PBench: Workload Synthesizer with Real Statistics for Cloud Analytics Benchmarking

Yan Zhou[†]
Renmin University, China
zhouyan2018@ruc.edu.cn

Chunwei Liu[†]
MIT CSAIL
chunwei@csail.mit.edu

Bhuvan Urgaonkar
Penn State and AWS
urgaonkb@amazon.com

Zhengle Wang
Renmin University, China
wangzhengle@cau.edu.cn

Magnus Mueller
Amazon Web Services
magnusmu@amazon.com

Chao Zhang*
Renmin University, China
cycchao@ruc.edu.cn

Songyue Zhang
Renmin University, China
zhangsongyue@ruc.edu.cn

Pascal Pfeil
Amazon Web Services
pfeip@amazon.de

Dominik Horn
Amazon Web Services
domhorn@amazon.de

Zhengchun Liu
Amazon Web Services
zcl@amazon.com

Davide Pagano
Amazon Web Services
dpagano@amazon.com

Tim Kraska
MIT and AWS
kraska@mit.edu

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

Ju Fan*
Renmin University, China
fanj@ruc.edu.cn

ABSTRACT

Cloud service providers commonly use standard benchmarks like TPC-H and TPC-DS to evaluate and optimize cloud data analytics systems. However, these benchmarks rely on fixed query patterns and fail to capture real execution statistics of production cloud workloads. Although some cloud database vendors have recently released real workload traces, these traces alone do not qualify as benchmarks, as they typically lack essential components (i.e., queries and databases). To overcome this limitation, this paper studies a new problem of *workload synthesis with real statistics*, which generates *synthetic workloads* that closely approximate real execution statistics, including key performance metrics and operator distributions. To address this problem, we propose PBench, a novel workload synthesizer that constructs synthetic workloads by (1) selecting and combining workload components from existing benchmarks and (2) augmenting new workload components. This paper studies the key challenges in PBench. First, we address the challenge of balancing performance metrics and operator distributions by introducing a multi-objective optimization-based component selection method. Second, to capture the temporal dynamics of real workloads, we design a timestamp assignment method that progressively refines workload timestamps. Third, to handle the disparity between the original workload and the candidate workload, we propose a component augmentation approach that leverages large language models (LLMs) to generate additional workload components while maintaining statistical fidelity. Experimental results show that PBench reduces approximation error by up to 6× compared to state-of-the-art methods.

PVLDB Reference Format:
Yan Zhou, Chunwei Liu, Bhuvan Urgaonkar, Zhengle Wang, Magnus Mueller, Chao Zhang, Songyue Zhang, Pascal Pfeil, Dominik Horn, Zhengchun Liu, Davide Pagano, Tim Kraska, Samuel Madden, and Ju Fan. PBench: Workload Synthesizer with Real Statistics for Cloud Analytics Benchmarking. PVLDB, 18(11): 3883 - 3895, 2025.

doi:10.14778/3749646.3749661

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ruc-datalab/PBench/tree/main.

1 INTRODUCTION

The growing significance of cloud-based database systems has created a demand for new benchmarking approaches. Existing benchmarks, notably those from TPC [5, 12, 27], are not well-suited for cloud workloads due to two key limitations. First, these benchmarks are not designed to accommodate the varying degrees of concurrency commonly observed in cloud databases, making them ineffective in accurately capturing the *performance characteristics* of modern cloud database systems. Second, recent studies [31, 32] have shown that the distribution of query operators (e.g., the frequency of joins and aggregates) in modern cloud workloads differs significantly from those in traditional benchmarks. Given that different operator distributions correspond to various business logic patterns and can lead to varying cost-optimal query optimization strategies in the cloud [11, 19, 38, 41], it is crucial for benchmark workloads to accurately reflect the real-world *operator distributions*.

