$. Note that when inserting n elements into a tree that currently has n elements and was just rebuilt at its root, the height of the tree remains constant throughout insertions. Furthermore, at the i-th level, $i \geq 0$, there will be at most $\frac{n}{n^{2^{-i}}}$ rebuilds and each rebuild costs

$$\sum_{j=i}^{\log\log n} \frac{2n^{2^{-i}}}{n^{2^{-j}}} \mathcal{B}^{\mathfrak{X}}_{2n^{2^{-j}}}.$$

Thus, we have the amortized cost of all rebuilds is

$$\frac{1}{n} \sum_{i=0}^{\log\log n} \frac{n}{n^{2^{-i}}} \sum_{j=i}^{\log\log n} \frac{2n^{2^{-i}}}{n^{2^{-j}}} \mathcal{B}^{\mathfrak{X}}_{2n^{2^{-j}}} = O(\sum_{i=0}^{\log\log n} (i+1) \frac{\mathcal{B}^{\mathfrak{X}}_{2n^{2^{-i}}}}{n^{2^{-i}}}).$$

Thus, the total cost of insertions is $O(\mathcal{T}^{\mathfrak{X}}_n \log\log n + \log(\bar{\delta}n) + \frac{\mathcal{B}^{\mathfrak{X}}_n}{n} \log^2\log n))$. □

D.4.4. Space Overhead

After n insertions, we will have $\log\log n$ levels. Right after the root was rebuilt, level i has at most $\frac{n}{n^{2^{-i}}}$ models. If n further insertions are performed, each level will have at most $\frac{n}{n^{2^{-i}}}$ new models. Thus, the total size of the models at level i is at most $2\frac{n}{n^{2^{-i}}}\mathcal{S}^{\mathfrak{X}}_{n^{2^{-i}}}$, and thus the total size of all models is $O(n\sum_{i=0}^{\log\log n} \frac{\mathcal{S}^{\mathfrak{X}}_{n^{2^{-i}}}}{n^{2^{-i}}})$. Furthermore, the total number of nodes in the tree is $O(n)$, and for each node we store a pointer to it and its lower and upper bounds, as well as a counter, which can be done in $O(\log n)$. Thus, the total space consumption is $O(n(\log n + \sum_{i=0}^{\log\log n} \frac{\mathcal{S}^{\mathfrak{X}}_{n^{2^{-i}}}}{n^{2^{-i}}}))$.

D.5. Proof of Corollary 4.2

Observe that for any distribution in $\mathfrak{X}$, we have that $\chi|R$ for any interval R has p.d.f at most $\frac{\rho_2}{\rho_1}$. According to Lemma 3.6, distributions with p.d.f at most $\frac{\rho_2}{\rho_1}$ are distribution learnable. Substituting the complexities proves Corollary.

D.6. Proofs for Cardinality Estimation

D.6.1. High Dimensions (Theorem 4.3)

Construction. By distribution learnablility we have an algorithm $\mathcal{A}$ that builds a model that we use for estimation. To prove the lemma, we use $\mathcal{A}$ and periodically rebuild models to answer queries. Specifically, $\mathcal{A}$ is called every k insertions, where if $\delta \geq \frac{2\varkappa}{\sqrt{n}}$, $k = \frac{\phi-\varkappa}{2\varkappa\delta}\sqrt{n}$, and when $\delta \leq \frac{2\varkappa}{\sqrt{n}}$, $k = n \times \min\{(\frac{\phi-\varkappa}{\varkappa(1+2\varkappa)})^2, 1\}$. That is, if currently there are n points inserted, and we insert n' new points, for $n' < k$, the algorithm will answer queries as $(n+n')\hat{c}$. However, when $n' = k$, the algorithm rebuilds the model and starts answering queries using the new model.

Query Time and Space Consumption. We use a single model with no additional data structure, so query time is $O(\mathcal{T}^{\mathfrak{X}}_n)$ and space complexity is $O(\mathcal{S}^{\mathfrak{X}}_n)$.

Insertion Complexity. To analyze the cost of insertions, first consider, $T(n)$, the total number of operations when we insert n new elements in a data structure that already has n elements. Consider the two cases where $\delta \geq \frac{2\varkappa}{\sqrt{n}}$

and $\delta \leq \frac{2\varkappa}{\sqrt{2}}$. In the first case, we have the we rebuild the model every $\frac{\phi-\varkappa}{2\varkappa\delta}\sqrt{n}$ insertions, so that there are at most $\frac{n}{\frac{\phi-\varkappa}{2\varkappa\delta}\sqrt{n}} = \frac{2\varkappa\delta}{\phi-\varkappa}\sqrt{n}$ rebuilds. If $\delta \geq \frac{2\varkappa}{\sqrt{2}}$, we rebuild the model every ρn times, so that the total number of rebuilds is $\frac{1}{\rho}$. Ensuring that $\phi \geq \varkappa + 1$, we have that $\frac{1}{\rho} \leq \varkappa^2(2\varkappa+1)^2$. In either case, each rebuild costs $O(\mathcal{B}^{\mathfrak{X}}_{2n})$ and besides rebuilds insertions takes constant time. Thus, if $\delta \geq \frac{2\varkappa}{\sqrt{n}}$, $T(n) = O(n + \frac{\delta}{\phi}\sqrt{n}\mathcal{B}^{\mathfrak{X}}_{2n})$ and if $\delta \leq \frac{2\varkappa}{\sqrt{2}}$ we have $T(n) = O(n + \mathcal{B}^{\mathfrak{X}}_{2n})$.

