Conformal Prediction for Verifiable Learned Query Optimization

Hanwen Liu
University of Southern California
Los Angeles, California, USA
hanwen_liu@usc.edu

Shashank Giridhara*
Amazon Web Services
Palo Alto, California, USA
smgiridh@amazon.com

Ibrahim Sabek
University of Southern California
Los Angeles, California, USA
sabek@usc.edu

ABSTRACT

Query optimization is critical in relational databases. Recently, numerous Learned Query Optimizers (LQOs) have been proposed, demonstrating superior performance over traditional hand-crafted query optimizers after short training periods. However, the opacity and instability of machine learning models have limited their practical applications. To address this issue, we are the first to formulate the LQO verification as a Conformal Prediction (CP) problem. We first construct a CP model and obtain user-controlled bounded ranges for the actual latency of LQO plans before execution. Then, we introduce CP-based runtime verification along with violation handling to ensure performance prior to execution. For both scenarios, we further extend our framework to handle distribution shifts in the dynamic environment using adaptive CP approaches. Finally, we present CP-guided plan search, which uses actual latency upper bounds from CP to heuristically guide query plan construction. We integrated our verification framework into three LQOs (Balsa, Lero, and RTOS) and conducted evaluations on several workloads. Experimental results demonstrate that our method is both accurate and efficient. Our CP-based approaches achieve tight upper bounds, reliably detect and handle violations. Adaptive CP maintains accurate confidence levels even in the presence of distribution shifts, and the CP-guided plan search improves both query plan quality (up to 9.84x) and planning time, with a reduction of up to 74.4% for a single query and 9.96% across all test queries from trained LQOs.

1 INTRODUCTION

A query optimizer is a performance-critical component in every database system. It translates declarative user queries into efficient execution plans [3, 45]. There have been numerous efforts to learn query optimizers (LQOs)(e.g., [18, 34, 35, 59]) to reduce the reliance

on manual tuning and expert intervention, and ultimately lead to more intelligent and responsive database systems. Unfortunately, LQOs suffer three main drawbacks. First, they can result in slow execution plans at the beginning of the learning process (sometimes orders of magnitude slower than the optimal plan [28]), where the probability of selecting disastrous plans is high. These disastrous plans at the beginning can slow the LQO's convergence to efficient query plans. Second, although LQOs can outperform traditional optimizers on average, they can perform catastrophically in the tail cases, especially when the training data is sparse [35]. Third, LQOs are normally trained for specific workload. Their performance degrades significantly when distribution shifts exist in the query workloads and the underlying data [35, 40, 49].

Given these drawbacks, verifying that the LQO's generated plans satisfy the critical *latency constraints* in real-life applications is crucial. Unfortunately, typical model checking techniques (e.g., [9, 11]) that have been successfully investigated to verify the properties of other database components, such as transaction management and concurrency control, fail when the search space to be explored grows drastically as in query optimizers. In addition, statistical variations of these techniques (e.g., [10]) do not perform verification during the runtime. Using these techniques, an LQO might be verified to be constraint-compliant a priori. However, during runtime, we may observe certain query plans that violate the constraints due to the unforeseen changes in the execution environment. In such case, these techniques should provide runtime verification. Additionally, these techniques should also be capable of verifying LQOs operating in dynamic environments.

Recently, Conformal Prediction (CP) [2, 55] has emerged as an efficient solution to perform runtime verification (e.g., [6, 56]) with *formal guarantees* (e.g., [8, 12, 29, 44]). In particular, CP is a rigorous statistical tool to quantify the uncertainty of the ML models' predictions while allowing users to specify the desired level of confidence in the quantification and being agnostic to the details of the ML models. CP-based runtime verification showed a great success in verifying cyber-physical systems such as autonomous cars [29], autonomous robots [41], and aircraft simulation [29, 44], among others. However, CP-based runtime verification was never explored in the context of database systems before.

In this paper, we present the first study of the LQO verification problem using CP. Specifically, we use CP to solve the LQO verification problem in three ways. First, we employ CP to provide user-controlled bounded ranges for the actual latency of constructed plans by LQOs even before executing them (e.g., verifying that an LQO plan for a specific query will never result in an execution time of more than 300 msec with a probability of at least 90%). Second, we go further and explore the use of CP to perform a runtime verification, with formal bounds, that can early detect any performance constraint violation during the LQO's plan construction process

based solely on the constructed partial plans so far and before the full plan is completed (e.g., with a user-defined confidence level of 95%, we can detect at the second step of building a query plan by the LQO that the eventual complete plan will fail to satisfy a specific latency constraint). This will help in planning how to handle such violations during the plan construction time and before execution (e.g., falling back to a traditional query optimizer for re-planning). For both scenarios, we introduce an adaptive CP framework to support LQOs in static cases (LQOs are trained and tested on the same workload) and in distribution shift cases (evaluating LQOs on different workloads). Additionally, we propose a CP-guided plan search algorithm that relies on upper bounds of the actual latency, instead of typical predicted costs by LQOs, to generate more optimal query plans within shorter time frames. We also provide rigorous theoretical proofs of our approaches to ensure correctness and frameworks that facilitate the integration of our CP-based verification approaches with LQOs in real-world environments.

Our experimental results on the JOB [28] and TPC-H [7] workloads confirm the correctness of latency bounds across multiple LQOs, including Balsa [59], Lero [64], and RTOS [62], all aligning with theoretical expectations. We then demonstrate the effectiveness of our adaptive CP framework under distribution shift by evaluating it on workloads transitioning to CEB [39] and JOBLight-train [24]. In runtime verification, we show that our CP-based methods accurately detect violations, and our violation handling reduces overall execution latency by 12,030.1 ms across 7 violating queries. Using the CP-guided algorithm, our approach improves plan quality in 33% of queries from a moderately trained LQO, achieving an additional 9.96% reduction in overall planning latency across all test queries. For well-trained LQOs, we observe better plan quality and faster query planning with our CP-guided plan search algorithm. These comprehensive experiments substantiate the correctness and effectiveness of our CP-based verification frameworks.

In summary, our novel contributions are as follows:

- We are the first to formulate the Learned Query Optimizer (LQO) verification as a Conformal Prediction (CP) problem.
- We develop CP-based latency bounds for LQOs, with formal proofs, to provide a user-defined confidence level a bounded range for the actual latency of query plans.
- We design CP-based runtime verification, with formal bounds, which detect and address long-latency query plans even before completing the plan construction.
- We propose an Adaptive CP framework for LQOs which aids in handling distribution shifts, enhancing the robustness of the verification framework for real-world scenarios.
- We introduce a generic CP-guided plan search algorithm that can enhance both the query plan quality and the planning time from a trained LQO.
- Our experimental evaluation using the proposed CP-based verification frameworks, across three LQOs and four workloads, demonstrates the correctness and effectiveness of our CP-based frameworks for LQOs.

We believe that our proposed CP-based verification approaches hold promising potential for future applications across other learned components in database systems.
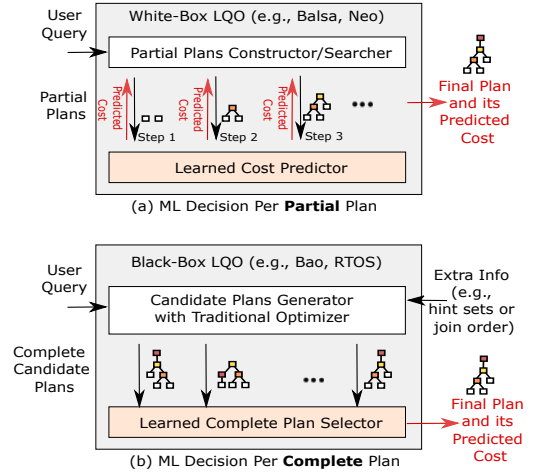

Figure 1: ML Decisions in Learned Query Optimizers (LQOs).

2 BACKGROUND

In this section, we first discuss the granularity levels of prediction decisions to be verified in learned query optimizers (Section 2.1). Then, we provide a brief introduction for the Conformal Prediction (Section 2.2) and Signal Temporal Logic (Section 2.3) tools that are used to build our verification framework and formally represent the performance constraints we verify LQOs against, respectively.

2.1 Granularity Levels of Decisions to be Verified in Learned Query Optimizers

While Learned Query Optimizers (LQOs) (e.g., [18, 34, 35, 59, 62, 64]) can improve the performance over traditional optimizers by adapting to complex queries and data distributions, their reliance on ML models to take decisions introduces variability and potential unpredictability in performance. Therefore, verifying LQOs against user-defined performance constraints is crucial to ensure that generated plans meet specific efficiency and reliability standards (e.g., the execution time of a specific query should be $\leq$ 100ms). Broadly, LQOs fall into three categories based on how ML is used. The first category uses ML to improve specific components of the optimizer (e.g., cardinality estimator [25, 49, 60] and cost estimator [37, 48]). The second category uses ML to construct the query plan from scratch, replacing the traditional optimizer (e.g., [35, 59]). The third category uses ML to steer the traditional optimizer in constructing better candidate plans and/or in selecting among them (e.g., [34, 62, 64]). In this paper, we focus on verifying the ML decisions made by LQOs in the second and third categories only, where ML is involved in constructing the query plan itself. However, the granularity level of these decisions differs between these two categories. Figure 1 shows a high-level overview of these two LQO categories, highlighting their ML decisions in red. In the second category, fine-grained prediction decisions are performed to construct the query plan step-by-step and predict the associated cost at each step[1]. For instance, Balsa [59] uses a learned value model to construct the optimized

---

[1]In this paper, we assume that the cost of a plan is indicative of its actual latency, where a higher cost corresponds to longer latency.

plan operator-by-operator and predict the intermediate cost for the final plan construction at each operator. We refer to the second category as *white-box* LQOs because we rely on these fine-grained prediction decisions during the verification process. In contrast, in the third category, learned models neither perform step-by-step plan construction nor intermediate cost predictions. Instead, these models are used to select the best plan from a set of candidate plans, either by predicting the high-level cost for each candidate [34] or by assigning a relative rank to all candidates [62]. These candidate plans are typically constructed by a traditional optimizer and based on auxiliary information, such as join orders [62] and hint sets [34]. Therefore, in this category, the selection decisions are mainly only on the level of the whole plan and its high-level associated cost, if available. We refer to the third category as *black-box* LQOs because we only access coarse-grained plan-level decisions (i.e., no partial-plan-level predictions) during the verification process.