To address these limitations, several cloud database vendors, including Snowflake [7] and Amazon Redshift [2], have released *customer workload traces* [31, 33], which are anonymized logs capturing *real execution statistics* from cloud workloads. These traces

Chao Zhang and Ju Fan are the corresponding authors. [†] indicates equal first author contribution.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749661

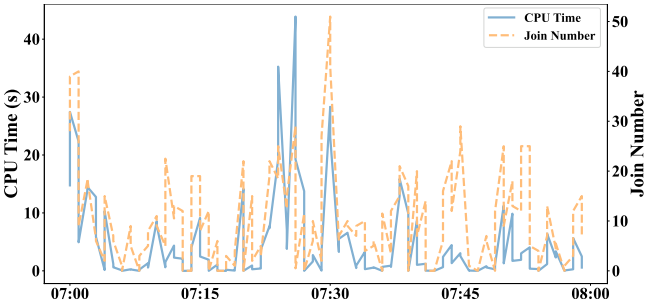


Figure 1: An Example Workload Trace from *Redset*

include key performance metrics (e.g., CPU Time) and workload characteristics (e.g., operator distributions), offering valuable insights into real-world cloud database behavior. Figure 1 illustrates an example workload trace from *Redset* [31], which provides real statistics, including a performance metric, CPU Time, and an operator distribution feature, Join Number.

Leveraging these traces presents an opportunity to develop benchmarks that better reflect real-world cloud database workloads. However, these traces alone do not constitute benchmarks, as they typically lack essential components like the original SQL queries and their underlying databases, making them unsuitable for directly evaluating cloud database system performance.

Workload Synthesis with Real Statistics. To address this issue, this paper introduces a new problem of *workload synthesis with real statistics*, which aims to generate *synthetic workloads* that closely approximate the performance metrics (e.g., CPU Time) and operator distributions (e.g., the frequency of joins, scans, and aggregations) observed in real cloud workload traces. Given that real workload traces lack the original SQL queries and their underlying databases, this work focuses on constructing synthetic workloads by (1) selecting and combining *workload components*, i.e., SQL queries and their corresponding databases, from *existing benchmarks*, such as TPC-H and TPC-DS, and (2) generating new components when existing queries and databases are insufficient.

Key Challenges. Effectively addressing the problem of *workload synthesis with real statistics* presents three key technical challenges.

(C1) Balancing approximation objectives. Selecting an appropriate set of workload components from existing benchmarks is challenging due to the need to simultaneously approximate both performance metrics and operator distributions. These two objectives are inherently intertwined, i.e., adjusting one (e.g., increasing the join number) often impacts the other (e.g., leading to higher CPU time). This interdependence complicates the workload component selection process, making it difficult to construct a synthetic workload that accurately captures the statistics of real workload traces.

(C2) Preserving temporal dynamics. Since cloud workloads exhibit varying levels of concurrency, short-term fluctuations (e.g., within 30 seconds), and periodic query patterns [20], naively distributing queries over time may fail to capture these temporal dynamics, leading to discrepancies between the synthetic and real workloads. Thus, the second challenge lies in effectively assigning timestamps to the selected workload components to preserve the temporal dynamics in real workload traces.

(C3) Augmenting workload components. Workload components from existing benchmarks may be insufficient to construct a synthetic workload that accurately reflects real traces due to the disparity between the original workload and the candidate workload. These disparities are often caused by different query patterns and underlying databases, and it is challenging to augment additional workload components of high quality due to the huge search space.

Our PBench Approach. In this paper, we propose PBench, a novel approach for workload synthesis that effectively approximates real statistics from cloud workload traces. PBench takes a real workload trace (e.g., from Redset [31] or Snowset [33]) as input and constructs a synthetic workload by strategically selecting and combining workload components from existing benchmarks.