Next, to analyze the total runtime of starting from 0 elements and inserting n new elements, we have that the amortized insertion is $\frac{1}{n}\sum_{i=1}^{\log n} T(\frac{n}{2^i})$. Now if $\delta \geq \frac{2\varkappa}{\sqrt{n}}$, this is $O(\frac{1}{n}\sum_{i=1}^{\log n}\frac{n}{2^i} + \frac{\delta}{\phi}\sqrt{\frac{n}{2^i}}\mathcal{B}^{\mathfrak{X}}_{\frac{2n}{2^i}}) = O(\frac{1}{n}\sum_{i=1}^{\log n}\frac{\delta}{\phi\sqrt{n2^i}}\mathcal{B}^{\mathfrak{X}}_{\frac{2n}{2^i}}) = O(n + \frac{\delta}{\phi}\sqrt{n}\mathcal{B}^{\mathfrak{X}}_{2n})$. Furthermore, if $\delta \leq \frac{2\varkappa}{\sqrt{n}}$, this is $O(n + \mathcal{B}^{\mathfrak{X}}_{2n})$. Thus, the amortized insertion cost is $O(\max\{\frac{\delta}{\phi\sqrt{n}}, \frac{1}{n}\}\mathcal{B}^{\mathfrak{X}}_{n})$

Accuracy. We show that if $\delta \geq \frac{2\varkappa}{\sqrt{n}}$, rebuilding the model every $\frac{\phi-\varkappa}{2\varkappa\delta}\sqrt{n}$, and when $\delta \leq \frac{2\varkappa}{\sqrt{n}}$ rebuilding the model every ρn for $\rho = \min\{(\frac{\phi-\varkappa}{\varkappa(1+2\varkappa)})^2, 1\}$ insertions is sufficient to answer queries with error at most $\phi\sqrt{n}$, whenever $\phi \geq \varkappa + 1$.

Assume a model was built using dataset $\boldsymbol{D}^i$, i.e., after i insertions. We study the error in answering after k new insertions, so that the goal is to answer queries on $\boldsymbol{D}^j$, $j = i + k$. Let χ and χ' be the distributions so that $\boldsymbol{D}^i \sim \chi$ and $\boldsymbol{D}^{i:j} \sim \chi'$.

Consider a model, $\hat{c}$ that was built on $\boldsymbol{D}^i$ using $\mathcal{A}$, so we have

$$\mathbb{P}(i\|\hat{c} - c_\chi\|_\infty \geq \sqrt{i}(\frac{\epsilon}{\sqrt{\varkappa_2}} + 1)) \leq \varkappa_1 e^{-\epsilon^2} \tag{13}$$

We are interested

$$\begin{aligned}\|j\hat{c} - c_{\boldsymbol{D}^j}\| &= \|i\hat{c} + k\hat{c} - c_{\boldsymbol{D}^i} - c_{\boldsymbol{D}^{i:j}}\| \\ &\leq \|i\hat{c} - c_{\boldsymbol{D}^i}\| + \|k\hat{c} - c_{\boldsymbol{D}^{i:j}}\|. \end{aligned}\tag{14}$$

For the first term, by Hoeffding's inequality we have

$$\mathbb{P}(|ic_\chi(q) - c_{\boldsymbol{D}^i}(q)| \geq \sqrt{i}\epsilon) \leq e^{-2\epsilon^2}, \tag{15}$$

Which combined with Ineq. 13 gives

$$\begin{aligned}\mathbb{P}\Big(|i\hat{c}(q) - c_{\boldsymbol{D}^i}(q)| \geq \sqrt{i}((1 + \frac{1}{\sqrt{\varkappa_2}})\epsilon + 1)\Big) \\ \leq (1 + \varkappa_1)e^{-2\epsilon^2}.\end{aligned}\tag{16}$$

For the second term, again by Hoeffding's inequality we have

$$\mathbb{P}(|kc_\chi(q) - c_{\boldsymbol{D}^{i:j}}(q)| \geq \sqrt{k}\epsilon) \leq e^{-2\epsilon^2}, \tag{17}$$

We also have that $\|\chi - \chi'\| \leq \delta$, which combined with Ineq. 17 gives

$$\mathbb{P}(|kc_\chi(q) - c_{\boldsymbol{D}^{i:j}}(q)| \geq \sqrt{k}\epsilon + k\delta) \leq e^{-2\epsilon^2},$$