2.2 Standard Conformal Prediction (CP)

We build our LQO verification framework, as shown later, based on Conformal Prediction (CP) [2, 55], a rigorous statistical tool that efficiently quantifies the uncertainty of the ML models' predictions. CP enables users to specify the desired level of confidence in the quantification while being agnostic to the details of the ML models. To introduce CP, assume that $R^{(0)}, R^{(1)}, \ldots, R^{(K)}$ are $K + 1$ independent and identically distributed (i.i.d) random variables, where each variable $R^{(i)}$ for $i \in \{0, \ldots, K\}$ is an estimate of the prediction error between the true output $y^{(i)}$, i.e., ground truth, for input $x^{(i)}$ and the predicted value of this output $\eta(x^{(i)})$ by the ML predictor η. Formally, this error can be expressed as:

$$R^{(i)} = \|y^{(i)} - \eta(x^{(i)})\|$$

where $\|\cdot\|$ denoting the absolute value. $R^{(i)}$ is commonly referred to as the *non-conformity score*, where a small score suggests a strong predictive model and a large score indicates poorer performance (i.e., less accurate predictions).

Now, assuming that $R^{(0)}$ belongs to test data and $R^{(1)}, \ldots, R^{(K)}$ are calibration data, the objective of CP is to quantify the uncertainty of $R^{(0)}$ using $R^{(1)}, \ldots, R^{(K)}$. Specifically, for a user-defined uncertainty probability $\delta \in [0, 1]$ (i.e., $1 - \delta$ is a confidence level), CP aims to compute an upper bound $C(R^{(1)}, \ldots, R^{(K)})$ for the prediction error $R^{(0)}$ such that:

$$\text{Prob}(R^{(0)} \leq C(R^{(1)}, \ldots, R^{(K)})) \geq 1 - \delta \quad (1)$$

This upper bound $C(R^{(1)}, \ldots, R^{(K)})$ can be efficiently determined by computing the $(1 - \delta)$th quantile of the empirical distribution of $R^{(1)}, \ldots, R^{(K)}$ and ∞, assuming training, calibration, and testing data originate from the same underlying distribution (i.e., the scores $R^{(0)}, R^{(1)}, \ldots, R^{(K)}$ are exchangeable) [2]. Although this assumption aligns with the data and workload scenarios used in most state-of-the-art workload-aware LQOs (e.g., [34, 35, 59]), we extend our LQO verification framework to support adaptive CP for distribution shifts [63] as shown later in Section 3.2. For simplicity, we refer to the upper bound $C(R^{(1)}, \ldots, R^{(K)})$ as C in the rest of the paper. Note that CP guarantees marginal coverage, which is not conditional on the calibration data [2].

2.3 Formal Representation of Performance Constraints to be Verified with CP

To formally represent the desired performance constraints to verify against LQOs, we employ Signal Temporal Logic (STL) [17], a CP-compliant formal logical language for verification. STL was originally introduced to verify the properties of time series data (e.g., signals), especially in the context of cyber-physical systems [33]. STL can also handle non-traditional time-series data where sequence or order matters. An STL specification ϕ is recursively defined as $\phi = True \mid \mu \mid \neg\phi \mid \phi \wedge \psi \mid \mathbf{G}_{[a,b]}\phi$, where ψ is an STL formula. $\neg$ and $\wedge$ are the *not* and *conjunction* operators, respectively. The *always* operator $\mathbf{G}_{[a,b]}\phi$ encodes that ϕ has to be always true for the entire duration or steps between a and b. μ is a predicate to check whether the semantics of the specification ϕ are achieved or not, i.e., $\mu : \mathbb{R}^n \to \{True, False\}$. For instance, we can define an operator $\mathbf{G}_{[0,N-1]}\phi$ to check whether the query plan generated by a LQO will *always* have a latency less than 750 msec at each of its N execution steps (i.e., partial plans). In this case, $x = (x_0, x_1, \ldots, x_{N-1})$ will represent the partial plan latencies at steps $0, 1, \ldots, N - 1$ and the condition $x_\tau < 750$ forms the semantics of the specification ϕ that needs to be checked at each step τ.

Moreover, we can use *robust* semantics $\rho^\phi(x)$, as in [17, 19], to extend the binary evaluation of STL satisfaction (i.e., $\mu(x)$) by providing a quantitative measure of the degree to which this satisfaction is achieved. Unlike traditional binary satisfaction, robust semantics $\rho^\phi(x)$ produces a real-valued metric: positive values indicate that the specification ϕ is satisfied, with the magnitude representing the strength of satisfaction, whereas negative values denote a *violation*, with the magnitude reflecting the severity of the violation. For example, considering the previously discussed specification ϕ with condition $x_\tau < 750$, the robust satisfaction $\rho^\phi(x)$ can be defined to provide a quantitative measure of how robustly all latencies x satisfy this condition by calculating $(750 - x_\tau)$ for each $x_\tau \in x$. In this case, $x_\tau = 100$ exhibits stronger robustness in satisfying ϕ than $x_\tau = 600$, whereas $x_\tau = 800$ results in a *violation*. More details about robust STL semantics are in [17, 19].

3 CP-BASED LATENCY BOUNDS FOR LQOS

As mentioned in Section 2.1, we focus on two categories of LQOs: *white-box* and *black-box*, both of which use learned models to construct the query plan itself. In *white-box* LQOs (e.g., [35, 59]), the learned model builds the query plan incrementally, constructing one partial plan at a time based on a predicted cost (Figure 1 (a)). Here, we employ CP to obtain *user-controlled bounded* ranges for the *actual latency* (not the predicted cost) of these constructed partial plans *before executing them*. For example, given a partial plan s and a user-defined confidence level (i.e., $1 - \delta$) of 90%, we can determine a latency range $[l_{min}^s, l_{max}^s]$ that the latency l^s of s will fall within with *at least* 90% probability, where l_{min}^s and l_{max}^s represent the lower and upper latency bounds, respectively. The intuition is to leverage CP to gain insights into the relationship between predicted costs and actual latencies of partial plans from the LQO's calibration query workloads, and then use these insights to obtain latency ranges for testing queries.

Similarly, in *black-box* LQOs (e.g., [34, 62]), we use CP to provide such user-controlled bounded ranges, yet for the end-to-end

latencies of complete plans rather than partial ones. This is because *black-box* LQOs rely on learned models solely to select the best plan among complete candidates (Figure 1 (b)).

Latency-Cost Non-conformity Score. A critical step in applying CP is defining the non-conformity score R (check Section 2.2), as it quantifies the deviation between the predicted and actual outcomes. In the LQO context, we focus on how the actual latency of a plan, whether partial or complete, deviates from its predicted cost[2]. Following the CP notation, we formally define a *latency-cost non-conformity score* $R^{(i)}$ for the plan at step τ in a query q_j to be:

$$R^{(i)} = \|t_\tau^{(j)} - \hat{c}_\tau^{(j)}\| \tag{2}$$

where $t_\tau^{(j)}$ is the actual latency of this plan and $\hat{c}_\tau^{(j)}$ is its predicted cost. Note that $R^{(i)}$ represents a score for a calibration plan (i.e., $R^{(i)} \in \{R^{(1)}, \ldots, R^{(K)}\}$) when q_j belongs to the calibration workload Q^{Cal} and represents a score for a testing plan $R^{(0)}$ when q_j belongs to the testing workload Q^{Tst}.

In the following, we introduce our approach for using CP to obtain the bounded latency ranges when the calibration and testing distributions are similar, i.e., static case, (Section 3.1), and then we extend it to handle distribution shifts in the testing distribution, i.e., distribution shift case (Section 3.2). Finally, we detail our proposed verification framework (Section 3.3).

3.1 Latency Bounds in Static Cases

Using equations 1 and 2, we can directly derive an upper bound C on the latency of any plan, whether partial or complete, in a testing query as the $(1-\delta)$th quantile of the latency-cost non-conformity scores such that:

$$P(\|t_\tau^{(j)} - \hat{c}_\tau^{(j)}\| \le C) \ge 1 - \delta \tag{3}$$

By reformulating Equation 3, we can compute a range for the actual latency $t_\tau^{(j)}$ of this plan, with confidence $1-\delta$, based on its predicted cost $\hat{c}_\tau^{(j)}$ and the upper bound C as follows:

$$P(\hat{c}_\tau^{(j)} - C \le t_\tau^{(j)} \le \hat{c}_\tau^{(j)} + C) \ge 1 - \delta \tag{4}$$

This allows us to estimate a *bounded* range for the actual latency even prior to executing the plan. However, the tightness of this range primarily depends on the upper bound C, which itself is influenced by the number of calibration plans used to establish it. Therefore, determining the sufficient number of calibration plans to construct a valid upper bound C is crucial. Here, we derive a lower bound on this number.

Lemma 1 (Lower Bound on Required Calibration Plans). *Let the latency-cost non-conformity scores of a testing plan $R^{(0)}$ and K calibration plans $R^{(1)}, \ldots, R^{(K)}$ be exchangeable and realizing i.i.d random variables, $\delta \in [0, 1]$ be a user-defined uncertainty probability, and C be an upper bound on the score $R^{(0)}$ of the testing plan, calculated at a confidence level of $1 - \delta$. Then, the lower bound on the number of calibration plans, i.e., K, to calculate C is $\frac{1-\delta}{\delta}$.*

Proof. If the scores $R^{(0)}, R^{(1)}, \ldots, R^{(K)}$ are exchangeable (i.e., independent of their order and are drawn from the same distribution),

---

[2]Note that, during calibration, we can obtain the actual latency of any partial or complete plan straightforwardly with tools like EXPLAIN ANALYZE in PostgreSQL [43].

then the joint distribution of these scores remains unchanged [2]. This means that the rank of any score, including $R^{(0)}$, is *uniformly* distributed on the ranks $\{1, \ldots, K + 1\}$. As a result, we can estimate the probability of the $R^{(0)}$'s rank in this uniform distribution using the $(1 - \delta)$th quantile as follows:

$$\text{Prob}(\text{Rank of } R^{(0)} \le \lceil (K + 1)(1 - \delta) \rceil) \ge 1 - \delta$$

where $\lceil \cdot \rceil$ denoting the ceiling function. However, according to [2], if $\lceil (K + 1)(1 - \delta) \rceil > K$, then the upper bound C becomes trivial and uninformative, yielding $C = \infty$. Therefore, to ensure that C is nontrivial, we need the following condition:

$$\lceil (K + 1)(1 - \delta) \rceil \le K$$

From this, we can easily get $K \ge \frac{1-\delta}{\delta}$, which means the lower bound on the number of calibration plans should be $\frac{1-\delta}{\delta}$.

3.2 Latency Bounds in Distribution Shift Cases

In the previous section, we assumed that the test data $\{R^{(0)}\}$ and the calibration data $\{R^{(1)}, \ldots, R^{(K)}\}$ are drawn from the same underlying distribution. However, this assumption does not hold in workload drift scenarios, i.e., new or evolving workloads, that are common in database applications [40, 57, 58]. For instance, slight changes in query patterns (e.g., filters on new columns), can violate the exchangeability assumption of $R^{(0)}, R^{(1)}, \ldots, R^{(K)}$ (see Section 2.2), leading to an invalid upper bound C. To address this, we adopt an adaptive CP variation, inspired by [38], which dynamically adjusts the upper bound to be $\tilde{C}$ based on the distribution shift in the testing workload only, assuming that this shift can be empirically estimated. This approach ensures that the newly calculated bounded latency range, based on $\tilde{C}$, preserves the user-specified confidence level $1 - \delta$, even in the presence of distribution shifts.