Technically, to address challenge (C1), we propose a **component selection** approach that effectively balances the approximation of both performance metrics and operator distributions. We formulate component selection as a multi-objective optimization problem using integer linear programming (ILP) and employ efficient algorithms to solve it. To address challenge (C2), we propose a **timestamp assignment** method that incrementally refines the temporal distribution of the selected workload components. We employ a simulated annealing-based approach that progressively adjusts workload timestamps, transitioning from a coarse-grained approximation to a fine-grained alignment with the temporal dynamics of real workload traces. To address challenge (C3), we propose a **component augmentation** approach that leverages large language models (LLMs) to generate new workload components. We design effective prompting strategies and a trial-and-error generation mechanism for the LLM to produce new workload components that significantly reduce the approximation error.

Differences from Existing Methods. Several existing studies have explored generating synthetic workloads based on standard benchmarks [32, 35]. However, they fall short in accurately approximating complex real cloud workload traces. First, CAB [32] employs heuristic strategies, such as randomly selecting queries from a predefined query pool, and lacks a principled method to ensure that the generated synthetic workload closely approximates the real workload. Second, while Stitcher [35] employs Bayesian optimization [15], it overlooks operator distributions. Given that the inherent interaction between performance metrics and operator distributions, as mentioned previously, this limitation hinders Stitcher's ability to generate balanced workloads. Moreover, Stitcher selects all queries from a single benchmark without filtering out irrelevant or redundant queries, thus lacking a *fine-grained* mechanism to select queries relevant to real workloads.

Downstream Tasks and Extensibility. By accurately fitting execution statistics from real workload traces, PBench is designed to support a wide range of downstream tasks that require realistic and representative workloads. In industrial settings, PBench can facilitate downstream tasks in ongoing industrial use cases involving advanced workload management tasks such as AutoWLM [29] and Redshift's autoscaling mechanism, Sage Scaling [25]. In this paper, we demonstrate the applicability of PBench to a downstream task, namely *database benchmarking* in our experiments. By preserving the high fidelity of the original workloads, PBench enables meaningful comparisons of performance across systems like Snowflake

Table 1: Notations

Notation	Description	Notation	Description
W	Original Workload	q'_j	Query of C_j
$\widetilde{W}$	Synthesized Workload	D'_j	Database of C_j
F	Performance Feature	F'_j	F of C_j
M	Performance Metrics	m'_{hj}	h-th M of C_j
O	Operator Distributions	o'_{uj}	u-th O of C_j
q_j	j-th Customer Query	T'_j	Duration of C_j
t_j	Timestamp of q_j	$\widetilde{W_{\mathbb{C}}}$	Workload by $\mathbb{C}$
D	Customer's Database	$x_j^{(i)}$	Number of C_j in $\Gamma^{(i)}$ of $\widetilde{W_{\mathbb{C}}}$
F_j	F of q_j	$t_{jk}^{(i)}$	t of k-th C_j in $\Gamma^{(i)}$ of $\widetilde{W_{\mathbb{C}}}$
$\Gamma^{(i)}$	i-th Time Window	F_j^G	j-th Generation Goal
$\gamma^{(ij)}$	$\Gamma^{(i)}$'s j-th Time Interval	$\mathbb{E}_{\mathbb{P}i}$	F_j^G 's Positive Example
$F^{(i)}$	W's F in $\Gamma^{(i)}$	$\mathbb{E}_{\mathbb{N}j}$	F_j^G 's Negative Examples
$\widetilde{F^{(i)}}$	$\widetilde{W}$'s F in $\Gamma^{(i)}$	C_{ij}^P	F_i^G 's j-th Positive Examples
$F^{(ij)}$	W's F in $\gamma^{(ij)}$	C_{ij}^N	F_i^G 's j-th Negative Example
$\widetilde{F^{(ij)}}$	$\widetilde{W}$'s F in $\gamma^{(ij)}$	$\widetilde{C}$	Synthesized Component
C_j	j-th Workload Component	$F_{\widetilde{C}}$	F of $\widetilde{C}$