And therefore, using Ineq. 13, we have

$$\begin{aligned}\mathbb{P}\Big(|k\hat{c}(q) - c_{\boldsymbol{D}^{i:j}}(q)| \geq \sqrt{k}\epsilon + \frac{k}{\sqrt{i}}(\frac{\epsilon}{\sqrt{\varkappa_2}} + 1) + k\delta\Big) \\ \leq (\varkappa_1 + 1)e^{-2\epsilon^2}.\end{aligned}\tag{18}$$

Combining Ineq. 14, 16 and 18, we have

$$\begin{aligned}\mathbb{P}\Big(|j\hat{c}(q) - c_{\boldsymbol{D}^j}(q)| \geq (\sqrt{i} + \frac{\sqrt{i}}{\varkappa_2} + \sqrt{k} + \frac{k}{\sqrt{i\varkappa_2}})\epsilon + \\ \frac{k}{\sqrt{i\varkappa_2}} + \sqrt{i} + k\delta\Big) \\ \leq 2(\varkappa_1 + 1)e^{-2\epsilon^2}\end{aligned}$$

As such, we have $\mathbb{E}[\frac{|j\hat{c}(q) - c_{\boldsymbol{D}^j}(q)| - (\frac{k}{\sqrt{i\varkappa_2}} + \sqrt{i} + k\delta)}{\sqrt{i} + \frac{\sqrt{i}}{\varkappa_2} + \sqrt{k} + \frac{k}{\sqrt{i\varkappa_2}}}] \leq \varkappa_3$, for some universal constant $\varkappa_3$ so that, $\mathbb{E}[|j\hat{c}(q) - c_{\boldsymbol{D}^j}(q)|] \leq \frac{k}{\sqrt{i\varkappa_2}} + \sqrt{i} + k\delta + \varkappa_3\sqrt{i} + \frac{\varkappa_3\sqrt{i}}{\varkappa_2} + \varkappa_3\sqrt{k} + \frac{\varkappa_3 k}{\sqrt{i\varkappa_2}}$. Assuming $k \leq i$, we have

$$\mathbb{E}[|j\hat{c}(q) - c_{\boldsymbol{D}^j}(q)|] \leq \varkappa(\sqrt{i} + \sqrt{k} + k\delta),$$

For some universal constant $\varkappa$.

Now if $\delta \geq \frac{2\varkappa}{\sqrt{i}}$ we let $k = \frac{\phi-\varkappa}{2\varkappa\delta}\sqrt{i}$ and otherwise set $k = \rho i$ for $\rho = \min\{(\frac{\phi-\varkappa}{\varkappa(1+2\varkappa)})^2, 1\}$.

First, consider the case where $\delta \geq \frac{1}{\sqrt{j}}$. Consider the error $\epsilon = \phi\sqrt{j} \geq \phi\sqrt{i}$, so it suffices to show that the error is at most $\phi\sqrt{i}$. Let $k = \frac{\phi-\varkappa}{2\varkappa\delta}\sqrt{i}$. Substituting this in, we want to show $\sqrt{\frac{\varkappa(\phi-\varkappa)}{2\delta}}\sqrt{i} - \frac{\phi-\varkappa}{2}\sqrt{i} \leq 0$. Indeed, for $\delta \geq \frac{2\varkappa}{\sqrt{i}}$, we have $\frac{\varkappa(\phi-\varkappa)}{2\delta}\sqrt{i} \leq \frac{(\phi-\varkappa)}{4}i$, so that

$$\begin{aligned}\sqrt{\frac{\varkappa(\phi-\varkappa)}{2\delta}}\sqrt{i} - \frac{\phi-\varkappa}{2}\sqrt{i} &\leq \frac{1}{2}\sqrt{(\phi-\varkappa)i} - \frac{\phi-\varkappa}{2}\sqrt{i} \\ &= \frac{1}{2}\sqrt{(\phi-\varkappa)i}(1 - \sqrt{\phi-\varkappa}) \\ &\leq 0\end{aligned}$$

Which proves $\mathbb{E}[|j\hat{c}(q) - c_{\boldsymbol{D}^j}(q)|] \leq \phi\sqrt{j}$ whenever $\delta \geq \frac{2\varkappa}{\sqrt{i}}$ and $\phi - \varkappa \geq 1$.

If $\delta \leq \frac{2\varkappa}{\sqrt{i}}$, we set $k = \rho i$ for $\rho = \min\{(\frac{\phi-\varkappa}{\varkappa(1+2\varkappa)})^2, 1\}$. We have

$$\varkappa(\sqrt{i} + \sqrt{\rho i} + \rho i\delta) \leq \varkappa\sqrt{i}(1 + \sqrt{\rho} + 2\sqrt{\rho}\varkappa)$$

So that we need to ensure $1 + \sqrt{\rho}(1 + 2\varkappa) \leq \frac{\phi}{\varkappa}$. Observe that $\sqrt{\rho} \leq \frac{\phi-\varkappa}{\varkappa(1+2\varkappa)}$ implies the above, so that setting ρ as above proves the result in this case.