Specifically, let $\mathcal{D}$ represent the distribution of the testing workload (i.e., $R^{(0)} \sim \mathcal{D}$) and $\mathcal{D}_0$ represent the distribution of the calibration workload (i.e., $R^{(1)}, \ldots, R^{(K)} \sim \mathcal{D}_0$). We can rigorously quantify the deviation between the calibration and test distributions using the total variation distance $TV(\mathcal{D}, \mathcal{D}_0) = \frac{1}{2} \int_x |P(x) - Q(x)| dx$, where $P(x)$ and $Q(x)$ denote the probability density functions (PDFs) of $\mathcal{D}$ and $\mathcal{D}_0$, respectively [16]. To realize this in our LQO context, we empirically estimate these PDFs of latency-cost non-conformity scores using kernel density estimators (KDEs) as Gaussian kernels. According to [38, 63], we can compute an adjusted uncertainty probability $\tilde{\delta}$ to account for the distribution shift from $\mathcal{D}_0$ to $\mathcal{D}$ as follows:

$$\tilde{\delta} = 1 - g^{-1}\left(g\left(\left(1 + \frac{1}{K}\right)g^{-1}(1 - \delta)\right)\right) \tag{5}$$

where δ is the original user-specified uncertainty probability, K is the number of calibration plans, and $g(\beta) = \max(0, \beta - \epsilon)$ and its inverse $g^{-1}(\beta) = \min(1, \beta + \epsilon)$ are two functions calculated based on the allowable distribution shift ϵ, which must be set to a value greater than or equal to $TV(\mathcal{D}, \mathcal{D}_0)$.

Then, similar to Equation 4, the new latency bounds are calculated as:

$$P(\hat{c}_\tau^{(j)} - \tilde{C} \le t_\tau^{(j)} \le \hat{c}_\tau^{(j)} + \tilde{C}) \ge 1 - \delta \tag{6}$$

where $\tilde{C}$ is the $(1-\tilde{\delta})$th quantile of the latency-cost non-conformity scores from the original calibration workload $Q^{Cal} \sim \mathcal{D}_0$.
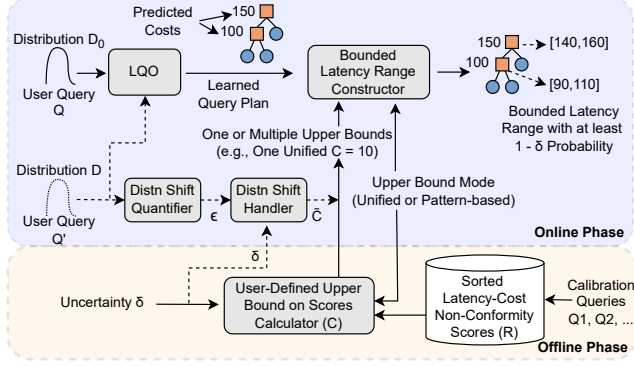
Figure 2: CP-based Bounded Latency Range Framework.

3.3 Framework Overview

Figure 2 gives an overview of our CP-based framework to provide bounded latency ranges before execution.

Offline Phase. After training the LQO, we first construct a set of latency-cost non-conformity scores using all plans - whether partial or complete - from the calibration query workload Q^{Cal}. For each plan, we collect its predicted cost during the LQO's planning phase and its actual latency from execution. These scores are then sorted in ascending order and stored to be used along with the user-specified uncertainty probability δ to compute any upper bound, whether C in the static case or $\tilde{C}$ in the distribution shift case.

Online Phase. The user first submits a testing query to the trained LQO, which generates a query plan with predicted costs (either per partial plan for white-box LQOs or a single cost for the entire plan in black-box LQO). In case there is a distribution shift in the testing queries from $\mathcal{D}_0$ to $\mathcal{D}$, queries are also sent (represented by a dashed line) to a distribution shift quantifier to determine the allowable distribution shift ϵ (check Section 3.2). This value, along with the user-defined parameter δ, is then used to construct the adjusted upper bound $\tilde{C}$. Hereafter, we use C to denote the upper bound for both the static and distribution shift cases since they are applied identically in subsequent steps. We support two modes for calculating the upper bound, namely *Unified* and *Pattern-based*, depending on the desired granularity level. In the *Unified* mode, non-conformity scores from all partial and complete plans are treated equally to construct a single upper bound value for C, applicable to both partial and complete plans of the testing query[3]. In the *Pattern-based* mode, we account for the internal structure of partial plans by setting a unique C value for each parent-child pattern. This C value is applied only when that pattern appears in the testing query. Note that pattern-based upper bounds are available only for white-box LQOs and are effective if we have sufficient calibration scores for each pattern (i.e., meeting the lower bound K in Lemma 1 for each pattern). Otherwise, the unified upper bound is preferable. Algorithm 1 illustrates how to construct the two types of upper bounds given a specific user-defined uncertainty probability δ. Once the upper bound(s) construction is done, the query plan along
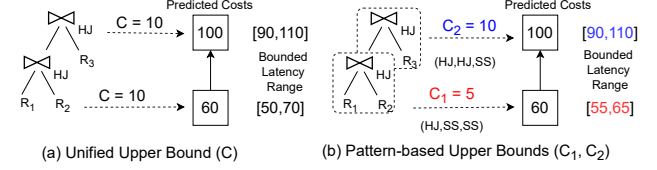

Figure 3: Bounded Latency Ranges with Different Types of Upper Bound (C).

with the upper bound(s) are passed to the bounded latency range constructor to obtain the bounded ranges as in Equation 4.

Figure 3 shows an example of using both unified and pattern-based upper bounds to calculate the bounded latency ranges for one testing query plan. Here, we assume a white-box LQO that constructs the plan from the bottom up. Initially, it constructs a Hash Join (HJ) at the first level, with Sequential Scan (SS) operations as left and right children. This parent-children pattern is labeled as (HJ, SS, SS) [4]. Similarly, the partial plan in the second level has the (HJ, HJ, SS) pattern. In this example, the LQO predicts 60 and 100 as costs for these two partial plans. In the case of using unified upper bound (Figure 3 (a)), we use a single value $C = 10$, resulting in latency ranges of [50, 70] and [90, 110] for the first and second partial plans, respectively. In the case of using pattern-based upper bounds (Figure 3 (b)), two different values $C_1 = 5$ and $C_2 = 10$ are used, resulting in latency ranges of [55, 65] and [90, 110] for the (HJ, SS, SS) and (HJ, SS, HJ) patterns, respectively.

4 CP-BASED RUNTIME VERIFICATION FOR WHITE-BOX LQOS PLAN CONSTRUCTION

Earlier (Section 3), we showed how CP can provide a bounded latency range for partial or complete query plans, helping to assess the uncertainty of LQO decisions before execution. Here, we aim to go further by exploring the use of CP to early detect any performance constraint violations during the plan construction process of *white-box* LQOs (e.g., [35, 59]), based solely on the constructed partial plans so far and before the full plan is completed.

Suppose $\mathcal{D}$ is an unknown distribution over the query plans generated by a white-box LQO. Let $X = (X_0, X_1, \dots) \sim \mathcal{D}$ represent a random query plan generated by the LQO, where X_τ is a random variable denoting the state of the generated partial plan at step τ (e.g., predicted cost or actual latency). Then, we can formally define the white-box LQO runtime verification problem as follows:

Definition 1 (The White-Box LQO Runtime Verification Problem). *Assuming a white-box LQO (e.g., [59]) and a testing query q that this LQO already finished constructing its partial plans till step τ and is still running, we aim to <u>verify</u> whether all generated partial plans by this LQO (past and future) result in a complete plan, represented by X, that satisfies a user-defined STL-based performance constraint ϕ with a confidence level $1 - \delta$, i.e., $Prob(X \models \phi) \geq 1 - \delta$, where $\delta \in [0, 1]$ is a constraint violation probability.*

---

[3]Note that unified upper bounds can be calculated for white-box and black-box LQOs

[4]We assume that the roles of the left and right child operators are not interchangeable. Therefore, the order of children in any pattern is important, i.e., (HJ, SS, HJ) and (HJ, HJ, SS) are different patterns.

Algorithm 1 Constructing a List of Upper Bound(s) C on the Latency-Cost Non-Conformity Scores

Require: List of sorted latency-cost non-conformity scores R, Uncertainty probability $\delta \in [0, 1]$, Upper bound type $T \in \{$Unified, Pattern-based$\}$
Ensure: List of upper bound(s) C
1: $K \leftarrow$ length of sorted R
2: **if** $K < \frac{1-\delta}{\delta}$ **then**   ▷ Calibration scores are not enough
3:     Get more R scores from calibration
4: **else**
5:     **if** T is Unified **then**
6:         $p \leftarrow \lceil (K+1)(1-\delta) \rceil - 1$ ▷ $(1-\delta)$th quantile index
7:         $\mathcal{U} \leftarrow R^{(p)}$ ▷ $(1-\delta)$th quantile of R
8:         C $\leftarrow \{\mathcal{U}\}$ ▷ List has one unified upper bound
9:     **else**   ▷ Upper bounds will be calculated based on patterns
10:         **for all** parent-children patterns in calibration **do**
11:             $R_{patt} \leftarrow$ List of sorted R of the current pattern
12:             $n_{patt} \leftarrow$ Size of R_{patt}
13:             **if** $n_{patt} < \frac{1-\delta}{\delta}$ **then**
14:                 Get more R_{patt} scores from calibration
15:             **else**
16:                 $p_{patt} \leftarrow \lceil (n_{patt}+1)(1-\delta) \rceil - 1$
17:                 $\mathcal{U}_{patt} \leftarrow R^{(p_{patt})}$
18:                 C $\leftarrow$ C $\cup \{\mathcal{U}_{patt}\}$
19:             **end if**
20:         **end for**
21:     **end if**
22:     **return** C
23: **end if**

Let $x = (x_0, x_1, \ldots)$ be the realization of $X = (X_0, X_1, \ldots)$, where $x_{\text{obs}} = (x_0, \ldots, x_\tau)$ represents the constructed partial plans till step τ and $x_{\text{un}} = (x_{\tau+1}, x_{\tau+2}, \ldots)$ represents the future unknown partial plans that will be predicted. Since existing *white-box* LQOs (e.g., [35, 59]) predict one partial plan at a time, then we can estimate the realization x at step τ, with its constructed plans so far (i.e., x_{obs}) and next prediction at step $\tau + 1$ as follows:

$$\hat{x} = (x_{\text{obs}}, \hat{x}_{\tau+1|\tau}) \tag{7}$$

As described in Definition 1, our goal is to verify the quality of the white-box LQO's complete query plan, represented by X, against a user-defined STL specification ϕ. We can use robust semantics $\rho^\phi(.)$ (check Section 2.3) to achieve that. First, we define $\rho^\phi(X)$ to indicate how robustly the specification ϕ is satisfied with the complete query plan, and $\rho^\phi(\hat{x})$ to denote the estimate of this robustness we obtained so far based on the observations x_{obs} and the prediction $\hat{x}_{\tau+1|\tau}$. Then, according to [12, 29], we can use CP (Equation 1) to define an upper bound C on the difference between the actual robustness $\rho^\phi(X)$ of the complete query and the estimate of this robustness $\rho^\phi(\hat{x})$ till step τ such that:

$$\text{Prob}(\rho^\phi(\hat{x}) - \rho^\phi(X) \leq C) \geq 1 - \delta \tag{8}$$