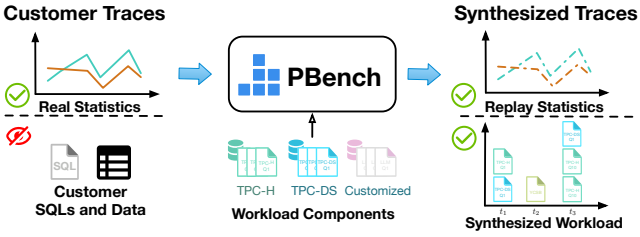


Figure 2: Workload Synthesis with Real Statistics

and Redshift. Moreover, PBench is an extensible framework that can incorporate more metrics (e.g., query plan patterns and execution behaviors). We illustrate this flexibility through an experiment that matches sub-query structures, showcasing its potential for supporting downstream tasks such as query optimization.

Contributions. Our contributions are summarized as follows.

(1) We introduce a novel problem of workload synthesis with real statistics, which is formally defined in Section 2.

(2) We propose PBench, an effective approach to workload synthesis with real statistics (Section 3). We develop effective techniques in PBench for component selection and timestamp assignment (Section 4). We also leverage an LLM-based method for component augmentation (Section 5).

(3) We have developed and open-sourced PBench to support cloud analytics benchmarking. To evaluate its effectiveness, we conducted extensive experiments on two widely-recognized cloud workload traces, *Snowset* [33] and *Redset* [31] (Section 6). Experimental results show that PBench achieves up to a 6x reduction in approximation error compared to state-of-the-art methods.

2 PROBLEM FORMALIZATION

Figure 2 illustrates the problem of workload synthesis with real statistics. Given a real workload trace where original SQL queries and underlying databases are unavailable, the problem is to generate a synthetic workload that, when executed, produces statistics closely approximate the target workload's performance metrics and operator distributions. In particular, given that real workload traces lack the original SQL queries and their underlying databases, this work focuses on constructing synthetic workloads by (1) combining *workload components*, i.e., SQL queries and their corresponding databases, from *existing benchmarks*, such as TPC-H and TPC-DS, and (2) generating new workload components as needed from scratch.

Then, the generated synthetic workload offers an effective solution for *downstream tasks*, like database benchmarking [9, 39, 40], performance tuning [30], and bug detection [3], allowing database developers to perform realistic evaluations while preserving privacy and avoiding exposure of sensitive workloads or data.

Below, we formally define the studied problem. The notations used throughout this paper are summarized in Table 1. We first formalize real workload traces derived from *Snowset* or *Redset*.

Definition 2.1. **Cloud Workload** W**.** A workload W is a time series of queries $\{q_1, \ldots, q_n\}$ to be run against a database D, where each timestamped query q_i is denoted as $\{D, \{t_i, q_i\}\}$. Note that while the queries and their corresponding databases are conceptually represented, they remain inaccessible in practice.

Definition 2.2. **Performance Feature** F**.** We formalize the real statistics of a workload trace using the performance feature $F =< M, O >$, where M represents a set of performance metrics, and O denotes operator distributions, capturing the percentage distribution of different SQL operators. To facilitate window-level synthesis, we aggregate the performance features within the i-th time window $\Gamma^{(i)}$, yielding the window-based performance feature representation $< \Gamma^{(i)}, M^{(i)}, O^{(i)} >$. Here, $M^{(i)}$ is the aggregated performance metrics vector, and $O^{(i)}$ is the aggregated operator distribution vector.

Recall that this work focuses on constructing synthetic workloads by combining *workload components* from *existing benchmarks*. Thus, we define the *workload component* as follows.

Definition 2.3. **Workload Component** C**.** The synthetic workload is constructed over a set of workload components $C_1, C_2, \ldots, C_n$, where each component C_j includes a query q'_j and its corresponding populated database D'_j, both derived from existing benchmarks.

Now, we are ready to define the problem of workload synthesis with real statistics as follows.