D.6.2. One Dimension (Theorem 4.4)

Construction. We build a B-tree like data structure. However, in addition to the content of each node, each node also keeps a counter of the number of elements inserted into the node. After each insertion, if a leaf node has more than k elements, the node is split into two, for $k = \frac{\epsilon^2}{4(\varkappa+1)^2}$. Thus, leaf nodes cover between $\frac{k}{2}$ to k elements, while the rest of the tree has its own fanout B. Leaf nodes do not store the elements associated with them, but build models to answer queries. To answer a query, the tree is traversed similar to typical range query answering with a B-tree. However, if a node is fully covered in a range, then the number of insertions in the node is used to answer queries. Otherwise, the node is recursively searched until reaching a leaf node. We will have at most 2 leaf nodes reached that will be partially covered by the query range. Finally, the model of each node are constructed by using the construction in Theorem 4.3 with $\phi = \varkappa + 1$.

Query Time and Space Consumption. Each query will take $O(\log n + 2\mathcal{T}_k^{\mathfrak{X}})$ where $\log n$ is due to the tree traversal and $2\mathcal{T}_k^{\mathfrak{X}}$ for the two model inferences needed. Moreover, the total space consumption is $O(\frac{n}{k}\mathcal{S}_k^{\mathfrak{X}} + \frac{n}{k}\log n)$

Insertion Complexity. Each leaf node will start with $\frac{k}{2}$ elements and will be split whenever it reaches k elements. Insertion of the $\frac{k}{2}$ elements in a node cost $O(k + \max\{\delta\sqrt{k}, 1\}\mathcal{B}_k^{\mathfrak{X}})$. Given n insertions, we have $\frac{2n}{k}$ insertions of $\frac{k}{2}$ elements in the nodes, so that the amortized cost of rebuilds is $O(\frac{1}{n}\frac{n}{k}(k + \max\{\delta\sqrt{k}, 1\}\mathcal{B}_k^{\mathfrak{X}})) = O(\max\{\frac{\delta}{\sqrt{k}}, \frac{1}{k}\}\mathcal{B}_k^{\mathfrak{X}}))$. Furthermore, traversing the tree nodes costs $O(\log n)$ per insertion, so that amortized insertion cost is $O(\max\{\frac{\delta}{\sqrt{k}}, \frac{1}{k}\}\mathcal{B}_k^{\mathfrak{X}}) + \log n) = O(\max\{\frac{\delta}{\epsilon}, \frac{1}{\epsilon^2}\}\mathcal{B}_{\epsilon^2}^{\mathfrak{X}}) + \log n)$.

Accuracy. Setting $\phi = \varkappa + 1$ in Theorem 4.3 and having $k \leq \frac{\epsilon^2}{4(1+\varkappa)^2}$ ensures that the expected error of the each model is at most $\phi\sqrt{k} = \frac{\epsilon}{2}$. Since each query is answered by making two model calls, the total expected error for answering queries is at most ϵ as required.

D.6.3. High dimension with any accuracy

Here we also discuss how we can use models to answer queries to arbitrary accuracy in high dimensions. We note that, as we see here, building data structures to answer queries is high dimension is difficult. We discuss this result only in the static setting.

Lemma D.1. *There exists a learned model that can answer cardinality estimation query with error up to ϵ with query time $O(\mathcal{T}^{\mathfrak{X}}_{(\frac{\epsilon}{2d})^2} + (\frac{4d^2}{\epsilon^2})^d \Pi_i k_i)$ and space complexity $O(\frac{4d^2 n}{\epsilon^2}\mathcal{S}^{\mathfrak{X}}_{(\frac{\epsilon}{2d})^2} + (\frac{4d^2 n}{\epsilon^2})^d)$, where k_i is the cardinality of the query in the i-th dimension.*

Algorithm 4 Cardinality Estimation with Grid

Require: Query $\boldsymbol{q}$, dimension to refine, i, set of models, M, and set of partition points S
Ensure: Estimate to cardinatliy of $\boldsymbol{q}$

```
procedure QUERY(q, i, S, M)
    if i = 0 then
        return use grid to answer q
    i_l ← index of q[i][0] in S[i]
    i_u ← index of q[i][1] in S[i]
    if i_u = i_l then
        return M[i][i_l](q)
    q_u ← q
    q_u[i][0] ← S[i_u]
    q_l ← q
    q_l[i][1] ← S[i_l + 1]
    if i_u = i_l + 1 then
        return M[i][i_l](q_l) + M[i][i_u](q_u)
    q[i][0] ← S[i_l + 1]
    q[i][1] ← S[i_u]
    return QUERY(q, i − 1, S, M)+M[i][i_l](q_l) + M[i][i_u](q_u)
```

Cardinality of a query in the i-th dimension is the number of points the would query we only consider the i-th dimension.

Construction. Assume we would like to obtain accuracy ϵ. We build a grid and materialize the exact result in each cell. Then, for queries, where part of a query partially overlaps a cell, we also build models to answer queries. Thus, a query is decomposed into parts that fully contain cell and parts that don't which are answered by models.