This upper bound[5] can be easily obtained from the calibration query workload Q^{Cal} by calculating the following non-conformity

---

[5]Note that upper bounds on the robustness values can be adjusted in the distribution shift cases using the same approach in Section 3.2.
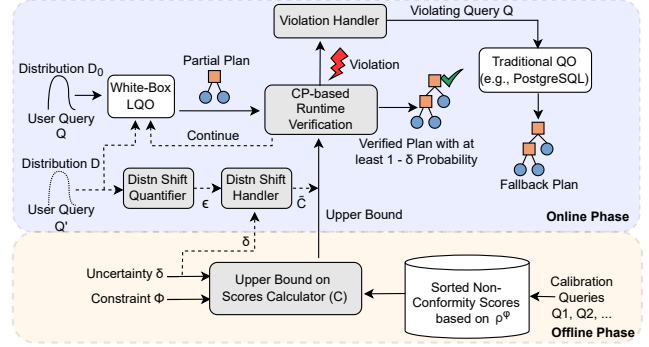

Figure 4: CP-based Runtime Verification Framework.

score $R^{(i)}$ for each partial plan in each calibration query $q_i \in Q^{Cal}$:

$$R^{(i)} = \rho^\phi(\hat{x}^{(i)}) - \rho^\phi(x^{(i)}) \tag{9}$$

where $x^{(i)}$ is the realization of X for query q_i (i.e., actual latencies and predicted costs for all partial plans in q_i) and $\hat{x}^{(i)}$ is the estimate of this realization till step τ only (i.e., $\hat{x}^{(i)} = (x_{\text{obs}}^{(i)}, \hat{x}_{\tau+1|\tau}^{(i)})$). Given that, we can define the following condition to verify whether the LQO satisfies ϕ or not.

Lemma 2 (The White-Box LQO Runtime Verification Condition). *Given a testing query q that uses LQO to generate its plan, represented by X, and with $\hat{x} = (x_{obs}, \hat{x}_{\tau+1|\tau})$ realizing the constructed and predicted partial plans at step τ, an STL constraint ϕ, a robust semantics measure $\rho^\phi(.)$ for this ϕ constraint, and a constraint violation probability $\delta \in [0, 1]$. Then, we can guarantee that these constructed and predicted partial plans $\hat{x}$ so far will result in a complete plan that satisfies the constraint ϕ with a confidence level $1 - \delta$, i.e., $Prob(X \models \phi) \geq 1 - \delta$, only if the robust semantics defined over these partial plans $\rho^\phi(\hat{x})$ is larger than C, where C is an upper bound calculated at a confidence level $1 - \delta$ and using equations 8 and 9. Otherwise, the resulting complete plan will cause a violation.*

Proof. By reformulating Equation 8, we can obtain:

$$P(\rho^\phi(X) \geq \rho^\phi(\hat{x}) - C) \geq 1 - \delta \tag{10}$$

If $\rho^\phi(\hat{x}) > C$, it implies:

$$P(\rho^\phi(X) > 0) \geq 1 - \delta \tag{11}$$

which, according to Section 2.3, further implies:

$$P(X \models \phi) \geq 1 - \delta \tag{12}$$

because $\rho^\phi(X) > 0$ directly implies that $X \models \phi$. Note that changing the constraint specification ϕ and/or the robust semantics measure $\rho^\phi(.)$ does not require retraining the white-box LQO to obtain valid guarantees because its prediction decisions, i.e., partial plans, are agnostic to any constraint specification.

4.1 Framework Overview

Figure 4 presents an overview of our CP-based runtime verification framework, which detects violations of user-defined performance constraints ϕ in the plans being constructed by white-box LQOs.

Offline Phase. Similar to our bounded latency range framework (Section 3.3), we start by constructing and sorting a set of non-conformity scores, obtained from the calibration queries and their partial plans. However, instead of constructing latency-cost based scores (Equation 2), we compute scores based on the difference between the actual robustness $\rho^\phi(X)$ of queries and their estimated robustness $\rho^\phi(\hat{x})$ at each partial plan step, assessing compliance with constraint ϕ using robustness measure $\rho^\phi(.)$ (Equation 9). These scores are then sorted and used to compute any upper bound, whether C in the static case or $\tilde{C}$ in the distribution shift case, at a user-defined confidence level $1 - \delta$ (Equation 8).

Online Phase. When a user submits a testing query, the white-box LQO starts to incrementally build the plan, adding one partial plan at a time. At each step τ, the runtime verification module uses the upper bound C and the estimated robustness $\rho^\phi(\hat{x})$ (representing all partial plans constructed up to τ and the expected one at $\tau + 1$) to check if $\rho^\phi(\hat{x}) > C$ (Lemma 2). If this condition holds, the LQO proceeds to construct the next partial plan at step $\tau + 1$. Otherwise, a violation is detected (e.g., exceeding a latency threshold). As a result, the violation handler discards the current plan under construction and sends the query to be re-planned by a traditional query optimizer (e.g., PostgreSQL [15]). This has been shown to be an effective solution, as highlighted in earlier works (e.g., [34]) and confirmed by our experimental evaluation (Section 6). The intuition here is that re-planning the query with a traditional optimizer and running it with the resulting average-performance plan incurs less overhead than executing a worst-case LQO-generated plan. Note that in case there is a distribution shift in the testing queries from $\mathcal{D}_0$ to $\mathcal{D}$, we construct the adjusted upper bound $\tilde{C}$ as in the online phase of our bounded latency range framework (Section 3.3).

5 CP-GUIDED PLAN SEARCH IN WHITE-BOX LQOS

In this section, we provide a simple yet effective approach for using CP to steer the decision-making process in white-box LQOs. Unlike sections 3 and 4, which focused on using CP to obtain bounded latency ranges for generated plans or to detect violations during the plan construction process (triggering a fallback to traditional optimizers), this section presents a CP-guided plan search algorithm designed to improve the quality of generated plans rather than just verifying them. Specifically, this algorithm utilizes CP-derived upper bounds on the actual latency of partial plans (Equation 4), to heuristically guide the plan search space navigation.

Intuition. White-box LQOs, such as Balsa [59] and Neo [35], use learned cost predictors to search over the space of partial plans, aiming to identify the plan with the lowest predicted cost. However, since the space of all partial plans at any step is far too large to exhaustively search, these LQOs typically find this plan heuristically by sorting predicted costs and then selecting the plan with the lowest cost. However, relying on the predicted costs may lead to sub-optimal plans if these predicted costs do not closely align with the actual latencies. To address this, we propose leveraging the CP-bounded upper bounds, which were discussed in Section 3, to guide the search for optimal partial plans at each step.

CP-Guided Plan Search Algorithm. Recall that for any partial plan at step τ, we can compute an upper bound U_τ on its actual

---

Algorithm 2 CP-Guided Plan Search

Require: Learned cost predictor LCP, Pattern-based upper bounds $\mathbf{C} = \{C_1, C_2, \ldots\}$ from Algorithm 1, Number of candidate complete plans n, Basic plan search algorithm BPS.
Ensure: Top-ranked plan final
1: queue ← Partial plans initialized with scans over relations
2: complete_plans ← []
3: **while** len(complete_plans) < n **and** queue is not empty **do**
4:     (state, $\hat{c}_{\text{state}}$) ← BPS.select_next_plan(queue)
5:     **if** state is a complete plan **then**
6:         complete_plans.add(state)
7:         **continue**
8:     **end if**
9:     List of (stateNew, $\hat{c}_{\text{stateNew}}$) ← Explore(LCP, state)
10:     **for all** pair in List of (stateNew, $\hat{c}_{\text{stateNew}}$) **do**
11:         U_{stateNew} ← LatencyUpperBound (stateNew, $\hat{c}_{\text{stateNew}}$, $\mathbf{C}$)
12:         BPS.insert_plan(queue, stateNew, $\hat{c}_{\text{stateNew}}$, U_{stateNew})
13:     **end for**
14: **end while**
15: Sort complete_plans by U_{state} values in an ascending order
16: final ← complete_plans[0]
17: **return** final

---

latency t_τ as $\hat{c}_\tau + C$ (right inequality in Equation 4), where $\hat{c}_\tau$ represents the predict cost of this partial plan and C is the upper bound on the error between t_τ and $\hat{c}_\tau$, calculated at a user-defined confidence level of $1 - \delta$. Based on this latency upper bound U_τ, we propose a generic CP-guided plan search algorithm that is compatible with basic plan search (BPS) algorithms. Algorithm 2 shows the details. We first initialize a priority queue with a set of partial plans, each representing a scan operation over a relation in the user query. We also initialize complete_plans to store the complete plans as they are identified (lines 1-2). At each iteration of the while loop, a partial plan, referred to as state, is retrieved from the priority queue according to BPS's logic for selecting the next plan (lines 3-4). This selection logic may involve fetching the partial plan with the minimum cost (Best-First Search), iterating over each state in the current queue (Beam Search [32]), or using other strategies. If state forms a complete plan, it is added to the set of complete plans (lines 5-8). Otherwise, the search continues from the current partial plan, state, by calling Explore(.), which generates a new set of partial plans along with their predicted costs.

For each new partial plan (stateNew), we define one function (described later) to compute its latency upper bound U_{stateNew} based on its predicted cost $\hat{c}_{\text{stateNew}}$ and the corresponding pattern-based upper bound on the latency-cost scores from $\mathbf{C}$. Then, these new states, stateNew, along with their corresponding values $\hat{c}_{\text{stateNew}}$ and U_{stateNew}, follow BPS's logic for inserting new plans (lines 10-12). This insertion logic may involve directly adding the new plans to the priority queue (non-optimized plan search). Alternatively, it may involve shrinking the queue to a specified size after inserting multiple plans, retaining only several promising plans for further exploration (Beam Search). The algorithm continues until n complete plans are identified. Finally, these complete plans are sorted based on their latency upper bounds, and the top-ranked plan is selected as the final plan.

Latency Upper Bound Calculation. We first extract the parent-children pattern of the input partial plan and then retrieve the corresponding upper bound on the latency-cost non-conformity scores $C_{\text{pat}} \in C$, referred to as `latencyCostUpperBound`. If no bound is found, `latencyCostUpperBound` is assigned the maximum value in C. This guarantees plan selection quality during the search in Algorithm 2. When a pattern is not found, adding $\max(C)$ to the predicted `cost` makes the latency upper bound U_τ very large. Thus, its priority in the CP-guided search is low compared to partial plans with patterns having values in C (i.e., trusted partial plans). More details of this calculation method can be found in [30].