Definition 2.4. **Workload Synthesis with Real Statistics.** Given the performance features F of a target workload W, the problem is to synthesize a new workload $\widetilde{W}$ such that, for each time window, the synthesized performance features $\langle \Gamma^{(i)}, \widetilde{M}^{(i)}, \widetilde{O}^{(i)} \rangle$ closely approximate the target features $\langle \Gamma^{(i)}, M^{(i)}, O^{(i)} \rangle$, minimizing the approximation error (e.g., using Mean Absolute Error). The synthesis process involves both (1) assembling workload components by selecting queries from existing benchmarking pools and (2) generating new workload components as needed from scratch.

Remarks. We assume that the source cloud cluster and the target cloud cluster for benchmarking are sufficiently similar in terms of node count, machine configurations, and database system, ensuring a high-fidelity evaluation environment. In addition, the quality of synthesized workloads depends on whether the system used to generate the initial workload trace is the same as the one used to synthesize and execute the new workload. This is because PBench is designed to produce high-fidelity synthetic workloads by closely

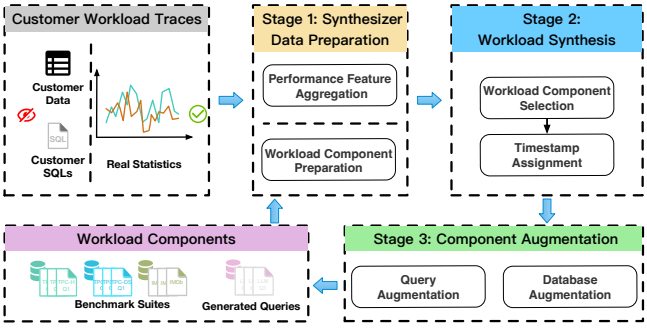


Figure 3: Overall Framework of PBench

matching system-specific performance features, which are influenced by the underlying system behavior. However, Once PBench has synthesized a benchmark from a particular system trace (e.g., Redshift or Snowflake), the resulting workload can be replayed on other platforms to enable cross-system comparisons.

Moreover, In this paper, we focus on matching *CPU time*, *scanned bytes*, and *operator distributions* because these are among the most critical performance metrics for cloud OLAP systems. For example, CPU time reflects computational resource consumption, while scanned bytes indicate storage I/O, both of which are key billing units in cloud environments and directly impact system cost and performance. Neverthless, we note that other metrics, particularly query plan patterns and execution behaviors, are also essential, especially for downstream tasks such as query optimization. We emphasize that PBench is designed as a highly extensible framework capable of targeting any performance or structural metric presented in the workload traces.

3 AN OVERVIEW OF PBENCH

3.1 Overall Framework of PBench

Figure 3 illustrates the overall framework of PBench, which synthesizes a workload that closely approximates the performance metrics M and operator distributions O of a real workload W trace through the following three stages: (1) **Synthesizer Data Preparation**, where workload components are collected from standard benchmarks. (2) **Two-Phase Workload Synthesis**, which first formulates the synthesis problem as an integer linear programming task and then assigns timestamps to preserve the temporal dynamics of the real workload W. (3) **Workload Component Augmentation**, where an LLM-based module generates additional queries to further enhance approximation effectiveness. Below, we present details of each of the three stages in PBench.

3.1.1 Synthesizer Data Preparation. We select widely-used benchmarks for workload synthesis, including TPC-H, TPC-DS, and YCSB. TPC-H and TPC-DS serve as medium (22 query templates) and heavy (99 query templates) analytical workload components, respectively, while YCSB represents lightweight transactional workload components. To prepare workload components, we consider two key factors: scale factor (*SF*) and *skewness*. The *SF* parameter controls the database size, where a larger *SF* leads to higher resource consumption for queries. The *skewness* parameter controls dataset skewness, influencing query execution patterns. For instance, a higher skew results in high ratios between CPU Time and Scanned Bytes. Once the workload components are prepared, we profile the queries to obtain their corresponding performance features. These features are aggregated at multiple levels to define the objectives for the workload synthesis tool, including: (1) Window-level aggregation to capture overall workload statistics, (2) Interval-level aggregation to reflect finer-grained execution patterns, and (3) Query-level features to model individual query characteristics. For additional details on this stage, please refer to Section 3.2.