Split the i-th dimension into $k = \frac{4d^2 n}{\epsilon^2}$ partitions, with each partition containing $(\frac{\epsilon}{2d})^2$ points. Let $S[i] = \{s_1^i, ..., s_k^i\}$ be the partition points in the i-th dimension, that is, for all j we have for $P[i,j] = \{\boldsymbol{p} \in \boldsymbol{D}, s_j^i \leq \boldsymbol{p}_i < s_{j+1}^i\}$, $|P[i,j]| = (\frac{\epsilon}{2d})^2$. Using theorem 4.3 to build a model for each set of points in P, we have that the expected error of each is $O(\frac{\epsilon}{2d})$. The models are stored in M, with $M[i][j]$ denoting the model corresponding to j-th partition in the i-th dimension. Now, to answer a query, we first decompose it into $2d + 1$ queries. $2d$ of the queries are answered by models, which reduce the original query to one that matches all facets of the grid cells. Then, the grid cells are used to answer the final query, and the answer is combined with the model estimates to find final query answer estimate.

This is presented in Alg. 4. The decomposition of the query is done by recursively moving the upper and lower facets of the query hyperretangle in the d-th dimensions to aligh with the grid cell. Thus, in the d-th dimensions, if the closest grid partition points, respectively larger and smaller than $q[d][0]$ and $q[d][1]$ (the lower and upper bound of the query in d-th dimension) are s_i and s_j, we decompose the query into three queries: q_1, q_2 and q_3, all the same as q but $q_1[d][1] = s_i$,

Algorithm 5 Learned Sorting

Require: An array A of length n to be sorted
Ensure: The sorted array

procedure SORT(A)
  if $n \leq \varkappa$ **then**
    return MERGESORT(A)
  $S \leftarrow$ random sample of A of size $\sqrt{n}$
  $S \leftarrow$ MERGESORT(S)
  $k \leftarrow n^{\frac{1}{8}}$
  $\hat{f} \leftarrow \mathcal{A}(S)$
  $B \leftarrow$ array of size k
  $B_{min}, B_{max} \leftarrow$ arrays tracking min/max $B[i]$ $\forall i$
  for i in n **do**
    $B[\lfloor \frac{k\hat{f}(A[i])}{n} \rfloor].append(A[i])$
    Update B_{min}, B_{max} for bucket $\lfloor \frac{k\hat{f}(A[i])}{n} \rfloor$
  for i **in** $k-2$ **do**
    if $B_{max}[i] > B_{min}[i + (2\varkappa + 1)]$ **then**
      return MERGESORT(A)
  for i **in** k **do**
    if $|B[i]| \geq (2\varkappa + 1)n^{\frac{4}{5}}$ **then**
      return MERGESORT(A)
    else
      $B[i] \leftarrow$ SORT($B[i]$)
  return MERGE(B) ▷ Alg. 6

$q_2[d][0] = s_j$ and $q_3[d][0] = s_i$, $q_3[d][1] = s_j$. Then, learned models are used to answer q_1 and q_2, while q_3 is further recursively decomposed along its $d-1$-th dimension (and after full decomposition is answered using the grid). Note that q_1 and q_2 can now be answered using models, because by grid construction, they fall in a part of the space with at most $(\frac{\epsilon}{2d})^2$ points.

Accuracy. Grid cells are exact, and, as discussed above each model is built on a dataset of size at most $(\frac{\epsilon}{2d})^2$, so that it will have expected error $O(\frac{\epsilon}{2d})$. Thus, combining the error of the $2d$ queries, the total model error is ϵ as desired.

Query Time and Space Complexity. There are $2d$ model calls, each model call costing $\mathcal{T}^{\mathfrak{X}}_{(\frac{\epsilon}{2d})^2}$. Furthermore, if the i-th dimension of the query covers k_i points, then total of at most $\frac{4d^2k_i}{\epsilon^2}$ partitions in the i-th dimension intersect the query, so that the total number of cells traversed will be $(\frac{4d^2}{\epsilon^2})^d \Pi_i k_i$. Thus, total query time is $O(\mathcal{T}^{\mathfrak{X}}_{(\frac{\epsilon}{2d})^2} + (\frac{4d^2}{\epsilon^2})^d \Pi_i k_i)$. Furthermore, the total cost of storing the models is $\frac{4d^2n}{\epsilon^2}\mathcal{S}^{\mathfrak{X}}_{(\frac{\epsilon}{2d})^2}$, and the cost of the grid is $(\frac{4d^2n}{\epsilon^2})^d$. Thus, total space complexity is $O(\frac{4d^2n}{\epsilon^2}\mathcal{S}^{\mathfrak{X}}_{(\frac{\epsilon}{2d})^2} + (\frac{4d^2n}{\epsilon^2})^d)$.