6 EXPERIMENTAL EVALUATION

We evaluated our CP-based frameworks using different benchmarks and multiple prototypes to address the following questions: (1) How effective are the multi-granularity CP-based latency bounds (Section 6.2)? (2) How effective does our adaptive CP handle distribution shift (Section 6.3)? (3) How effective is our runtime verification (Section 6.4)? (4) How much performance gain can be achieved through effective violation detection and handling (Section 6.5)? (5) What benefits does CP-guided plan search provide in terms of plan quality and planning time (Section 6.6)? (6) What is the sensitivity of the hyper-parameters of our CP-based approach and their effects on the LQO verification process (Section 6.7)?

6.1 Experimental Setup

CP Integration with three LQOs. *Balsa [59]* is used as *white-box* LQO, owing to its leading performance in this category [14, 59, 64]. For bounded latency ranges, we verify both the unified-based and pattern-based upper bounds. For runtime verification, we calculate the robustness $\rho^\phi(\hat{x})$ (see Section 4) and compare it with the corresponding bound. If a violation is detected, our violation handler resolves it by reverting to PostgreSQL [15]. Finally, we perform CP-guided plan search to compare with the original Balsa. *Lero [64]* generates multiple candidate query plans and uses a learned oracle to rank them. The oracle applies pairwise comparisons to predict the more efficient plan, selecting the top-ranked one as the final output. Since Lero operates as a *black-box* LQO without directly accessible cost information, we use PostgreSQL's predicted costs as a reference and the actual latency to construct the CP model. In this context, the predicted cost $\hat{c}$ is available, and we use CP to derive a bounded range for the actual runtime t based on $\hat{c}$. We then apply CP models at various granularities to estimate the bounded range for the entire plan, individual levels, and identified patterns. *RTOS [62]* focuses on join order selection, leveraging a DRL framework in conjunction with Tree-LSTM [50] to effectively capture the structural information of query plans. RTOS outputs a join order hint, which we then inject into PostgreSQL to generate a complete query plan. As another representative of the *black-box* LQO, the CP integration in RTOS is similar to Lero's, in that we use PostgreSQL's predicted cost as a reference. Given that RTOS has less control over the selection of plan operators, we primarily use RTOS to validate our CP-based bounded latency range framework.

Notably, existing LQOs face significant limitations in handling distribution shifts, restricting their ability to generalize across different workloads. They are either hard-coded to specific schemas in their implementations (e.g., RTOS [62]) or require processing all training queries upfront to define the model structure, making it impossible to optimize unseen queries dynamically. Therefore, to explicitly enable LQOs to operate across distributions, we modified their prototype frameworks to support changing distributions.

Benchmarks. We evaluate the integration of CP with these LQOs on four widely used benchmarks - Join Order Benchmark (JOB) [28], Cardinality Estimation Benchmark (CEB) [39], JOBLight-train [24], and TPC-H [7]. For the static case evaluation, we use the JOB and TPC-H workloads. The JOB workload consists of 113 analytical queries over a real-world dataset from the Internet Movie Database. These queries involve complex joins and predicates, ranging from 3-16 joins, averaging 8 joins per query. For our experiments, we select 33 queries for the LQO training, while the remaining 80 queries are used for calibration and testing. TPC-H features synthetically generated data under a uniform distribution. We use a scale factor of 1 and templates for queries 3, 5, 7, 8, 10, 12, 13, and 14 to generate workloads, creating 130 queries with varying predicates. Of the generated queries, 60 are used for the LQO training, while the remaining 70 are used for calibration and testing. For the distribution shift case evaluation, we use JOBLight-train and CEB workloads along with JOB. JOBLight-train consists of synthetically generated queries with 3-table joins. CEB employs hand-crafted templates and query generation rules to construct challenging, large queries.

Hardware Settings. The experiments related to Balsa and RTOS were conducted on an Ubuntu 22 machine with an 8-core Intel Xeon D-1548 CPU @ 2.0GHz and 64 GB of RAM. The experiments related to Lero were conducted on an Ubuntu 22 machine with a 10-core Intel Xeon Silver 4114 @ 2.2GHz and 64 GB of RAM.

Evaluation Metrics. (1) *Coverage EC_m*: To empirically validate the CP marginal guarantees of Equation 1, we conduct the experiment over M iterations. For each iteration, we sample K calibration queries, $\{Q^{(1)}, \ldots, Q^{(K)}\}$, and N test queries, $\{Q_1^{(0)}, \ldots, Q_N^{(0)}\}$. Then, we calculate EC_m for iteration m using the following formula:

$$EC_m = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}(R_{m,n}^{(0)} \leq C(R_m^{(1)}, \ldots, R_m^{(K)})). \qquad (13)$$

This value measures the validity condition on the test set when applying our constructed C, indicating how many test cases are successfully covered. (2) *Frequency Density:* Across all sampling iterations, we calculate the frequency of each coverage level. To more effectively display the data, we use Kernel Density Estimation (KDE) for density representation. Intuitively, a higher frequency density for a specific coverage indicates a greater likelihood of its occurrence during sampling. (3) *CP Upper Bound C:* We use the C value to compare the spans of non-conformity scores across different hyper-parameter settings. (4) *Non-conformity Scores:* We display the distribution of non-conformity scores in the runtime verification context to visually validate how runtime constraints are satisfied. (5) *Execution Latency:* The actual execution latency, measured in milliseconds (ms), is used to assess the quality of generated query plans. (6) *Planning Time:* The time taken to generate a query plan is used to evaluate the algorithm's search efficiency during the planning, measured in milliseconds (ms).

Default Parameters. Unless otherwise mentioned, we run multiple sampling iterations ($M = 1000$) to observe the empirical coverage. We set $\delta = 0.1$ and the calibration-test split to be 50%-50%. In each iteration, we randomly select a fixed-size calibration set to generate non-conformity scores and then construct C based on the given $1 - \delta$. When validating a testing query, we evaluate on each operator i in the query for unified-based upper bounds or each pattern i (parent-children structure) for pattern-based upper bounds. Each instance is treated as a test step i, where we have the predicted cost $\hat{c}_i$ and the actual latency t_i. We normalize the predicted costs in Lero and RTOS cases with $f(\hat{c}) = \hat{c}/40$ and $\hat{c}/100$, respectively, to align these costs with actual latencies[6].

6.2 Bounded Range of Plan Actual Latency

Unified-based Upper Bound. We treat all the partial plans of a query plan equally with a single upper bound value for C (check Section 3.3). Figure 5 shows the empirical coverage in this case. We perform experiments on both the JOB and TPC-H workloads[7]. According to Equation 4, the CP theory predicts that the most frequent coverage should be greater than $1 - \delta = 0.9$, as reflected by the peak of the curve in both graphs. For both workloads, the peak of all the curves demonstrates this trend, empirically validating the correctness of applying CP with LQOs. Besides, if the non-conformity scores of an LQO span a broader range, they lead to a relaxed coverage curve with a higher coverage at the peak. We observe this phenomenon in RTOS with the JOB workload and in Lero with the TPC-H workload.
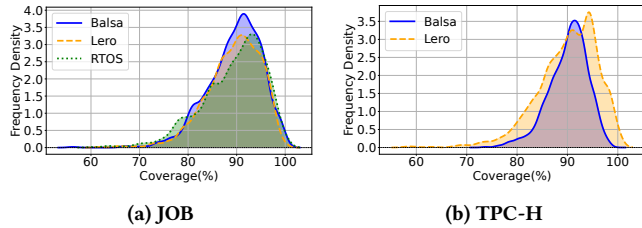
(a) JOB   **(b) TPC-H**

Figure 5: Unified-based Upper Bounds.

Pattern-based Upper Bound. Pattern-based upper bound provides finer granularity for generating a bounded range. In this experiment, we examine the top 3 and the least 3 frequently occurring patterns in Balsa on the JOB workload. Figure 6 (a) displays the top 3 popular patterns: (NL, NL, IS), (NL, HJ, IS), and (HJ, NL, SS). The peak coverage reaches 0.9, with a mean C value of 3056, indicating that Balsa's actual latencies vary within a range of ±3056 ms. Figure 6 (b) shows the least 3 popular patterns. Given that they have fewer appearances, which slightly exceeds the K^* threshold (Lemma 1), the curve is not as symmetric as the previous one. However, we also observe that the empirical coverage peak surpasses 90%, indicating reliable, bounded latency. This also shows that the CP theory holds its ground when the value of K is low yet greater than the K^* threshold.

[6]Note that we do not normalize the predicted costs for Balsa as it aims to predict the expected latency of the generated plan. So costs and latencies are well-aligned.
[7]For RTOS, we only evaluate on the JOB workload, as its open-source version is hard-coded to IMDB schemes.
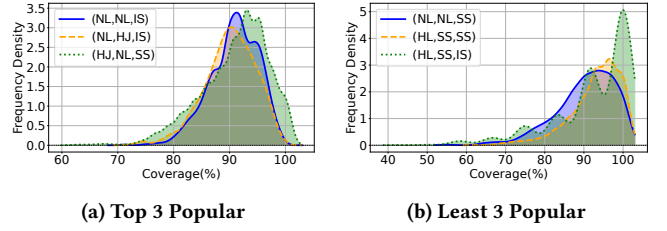
(a) Top 3 Popular   **(b) Least 3 Popular**

Figure 6: Valid Pattern-based Upper Bounds (Balsa on JOB).

6.3 Adaptive CP under Distribution Shift

We perform evaluations on Balsa [59] and RTOS [62] for distribution shift analysis. Our approach is inspired from existing works on distribution shift [40, 57, 58], where the LQOs are trained on one distribution and tested on another. Regarding the selection of distributions, we follow [40] and use the following distributions: JOB [28], CEB [39], JOBLight-train [24].

Balsa. We first quantify the total variation distance of calibration distribution JOB ($\mathcal{D}_0$) and test distribution CEB ($\mathcal{D}$): $TV(\mathcal{D}, \mathcal{D}_0)$. Following the computation in Section 3.2, we randomly select 500 plans from JOB and CEB to empirically compute $tv = TV(\mathcal{D}, \mathcal{D}_0) = 0.0736$. We then set the allowed distribution shift $\epsilon = 0.08$ to ensure that ϵ exceeds the estimated distribution shift tv.

Adaptive CP Validation. To maintain the original $(1 - \delta)$ confidence level for the latency bounds, an adjusted upper bound $\tilde{C}$ for the new distribution is obtained by an adjusted uncertainty probability $\tilde{\delta}$. We set the uncertainty probability $\delta = 0.2$. We sample $K = 300$ calibration plans from JOB ($\mathcal{D}_0$). We then compute the $\text{Prob}(R^{(0)} \leq C(R^{(1)}, \ldots, R^{(K)}))$ and $\text{Prob}(R^{(0)} \leq \tilde{C}(R^{(1)}, \ldots, R^{(K)}))$ for the non-adaptive and adaptive methods. Figure 7 shows the related results. In Figure 7a (without performing adaptive CP), the convergence is around 0.62, which is less than the expected $1 - \delta = 0.8$. This shows the previously computed upper bound C was not suitable for the new distribution. However, in Figure 7b (with adaptive CP), the coverage concentration is around 0.8, which shows our adjusted $\tilde{C}$ performs well with the new distribution.
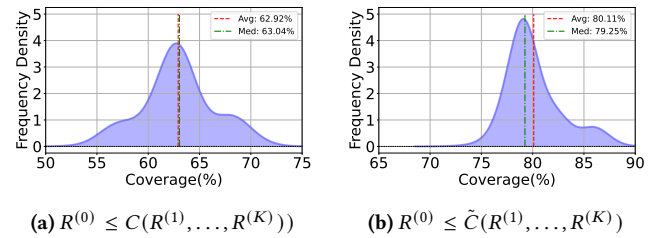
(a) $R^{(0)} \leq C(R^{(1)}, \ldots, R^{(K)})$ **(b)** $R^{(0)} \leq \tilde{C}(R^{(1)}, \ldots, R^{(K)})$

Figure 7: JOB Distribution Shift with Adaptive CP ($\delta = 0.2$).

RTOS. We train RTOS on JOB ($\mathcal{D}_0$) and sample $K = 300$ plans to construct the calibration set. Then, we introduce a new distribution, JOB-light ($\mathcal{D}$), for testing. The TV distance between these two distributions is $tv = 0.24916$, then we set $\epsilon = 0.25$. Figure 8 shows the result for uncertainty levels $\delta = 0.45$. The convergence of $\text{Prob}(R^{(0)} \leq C(R^{(1)}, \ldots, R^{(K)}))$ is around 0.2 and $\text{Prob}(R^{(0)} \leq \tilde{C}(R^{(1)}, \ldots, R^{(K)}))$ is 0.55 which is exactly $(1 - \delta)$.

This demonstrates that our adaptive CP methods work well with different prototypes and different uncertainty conditions.
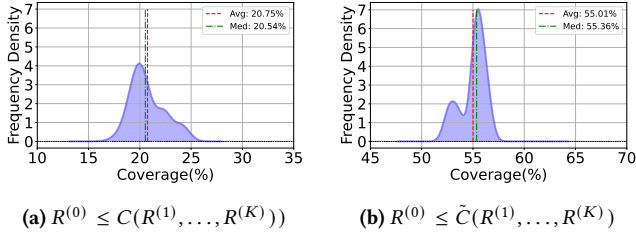

(a) $R^{(0)} \leq C(R^{(1)}, \ldots, R^{(K)}))$ **(b)** $R^{(0)} \leq \tilde{C}(R^{(1)}, \ldots, R^{(K)})$

Figure 8: RTOS Distribution Shift with Adaptive CP ($\delta = 0.45$).

6.4 Runtime Verification

We conduct runtime verification on the JOB workload using Balsa as a white-box LQO to validate Lemma 2, specifically demonstrating:

$$P(X \models \phi) \geq 1 - \delta, \quad \text{if} \quad \rho^{\phi}(\hat{x}) > C$$

We define our performance constraint with the following STL specification: $\phi = G(X < threshold)$, where G is the *always* operator defined in Section 2.2. We use this specification to bound the actual latency X when running an LQO's plan, whether partial or complete: the value of X is expected to always be less than the threshold. We set the threshold at 1000 and 2000, which implies that the cumulative latency of operations should not exceed 1000 ms and 2000 ms in the database context. We use this STL to detect violations and avoid unexpected long latency in execution. Based on ϕ, we define robust semantics: $\rho^{\phi}(x) = threshold - x$.

From the calibration queries, we construct the value of C and use this value to verify whether $\rho^{\phi}(\hat{x}) > C$. If this holds true, the actual latency adheres to the STL specification. Figure 9 shows the non-conformity score distribution with different thresholds. Our unified-based upper bound C covers $1 - \delta = 90\%$ of the non-conformity scores (left side of the red dashed line).
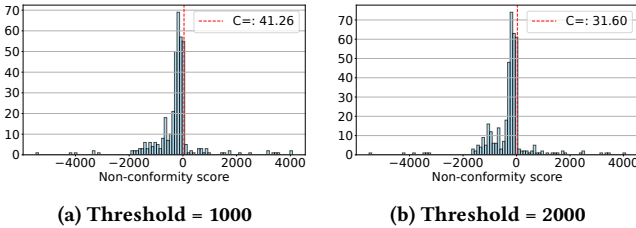

(a) Threshold = 1000 **(b) Threshold = 2000**

Figure 9: Non-conformity Scores in Runtime Verification.

$\boldsymbol{\phi = G(X < 1000)}$: We found that for 27 of the 30 queries ($|Q^{\text{test}}| = 30$), it holds that $\rho^{\phi}(\hat{x}) > C$ implies $X \models \phi$, confirming the correctness of runtime verification (Lemma 2). We also validated Equation 8 and found that 28 of the 30 test queries satisfy $\rho^{\phi}(\hat{x}) - \rho^{\phi}(X) \leq C$, which is greater than $(1 - \delta) = 0.9$, verifying the coverage of CP.
$\boldsymbol{\phi = G(X < 2000)}$: This is a looser threshold. We found that for 29 of the 30 queries ($|Q^{\text{test}}| = 30$), it holds that $\rho^{\phi}(\hat{x}) > C$ implies $X \models \phi$. A larger threshold demonstrates better coverage. We also

validated Equation 8 and found that 29 of the 30 test queries satisfy $\rho^{\phi}(\hat{x}) - \rho^{\phi}(X) \leq C$, further confirming our method.

6.5 Violation Detection and Handling

We perform violation detection using the JOB workload as discussed in Lemma 2 over the constraint ($\phi = G(X < 2000)$). If violations are detected, we introduce PostgreSQL to assist in generating a new query plan for execution. We compare two scenarios: *with CP* and *without CP*, representing CP-based violation detection and normal LQO planning, respectively. In this section, we focus on comparing the plan quality between these two methods.

Balsa with Violation Detection. Figure 10 presents the comparison results for Balsa. In total, 10 queries were flagged as potential violations. We trigger PostgreSQL to re-generate the query plans. Notably, for 7 out of these 10 queries, the query plans generated by PostgreSQL outperformed the Balsa-generated plans. The overall latency savings for these 7 queries amounted to 22.12%. For the remaining 3 queries, we observed that although Balsa results in better plans for them, these plans still violate the user constraint. That is why these queries are still detected by our verification framework.
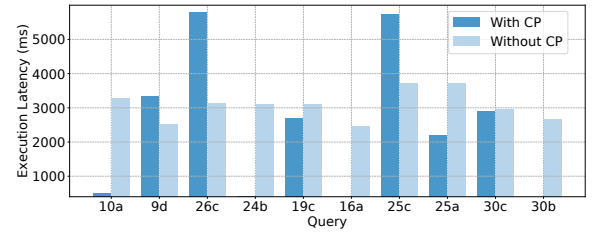

Figure 10: Violation Detection ($\phi = G(X < 2000)$): Latency Comparison With and Without CP for Balsa.

6.6 CP-Guided Actual Latency Upper Bound Query Optimizer

In this section, we still use Balsa as a representative white-box LQO to conduct CP-guided plan search experiments. Considering Balsa uses beam search [32] internally, our discussion revolves around CP-guided beam search. We evaluated Balsa at different training epochs: 50, 100, and 150, corresponding to moderately trained, well-trained, and highly trained Balsa, respectively. To evaluate our method, we use 33 queries from template b as the test set and the other 47 queries as the calibration set. The comparison experiments are conducted five times, and the average is reported to reduce the impact of system fluctuations on the planning and execution time.

6.6.1 Plan Improvement. Using the CP-guided plan search, we employ the CP-bounded latency upper bound as a heuristic to guide the beam search in constructing complete query plans. We evaluate whether this approach yields better results compared to the vanilla Balsa. Figure 11 shows the queries where we achieve improvements. Plan enhancements are observed in 11 out of 33 test queries while the rest (almost) maintained the same plan quality.

For a well-trained Balsa (100 iterations), our algorithm improves the plan quality for queries 14b, 28b, 6b and 9b as seen in Figure 12a, demonstrating consistent plan improvement. Even for the highly
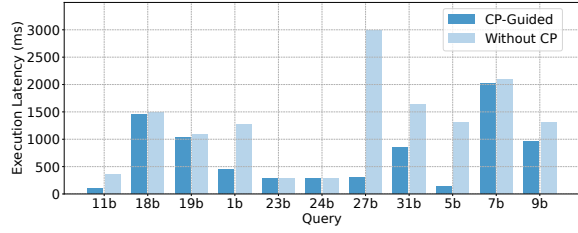
Figure 11: Plan Quality Comparison: CP Guided Algorithm vs. Balsa (50 iterations).

trained Balsa (150 iterations), we also observe several improved queries as seen in Figure 12b. Although Balsa can reliably and efficiently generate high-quality query plans at this stage, the CP-guided algorithm can still achieve better plans, even within this limited potential improvement search space. This further proves the effectiveness of our algorithm.
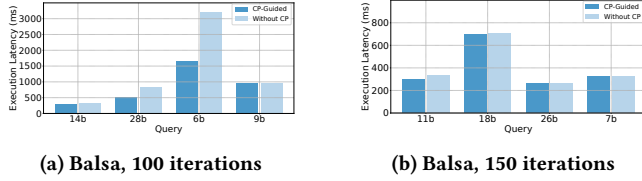

(a) Balsa, 100 iterations

(b) Balsa, 150 iterations

Figure 12: Plan Quality Comparison.

We also observe that our algorithm achieves greater improvements in plan quality during the early training stages of Balsa. This aligns with the intuition that it is easier to make improvements within a larger discovery space. As the number of training iterations increases, Balsa becomes progressively more refined, which naturally narrows the scope for further improvement.

We perform a deep-dive analysis for the queries where we achieve significant improvements: Query 27b in Figure 11 and Query 6b in Figure 12 (a). When we closely compare the query plans generated by CP-Guided and those without CP guidance, we observe that in Query 6b, Balsa originally selects a pattern of (NL, NL, IS). However, in the CP-guided plan search algorithm, we instead select a pattern of (HJ, NL, SS). The (HJ, NL, SS) pattern aligns with the valid patterns established for our reliable CP construction, whereas (NL, NL, IS) is not among them. By following our algorithm and being guided by CP, Query 6b achieves 48.52% latency reduction by replacing this pattern. For Query 27b, our CP-guided approach has an even greater impact. Without CP guidance, Balsa generates a left-deep tree; however, under CP guidance, it produces a bushy tree, resulting in a 9.84x improvement in latency. Query-level analysis reveals that our algorithm not only favors reliable patterns to construct the entire query plan but can also systematically optimize the structure of query plan, significantly enhancing the overall query plan quality.