Algorithm 6 Merge Step

Require: An array of sorted buckets
Ensure: Buckets merged into a sorted array

procedure MERGE(B)
  $A_s \leftarrow$ empty array of size n
  $A_s[: len(B[1])] \leftarrow B[1]$
  $j \leftarrow len(B[1])$
  for $b \leftarrow 2$ to k **do** ▷ Merges $A_s[: j]$ with $B[b]$
    $j \leftarrow j + len(B[b])$ ▷ Iterator for A_s
    $i \leftarrow len(B[b])$ ▷ Iterator for $B[b]$
    while $i > 0$ **do**
      if $B[b][i] > A_s[j]$ **then**
        $A_s[j+i] \leftarrow B[b][i]$
        i--
      else
        $A_s[j+i] \leftarrow A_s[j]$
        j--
    $j \leftarrow j + len(B[b])$
  return A_s

D.7. Sorting

D.7.1. USING LEARNED MODEL (THEOREM 4.5)

Algorithm. The algorithm is presented in Alg. 5. A sample of the array is first created and a model is built using the sample. Then, using the model, the array is split into $n^{\frac{1}{8}}$ buckets, where we theoretically show, using such a number of buckets, based on the accuracy of the model and with high probability, merging the buckets can be done by in linear time (because there will be limited overlap between the buckets) and each bucket will not be too big. Indeed, we first make sure the two properties mentioned before hold (otherwise the algorithm quits and reverts to merge sort), and then proceed to merge the buckets.

Correctness. If all the created buckets are sorted, the merge step simply merges them and thus returns a sorted array correctly. At the base case, merge sort is used, so the buckets will be sorted correctly. Thus, using the invariant above, the algorithm is correct.

Time Complexity. Consider sorting an array $A \sim \chi$ of size n. We take a subset S, of size $\sqrt{n}$ from A without checking the elements to preserve the i.i.d assumption. We sort them and use the algorithm $\mathcal{A}$ to obtain a model $\hat{r}$. We have that

$$\mathbb{P}(\|\hat{r} - r_\chi\|_\infty \geq \sqrt{n}\epsilon_1) \leq \varkappa_1 e^{-\varkappa_2(\epsilon_1 n^{-\frac{1}{4}} - 1)^2}. \quad (19)$$

Note that

$$\|\hat{r} - r_A\|_\infty \leq \|\hat{r} - r_\chi\|_\infty + \|r_\chi - r_A\|,$$

And, by DWK,

$$\mathbb{P}(\|r_A - r_\chi\|_\infty \geq \epsilon_3) \leq 2e^{-2(\frac{\epsilon_3}{\sqrt{n}})^2}.$$

Set $\epsilon_1 = n^{\frac{1}{4}}(\sqrt{\frac{\log\log n}{\varkappa_2}} + 1)$ and $\epsilon_3 = \sqrt{\frac{n \log\log n}{2}}$, we

have

$$\mathbb{P}(\|\hat{r} - r_\chi\|_\infty \geq \sqrt{n}\epsilon_1 \text{ or } \|r_A - r_\chi\| \geq \epsilon_3) \leq \varkappa_1 e^{-\log\log n} + 2e^{-\log\log n} = \frac{\varkappa_1 + 2}{\log n}$$

Thus,

$$\mathbb{P}(\|\hat{r} - r_A\| \geq \sqrt{n}\epsilon_1 + \epsilon_3) \leq \frac{\varkappa_1 + 2}{\log n}$$

and thus, whenever $\log\log n \geq 2$ and for $\varkappa = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{\varkappa_2}}$

$$\mathbb{P}(\|\hat{r} - r_A\|_\infty \geq \varkappa n^{\frac{3}{4}}\sqrt{\log\log n}) \leq \frac{\varkappa_1 + 2}{\log n}.$$

To simplify, observe that $n^{3/4}\sqrt{\log\log n} \leq n^{4/5}$ for $n \geq e$ so that

$$\mathbb{P}(\|\hat{r} - r_A\|_\infty \geq \varkappa n^{\frac{4}{5}}) \leq \frac{\varkappa_1 + 2}{\log n}.$$

Recall that we use k buckets and consider the i-th bucket. The elements, x, assigned to it must have $\frac{i}{k} \leq \frac{1}{n}\hat{r}(x) < \frac{i+1}{k}$. Combining this with the above, we have that whenever $\|\hat{r} - r_A\|_\infty \leq \varkappa n^{\frac{4}{5}}$ holds, for the elements x in the i-th bucket, we must have $\frac{1}{n}\hat{r}(x) \geq \frac{i}{k} - \varkappa n^{-\frac{1}{5}}$ and that $r_A(x) \leq \frac{i+1}{k} + \varkappa n^{-\frac{1}{5}}$. Therefore, we must have $r_A(x) \in [\frac{in}{k} - \varkappa n^{\frac{4}{5}}, \frac{(i+1)n}{k} + \varkappa n^{\frac{4}{5}}]$. There are at most $\frac{n}{k} + 2\varkappa n^{\frac{4}{5}}$ elements in this set. Setting $k = n^{\frac{1}{5}}$, we have that whenever $\|\hat{r} - r_A\|_\infty \leq \varkappa n^{\frac{4}{5}}$ holds, all buckets will have at most $(2\varkappa + 1)n^{\frac{4}{5}}$ elements. Thus, the probability that a bucket will have more than $(2\varkappa + 1)n^{\frac{4}{5}}$ elements is at most $\frac{\varkappa_1+2}{\log n}$.