6.6.2 Planning Time Comparison. For a moderately trained Balsa, we observe an improvement in planning time. Without CP assistance, the total planning time for all test queries is *6178.60 ms*; however, with our CP-guided algorithm, it is reduced to *5563.40*

ms, achieving an overall improvement of 9.96%. This demonstrates that our CP-guided approach can mitigate suboptimal LQO behaviors and accelerate the plan search. For the single query level, we can also observe Query 4b in Figure 13 reduces 74.40% planning time. This effect can be attributed to the optimization target of the CP-guided algorithm—the actual latency upper bound—which acts as a stricter heuristic than previously cost itself. This leads to a more direct search path within the search space. Compared to a moderately trained Balsa, our algorithm constrains the search scope, thereby reducing planning time.
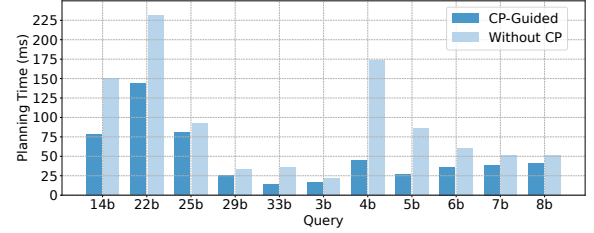

Figure 13: Planning Time Comparison: CP Guided Algorithm vs. Balsa (50 iterations).

Figure 14 illustrates that even with a highly trained Balsa, our algorithm improves planning time for 17 out of 33 queries. We also observe that as the number of LQO training iterations increases, the overall planning time for both CP-guided and without CP methods decreases. Comparing Figure 13 and Figure 14, we can see that the impact of our CP-guided algorithm on planning time is more pronounced at lower training iterations. This is because, with more extensive training, the LQO has a more refined initial search direction, resulting in a relatively smaller search space for our algorithm. Notably, for Queries 7b and 18b, we achieve improvements in both plan quality and planning time. These observations further demonstrate the effectiveness of our CP-guided algorithm.
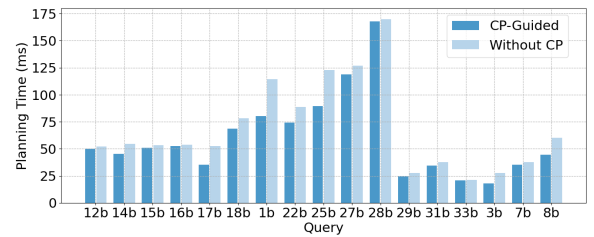

Figure 14: Planning Time Comparison: CP Guided Algorithm vs. Balsa (150 iterations).

6.7 Hyperparameter Microbenchmarking

In this section, we discuss two types of hyperparameter and observe their impact on the coverage. More evaluations can be found in [30]. **Impact of Changing the Sampling Iterations.** We begin by examining how the first hyper-parameter—sampling iterations—affects empirical coverage. We test with 100, 500, and 1000 sampling iterations. For each sampling iteration setting, we plot the density

of each coverage. Figure 15 (a) and Figure 15 (b) illustrate Balsa's performance on the JOB and TPC-H workloads, respectively. When the number of sampling iterations is low, the curve appears less smooth due to limited sampling. Since empirical coverage approximates the inherent coverage properties of CP, insufficient sampling fails to capture the expected behavior according to CP theory. With more iterations, the curve smooths, more accurately reflecting the intrinsic coverage properties of CP theory. Additionally, the curve displays a sharper peak shape. We also observe that the JOB workload exhibits a higher frequency density than the TPC-H workload. This is because JOB contains more joins, leading to a greater number of validation data points, which increases the frequency density.
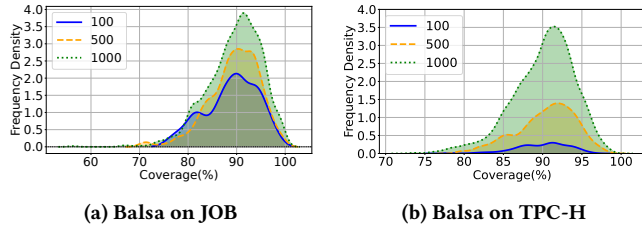

(a) Balsa on JOB   **(b) Balsa on TPC-H**

Figure 15: Impact of Changing the Sampling Iterations.

Impact of Uncertainty Probability δ. The second hyper-parameter is the uncertainty probability δ. We varied δ across four values: 0.1, 0.2, 0.3, and 0.4. Similar to the previous discussion, we expect the peaks of the coverage curve to align with $1 - \delta$, meaning the corresponding peaks should align with 0.9, 0.8, 0.7, and 0.6. Figure 16 illustrates this trend. Additionally, we observe that as δ decreases, the area under the curve becomes sharper and narrower, indicating a more concentrated coverage distribution. This suggests that with smaller values of δ (e.g., $\delta = 0.1$ in our graph), obtaining C values in a single sampling iteration is more likely to yield values centered around the expected confidence level of $1 - \delta$.
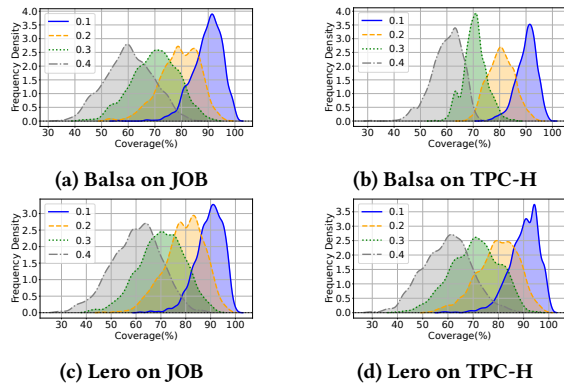

(a) Balsa on JOB   **(b) Balsa on TPC-H**

(c) Lero on JOB   **(d) Lero on TPC-H**

Figure 16: Impact on Choice of δ.

7 RELATED WORK

Learned Query Optimization (LQO). In recent years, numerous ML-based methods have been proposed to enhance query optimization. One line of work uses ML to improve cardinality estimates and predict query plan costs [25, 37, 39, 48, 49, 60, 61], but these gains have not been shown to yield better plans [39]. Consequently, research has shifted to directly learning plan optimization [18, 26, 34–36, 59, 62, 62, 64], either by constructing plans from scratch (e.g., Neo [35], Balsa [59]) or by selecting among candidates generated by traditional optimizers (e.g., Bao [34], Lero [64]). Although all LQOs have demonstrated improved query performance, they typically do not consider the robustness issues (no guarantees on stability or regression avoidance). Kepler [18] and Roq [23] are the closest works to our objective. Kepler employs robust neural network prediction techniques to reduce tail latency and minimize query regressions. Specifically, it utilizes Spectral-normalized Neural Gaussian Processes [31] to quantify its confidence in plan prediction and falls back to the traditional optimizer when uncertain. Roq introduces robustness notions in the context of query optimization and incorporates a ML pipeline to predict plan cost and risk. However, neither method provides theoretical guarantees or formally formulates LQO verification. To our knowledge, our work is the first to address the verification problem in LQOs by providing formal guarantees and using them to guide the plan construction process. **Conformal Prediction (CP).** CP is introduced to provide a robust statistical framework for quantifying prediction uncertainty (e.g., [2, 46, 55]). Extensive research has explored the application of CP in distribution-agnostic settings, delivering reliable performance guarantees even in non-stationary environments (e.g., [1, 21, 27, 44, 63]). Additionally, extensions of CP have been applied to time-series data [13, 52] and STL-based runtime verification in real-time systems (e.g., autonomous cars [29], autonomous robots [41], aircraft simulation [29, 44]). Recently, several work discuss applying CP within different distribution shift conditions [4, 22, 52, 63]. Besides, CP has been adapted for policy evaluation in reinforcement learning [20, 51], time-series forecasting [47], and outlier detection [5]. It has also been employed to monitor risks in evolving data streams [42] and detect change points in time-series data [53, 54].

8 CONCLUSION

To the best of our knowledge, we are the first to introduce Conformal Predication for verifying learned database components, with a focus on the learned query optimization. Our framework encompasses CP-based latency bounds across multiple granularities and runtime verification. Our framework also employs an adaptive CP approach for handling distribution shifts. Further, we introduce a CP-guided plan search algorithm capable of enhancing LQOs. We have demonstrated that CP provides a flexible, lightweight verification approach that establishes trustworthy prediction boundaries. Our methods can be deployed in real-world production environments, achieving formal verification without significant computational overhead. Our evaluation shows that CP can be used to achieve tight upper bounds on actual latency using predicted cost. Adaptive CP maintains the confidence levels even under distribution shift. CP-guided LQOs produce plans with up to 9.84x better actual latency, over the entire workload CP-guided LQOs show a 9.96% reduction in actual latency. Our CP framework also lays the groundwork for broader applications across various learned components within database systems.

REFERENCES

[1] Anastasios N. Angelopoulos and Stephen Bates. 2022. A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification. arXiv:2107.07511 [cs.LG] https://arxiv.org/abs/2107.07511

[2] Anastasios N. Angelopoulos and Stephen Bates. 2023. Conformal Prediction: A Gentle Introduction. *Found. Trends Mach. Learn.* 16, 4 (mar 2023), 494–591. https://doi.org/10.1561/2200000101

[3] Brian Babcock and Surajit Chaudhuri. 2005. Towards a Robust Query Optimizer: A Principled and Practical Approach. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data.* 119–130.

[4] Rina Foygel Barber, Emmanuel J Candes, Aaditya Ramdas, et al. 2023. Conformal Prediction Beyond Exchangeability. *The Annals of Statistics* 51, 2 (2023), 816–845.

[5] Stephen Bates, Emmanuel Candès, Lihua Lei, et al. 2023. Testing for outliers with conformal p-values. *The Annals of Statistics* 51, 1 (Feb. 2023). https://doi.org/10.1214/22-aos2244

[6] Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* 20, 4, Article 14 (sep 2011), 64 pages. https://doi.org/10.1145/2000799.2000800

[7] TPC-H Benchmark. [n. d.]. http://www.tpc.org/tpch/.

[8] Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, et al. 2019. Neural Predictive Monitoring. In *Runtime Verification*, Bernd Finkbeiner and Leonardo Mariani (Eds.). Springer International Publishing, Cham, 129–147.

[9] Simin Cai, Barbara Gallina, Dag Nyström, et al. 2016. A Formal Approach for Flexible Modeling and Analysis of Transaction Timeliness and Isolation. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems* (Brest, France) *(RTNS '16).* Association for Computing Machinery, New York, NY, USA, 3–12. https://doi.org/10.1145/2997465.2997495

[10] Simin Cai, Barbara Gallina, Dag Nyström, et al. 2019. Statistical Model Checking for Real-Time Database Management Systems: A Case Study. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (Zaragoza, Spain). IEEE Press, 306–313. https://doi.org/10.1109/ETFA.2019.8869326

[11] Simin Cai, Barbara Gallina, Dag Nyström, et al. 2018. Specification and Formal Verification of Atomic Concurrent Real-Time Transactions. In *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC).* 104–114. https://doi.org/10.1109/PRDC.2018.00021

[12] Francesca Cairoli, Nicola Paoletti, and Luca Bortolussi. 2023. Conformal Quantitative Predictive Monitoring of STL Requirements for Stochastic Processes. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control* (San Antonio, TX, USA) *(HSCC '23).* Association for Computing Machinery, New York, NY, USA, Article 1, 11 pages. https://doi.org/10.1145/3575870.3587113

[13] M. Cauchois, S. Gupta, A. Ali, et al. 2020. Robust validation: Confident predictions even when distributions shift. *arXiv preprint arXiv:2008.04267* (2020).

[14] Xu Chen, Zhen Wang, Shuncheng Liu, et al. 2023. BASE: Bridging the Gap between Cost and Latency for Query Optimization. *Proc. VLDB Endow.* 16, 8 (April 2023), 1958–1966. https://doi.org/10.14778/3594512.3594525

[15] PostgreSQL DBMS. [n. d.]. PostgreSQL DBMS. https://www.postgresql.org/.

[16] Luc Devroye, László Györfi, and Gábor Lugosi. 2013. *A Probabilistic Theory of Pattern Recognition.* Vol. 31. Springer Science & Business Media.

[17] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems* (Klosterneuburg, Austria) *(FOR-MATS'10).* Springer-Verlag, Berlin, Heidelberg, 92–106.

[18] Lyric Doshi, Vincent Zhuang, Gaurav Jain, et al. 2023. Kepler: Robust Learning for Parametric Query Optimization. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[19] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of Temporal Logic Specifications for Continuous-Time Signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291. https://doi.org/10.1016/j.tcs.2009.06.021

[20] Daniele Foffano, Alessio Russo, and Alexandre Proutiere. 2023. Conformal Off-Policy Evaluation in Markov Decision Processes. arXiv:2304.02574 [cs.LG] https://arxiv.org/abs/2304.02574

[21] M. Fontana, G. Zeni, and S. Vantini. 2023. Conformal prediction: A unified review of theory and new challenges. *Bernoulli* 29, 1 (2023), 1–23.

[22] Isaac Gibbs and Emmanuel J. Candès. 2021. Adaptive Conformal Inference Under Distribution Shift. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21).* Curran Associates Inc., Red Hook, NY, USA, Article 128, 13 pages.

[23] Amin Kamali, Verena Kantere, Calisto Zuzarte, et al. 2024. Roq: Robust Query Optimization Based on a Risk-aware Learned Cost Model. arXiv:2401.15210 [cs.DB] https://arxiv.org/abs/2401.15210

[24] Andreas Kipf, Michael Freitag, Dimitri Vorona, et al. 2019. Estimating Filtered Group-By Queries is Hard: Deep Learning to the Rescue. In *1st International Workshop on Applied AI for Database Systems and Applications.*

[25] Andreas Kipf, Thomas Kipf, Bernhard Radke, et al. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *9th Biennial Conference on Innovative Data Systems Research (CIDR '19).*

[26] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, et al. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *arXiv preprint arXiv:1808.03196* (2018).

[27] J. Lei, M. G'Sell, A. Rinaldo, et al. 2018. Distribution-Free Predictive Inference for Regression. *J. Amer. Statist. Assoc.* 113, 523 (2018), 1094–1111.

[28] Viktor Leis, Andrey Gubichev, Atanas Mirchev, et al. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (nov 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[29] Lars Lindemann, Xin Qin, Jyotirmoy V. Deshmukh, et al. 2023. Conformal Prediction for STL Runtime Verification. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)* (San Antonio, TX, USA) *(ICCPS '23).* Association for Computing Machinery, New York, NY, USA, 142–153. https://doi.org/10.1145/3576841.3585927

[30] Hanwen Liu, Shashank Giridhara, and Ibrahim Sabek. 2025. Conformal Prediction for Verifiable Learned Query Optimization. arXiv:2505.02284 [cs.DB] https://arxiv.org/abs/2505.02284

[31] Jeremiah Zhe Liu, Zi Lin, Shreyas Padhy, et al. 2020. Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20).* Curran Associates Inc., Red Hook, NY, USA, Article 629, 15 pages.

[32] Bruce P Lowerre and B Raj Reddy. 1976. Harpy, a connected speech recognition system. *The Journal of the Acoustical Society of America* 59, S1 (1976), S97–S97.

[33] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Yassine Lakhnech and Sergio Yovine (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 152–166.

[34] R. Marcus et al. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD.*

[35] Ryan Marcus, Parimarjan Negi, Hongzi Mao, et al. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (jul 2019), 1705–1718. https://doi.org/10.14778/3342263.3342644

[36] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning For Join Order Enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management.* 1–4.

[37] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proc. VLDB Endow.* 12, 11 (July 2019), 1733–1746. https://doi.org/10.14778/3342263.3342646

[38] Alnur Ali Maxime Cauchois, Suyash Gupta et al. 2024. Robust Validation: Confident Predictions Even When Distributions Shift. *J. Amer. Statist. Assoc.* 119, 548 (2024), 3033–3044.

[39] Parimarjan Negi, Ryan Marcus, Andreas Kipf, et al. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *Proc. VLDB Endow.* 14, 11 (July 2021), 2019–2032. https://doi.org/10.14778/3476249.3476259

[40] Parimarjan Negi, Ziniu Wu, Andreas Kipf, et al. 2023. Robust Query Driven Cardinality Estimation Under Changing Workloads. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1520–1533.

[41] Baptiste Pelletier, Charles Lesire, Christophe Grand, et al. 2023. Predictive Runtime Verification of Skill-based Robotic Systems using Petri Nets. In *2023 IEEE International Conference on Robotics and Automation (ICRA).* 10580–10586. https://doi.org/10.1109/ICRA48891.2023.10160434

[42] Aleksandr Podkopaev and Aaditya Ramdas. 2022. Tracking the Risk of a Deployed Model and Detecting Harmful Distribution Shifts. arXiv:2110.06177 [stat.ML] https://arxiv.org/abs/2110.06177

[43] PostgreSQL. [n. d.]. PostgreSQL Explain Analyze. https://www.postgresql.org/docs/current/sql-explain.html.

[44] Xin Qin, Yuan Xia, Aditya Zutshi, et al. 2022. Statistical Verification of Cyber-Physical Systems using Surrogate Models and Conformal Inference. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS).* 116–126. https://doi.org/10.1109/ICCPS54341.2022.00017

[45] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, et al. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data.* 23–34.

[46] Glenn Shafer and Vladimir Vovk. 2007. A Tutorial on Conformal Prediction. arXiv:0706.3188 [cs.LG] https://arxiv.org/abs/0706.3188

[47] Kamile Stankeviciute, Ahmed M. Alaa, and Mihaela van der Schaar. 2021. Conformal Time-series Forecasting. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 6216–6228. https://proceedings.neurips.cc/paper_files/paper/2021/file/312f1ba2a72318edaaa995a67835fad5-Paper.pdf

[48] Ji Sun and Guoliang Li. 2019. An End-To-End Learning-Based Cost Estimator. *Proc. VLDB Endow.* 13, 3 (Nov. 2019), 307–319. https://doi.org/10.14778/3368289.3368296

[49] Ji Sun, Jintao Zhang, Zhaoyan Sun, et al. 2021. Learned Cardinality Estimation: A Design Space Exploration and a Comparative Evaluation. *Proc. VLDB Endow.* 15, 1 (sep 2021), 85–97. https://doi.org/10.14778/3485450.3485459

[50] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. *arXiv preprint arXiv:1503.00075* (2015).

[51] M. F. Taufiq, J.-F. Ton, R. Cornish, et al. 2022. Conformal Off-Policy Prediction in Contextual Bandits. *arXiv preprint arXiv:2206.04405* (2022).

[52] Ryan J. Tibshirani, Rina Foygel Barber, Emmanuel J. Candès, et al. 2019. Conformal Prediction under Covariate Shift. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 227, 11 pages.

[53] Denis Volkhonskiy, Evgeny Burnaev, Ilia Nouretdinov, et al. 2017. Inductive Conformal Martingales for Change-Point Detection. In *Proceedings of the Sixth Workshop on Conformal and Probabilistic Prediction and Applications (Proceedings of Machine Learning Research, Vol. 60)*, Alex Gammerman, Vladimir Vovk, Zhiyuan Luo, and Harris Papadopoulos (Eds.). PMLR, 132–153. https://proceedings.mlr.press/v60/volkhonskiy17a.html

[54] Vladimir Vovk. 2021. Testing Randomness Online. *Statist. Sci.* 36, 4 (Nov. 2021). https://doi.org/10.1214/20-sts817

[55] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. 2005. *Algorithmic Learning in a Random World*. Springer-Verlag, Berlin, Heidelberg.

[56] Cristina M. Wilcox and Brian C. Williams. 2010. Runtime Verification of Stochastic, Faulty Systems. In *Proceedings of the First International Conference on Runtime Verification* (St. Julians, Malta) *(RV'10)*. Springer-Verlag, Berlin, Heidelberg, 452–459.

[57] Peizhi Wu and Zachary G. Ives. 2024. Modeling Shifting Workloads for Learned Database Systems. *Proc. ACM Manag. Data* 2, 1, Article 38 (March 2024), 27 pages. https://doi.org/10.1145/3639293

[58] Peizhi Wu, Ryan Marcus, and Zachary G. Ives. 2023. Adding Domain Knowledge to Query-Driven Learned Databases. arXiv:2312.01025 [cs.DB] https://arxiv.org/abs/2312.01025

[59] Z. Yang et al. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *SIGMOD*.

[60] Zongheng Yang, Amog Kamsetty, Sifei Luan, et al. 2020. NeuroCard: One Cardinality Estimator for All Tables. *arXiv:2006.08109 [cs]* (June 2020). arXiv:2006.08109 [cs]

[61] Zongheng Yang, Eric Liang, Amog Kamsetty, et al. 2019. Deep Unsupervised Cardinality Estimation. *Proc. VLDB Endow.* 13, 3 (nov 2019), 279–292. https://doi.org/10.14778/3368289.3368294

[62] Xiang Yu, Guoliang Li, Chengliang Chai, et al. 2020. Reinforcement Learning With Tree-LSTM for Join Order Selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1297–1308.

[63] Yiqi Zhao, Bardh Hoxha, Georgios Fainekos, et al. 2024. Robust Conformal Prediction for STL Runtime Verification Under Distribution Shift. In *2024 ACM/IEEE 15th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 169–179.

[64] Rong Zhu, Wei Chen, Bolin Ding, et al. 2023. Lero: A Learning-To-Rank Query Optimizer. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1466–1479.