Furthermore, the largest element in the i-th bucket or before will have $r_A(x) < (i + 1 + \varkappa)n^{\frac{4}{5}}$ and the smallest element in the $i + 2\varkappa + 1$-th bucket or after will have $r_A(x) \geq (i + 1 + \varkappa)n^{\frac{4}{5}}$, so that the content of buckets up to i are less than the content of the buckets from $i + 2\varkappa + 1$ onwards. Thus, whenever $\|\hat{r} - r_A\|_\infty \leq \varkappa n^{\frac{4}{5}}$ holds, if all the buckets are sorted, then the algorithm takes at most $\sum_{i=1}^{k}\sum_{j=0}^{2\varkappa+1}|B_{i-j}|$ number of operations to merge the sorted array, which is $O(n)$, where $|B_i|$ is the number of elements in the i-th bucket.

Finally, let T_n be the expected number of operations it takes to sort an array with n elements i.i.d sampled from some a distribution learnable class. Recall that we do merge sort if $\|\hat{r} - r_A\|_\infty \leq \varkappa n^{\frac{4}{5}}$ does not hold, and recursively sort the array if it does. Thus, whenever we recursively sort an array, the array will have at most $(2\varkappa + 1)n^{\frac{4}{5}}$ i.i.d elements, distributed from a conditional distribution of the original distribution. Thus, we have

$$\begin{aligned} T(n) &\leq O(\mathcal{B}^{\mathfrak{X}}_{\sqrt{n}} + \mathcal{T}^{\mathfrak{X}}_{\sqrt{n}}n) + \mathbb{P}(\text{merge sort})n\log n + \mathbb{P}(\text{recursively sort})n^{\frac{1}{5}}T((2\varkappa+1)n^{\frac{4}{5}}) \\ &\leq O(\mathcal{B}^{\mathfrak{X}}_{\sqrt{n}} + \mathcal{T}^{\mathfrak{X}}_{\sqrt{n}}n) + \frac{\varkappa+2}{\log n}n\log n + n^{\frac{1}{5}}T((2\varkappa+1)n^{\frac{4}{5}}) \\ &= O(\mathcal{T}^{\mathfrak{X}}_{\sqrt{n}}n\log\log n + \sqrt{n}\mathcal{B}^{\mathfrak{X}}_{\sqrt{n}} + \sum_{i=0}^{\log\log n} n^{1-\frac{1}{2}(\frac{4}{5})^i}\mathcal{B}^{\mathfrak{X}}_{n^{\frac{1}{2}(\frac{4}{5})^i}}). \end{aligned}$$

Space Comlexity. First, observe that we only create one model at a time, so the maximum size used for modeling is $\mathcal{S}^{\mathfrak{X}}_{\sqrt{n}} + \sqrt{n}\log n$. Moreover, the depth of recursion is at most $O(\log\log n)$, and the overhead of storing B is dominated by the first of recursion, whose overhead is $O(n^{\frac{1}{8}}\log n + O(n\log n))$, giving overall space overhead of $O(\mathcal{S}^{\mathfrak{X}}_{\sqrt{n}} + n\log n)$

Algorithm 7 Sorting Using Distribution Model

Require: An unsorted array A of size n
Ensure: A sorted array

```
procedure SORT(A)
    if n ≤ 10 then
        return MERGESORT(A)
    A' ← new array, A'[i] initialized as linked list, ∀i
    for i ← 1 to n do
        i' ← ⌈r̂(A[i])⌉
        A'[i'].append(A[i])
    return MERGE(B)                    ▷ Alg. 6
```

D.7.2. Using data distribution (Theorem 4.6)

Alg. 7 shows how to sort an array given an approximate model of the data distribution $\hat{[r]}$. The algorithm is very similar to Alg. 7, but uses n buckets and merges each bucket using merge sort (and thus no recursive sorting of the buckets).

Correctness. The algorithm creates buckets, sorts them independently. The sort is done by merge sort so it is correct, and thus merging the sorted buckets creates a sorted array.

Running time Let $T(A)$ be the number of operations the algorithm performs on an array A, and let $\mathcal{T}(n) = \mathbb{E}_{A\sim\chi^n}[T(A)]$ be the expected run time of the algorithm on an input of size n.

First, assume we use $\lfloor nr_\chi(x)\rfloor$ to map an element x to a location in the array S. After the mapping, we study the expected time to sort the elements in $S[i:j]$ where $j = i+k$ and $k > 0$. Note that the probability that an element is mapped to location $[i:j]$, i.e., $\mathbb{P}_{x\sim\chi}(\frac{i}{n} \leq r_\chi(x) < \frac{j}{n})$, is

$\frac{k}{n}$. Let $N_{i:j} = |S[i:j]|$ be the number of elements mapped to $S[i:j]$. We have that

$$\mathbb{P}_{A\sim\chi^n}(N_{i:j} = z) = \mathbb{C}(n,z)(\frac{k}{n})^z(1-\frac{z}{n})^{n-z}.$$

Thus, we have

$$\begin{aligned}\mathbb{E}_{A\sim\chi^n}[N_{i:j}\log(N_{i:j})] &= \sum_{z=1}^{n}\mathbb{P}_{A\sim\chi^n}(N_{i:j}=z)[z\log(z)]\\ &= \sum_{z=1}^{n}\mathbb{C}(n,z)(\frac{k}{n})^z(1-\frac{k}{n})^{n-z}z\log z\\ &= \sum_{i=1}^{4ek}\mathbb{C}(n,i)(\frac{k}{n})^i(1-\frac{k}{n})^{n-i}i\log i\\ &\quad+\sum_{i=4ek}^{n}\mathbb{C}(n,i)(\frac{k}{n})^i(1-\frac{k}{n})^{n-i}i\log i\end{aligned}$$

For the first part of the summation, we have

$$\begin{aligned}&\sum_{i=1}^{4ek}\mathbb{C}(n,i)(\frac{k}{n})^i(1-\frac{k}{n})^{n-i}i\log i\\ &\qquad\le \log(4ek)\sum_{i=1}^{4ek}\mathbb{C}(n,i)(\frac{k}{n})^i(1-\frac{k}{n})^{n-i}i\\ &\qquad\le k\log(4ek).\end{aligned}$$

For the second part, we have

$$\begin{aligned}&\sum_{i=4ek}^{n}\mathbb{C}(n,i)(\frac{k}{n})^i(1-\frac{k}{n})^{n-i}i\log i\\ &\le \sum_{i=4ek}^{n}\frac{1}{\sqrt{i}}(\frac{en}{i})^i(\frac{k}{n})^i(1-\frac{k}{n})^{n-i}i\log i\\ &= \sum_{i=4ek}^{n}(\frac{ekn}{i(n-k)})^i(1-\frac{k}{n})^n\sqrt{i}\log i\\ &\le \sum_{i=4ek}^{n}(\frac{2ek}{i})^i(1-\frac{k}{n})^n\sqrt{i}\log i\\ &\le \sum_{i=4ek}^{n}(\frac{1}{2})^i i\\ &\le 2.\end{aligned}$$

So that

$$\mathbb{E}_{A\sim\chi^n}[N_{i:j}\log(N_{i:j})] \le (j-i)\log(4e(j-i))+2.$$

Now recall that we use $\lfloor\hat{r}\rfloor$ with error $\|\hat{r}-nr_\chi\|_\infty \le \epsilon$ to map the elements to an array S'. Thus, if, for any x, $\lfloor nr_\chi(x)\rfloor = j$, $\hat{r}\in\{j-\lfloor\epsilon\rfloor,...,j+\lceil\epsilon\rceil\}$. Let $\bar{N}_{i:j}$ be the number of elements mapped to positions $[i:j]$ using $\hat{r}$. Note that $\bar{N}_{i:j}\le N_{i-\epsilon:j+\epsilon}$. Thus, dividing S' into groups of ϵ and sorting each separately, we have that the total cost of sorting the groups is

$$\begin{aligned}&\sum_{j=0}^{\frac{n}{\epsilon}}\mathbb{E}_{A\sim\chi^n}[\bar{N}_{j\epsilon:(j+1)\epsilon}\log(\bar{N}_{j\epsilon:(j+1)\epsilon})]\\ &\le \sum_{j=0}^{\frac{n}{\epsilon}}\mathbb{E}_{A\sim\chi^n}[N_{(j-1)\epsilon:(j+2)\epsilon}\log(N_{(j-1)\epsilon:(j+2)\epsilon})]\\ &\le O(\epsilon\log\epsilon)\end{aligned}$$

Finally, note that r_χ is a non-decreasing function. Therefore, if $x\le y$, we have $\lfloor nr_\chi(x)\rfloor \le \lfloor nr_\chi(y)\rfloor$. Given that $\|\hat{r}-nr_\chi\|_\infty\le\epsilon$, we have that if $x\le y$, $\hat{r}(y)\ge\hat{r}(x)-2\epsilon$. Consequently, if x is mapped to the i-th group by $\hat{r}$, all elements in the j-th group with $j<i-2$ are less than x. This means, to merge the sorted groups, we start with the first group and iteratively merge the next group with the merged array so far. Performing each merge from the end of the two sorted arrays (as done in merging using learned data distribution), each merge will cost at most 3ϵ, so the total cost of merging all the $\frac{n}{\epsilon}$ sorted groups is $O(\frac{n}{\epsilon}\epsilon)=O(n)$. Putting everything together,r when each model call costs $\mathcal{T}_n^{\mathfrak{X}}$, we have that expected time complexity of the algorithm is $O(n\mathcal{T}_n^{\mathfrak{X}}+n\log\epsilon)$. Moverover the space overhead of the algorithm is $O(n\log n+\mathcal{S}_n^{\mathfrak{X}})$ where the $\log n$ factor is to keep a pointer to the elements of the array (instead of copying them).