3.1.2 Two-Phase Workload Synthesis. Given a target workload trace W, we divide it into multiple time windows and aggregate performance metrics and operator distributions within each window. These aggregated statistics, referred to as *aggregated performance features*, serve as the approximation objectives, such as total CPU time or the number of join operators, as illustrated in Figure 1.

We then propose a *two-phase framework*. In the first phase, we formulate the synthesis task as an optimization problem that selects and combines existing workload components to approximate the performance features of each time window $\Gamma^{(i)}$. To solve the problem, we employ integer linear programming (ILP) to determine the optimal combination of workload components. In the second phase, although the ILP-based synthesis yields a coarse-grained workload for each window, it does not determine the precise temporal distribution of queries. To address this, we introduce a timestamp assignment mechanism, which assigns a timestamp $t_{jk}^{(i)}$ to each instance of a workload component C_j within window $\Gamma^{(i)}$. Since this problem is computationally challenging, we employ a *Simulated Annealing (SA)* algorithm to iteratively refine the assignment and improve the temporal fidelity of the synthesized workload.

The simulated annealing process yields a locally optimal solution to the timestamp assignment problem. The performance gap between the candidate query set and the actual user workload, as addressed in Section 3.1.3, is effectively mitigated by the augmentation phase, which in turn enhances the SA algorithm's ability to converge toward a near-optimal solution. For more details, please refer to Section 4.

3.1.3 Workload Component Augmentation. Simply selecting workload components from existing benchmarks (like TPC-H and TPC-DS) is often insufficient to approximate the performance features of real workloads due to differences between the original workload and the candidate workload. To bridge this gap, we design an LLM-based workload augmenter that automatically generates additional workload components to better approximate the real workload. The high-level idea is to first identifying underperforming time windows, then guiding the LLMs to generate additional components for these time windows, and finally enabling query refinement by injecting expert hints. Selecting an appropriate database for the generated queries poses another challenge. Even though we prepare a set of benchmark databases in advance, discrepancies in data distribution between the generation target and the available databases may hinder the LLM's ability to generate queries with the desired performance characteristics [13]. To address this, we implement a heuristic-driven database selection mechanism that dynamically re-selects or regenerates databases when the LLM fails

to produce a suitable query after multiple attempts. For additional details on this stage, please refer to Section 5.

3.2 Synthesizer Data Preparation

This stage comprises two key processes: workload trace preparation and workload component preparation.

3.2.1 Workload Trace Preparation. The statistics collected from the cloud may be query-level statistics (like *Snowset* or *Redset*) of one specific cluster and database or instance-level time series data which are denoted as $< q_i, F_i, t_i >$, where q_i is inaccessible. When handling query-level statistics, it is required to set a generation granularity and aggregate query-level statistics into performance features $F^{(ij)}$. To approximate target statistics at the time-interval level, we assume a *uniform distribution* of resource consumption over time during query execution. Note that assuming a uniform distribution of resource consumption over time is a simplification and may not capture the actual execution behavior of all queries. We adopted this assumption because fine-grained runtime metrics, specifically, time-series data that show how resources (e.g., CPU time or I/O) are consumed throughout a query's lifecycle, are typically unavailable in real-world workload traces.

To determine an appropriate time window length, we employ an empirical approach to balance generation accuracy and generation efficiency. After finalizing a time window, we aggregate the feature F of all queries running within the time window $\Gamma^{(i)}$ to obtain the corresponding generation target $F^{(i)}$. The aggregation function may vary depending on the indicators or input datasets. For example, as for performance indicators like CPU Time and Scanned Bytes, we can sum up the resource consumption of all queries. For operator distribution, since *Snowset* provides information about operator runtime and *Redset* provides the number of operators for each query, we can aggregate and transform them into statistics using average or sum, thus we obtain the operator distribution of *Snowset* and operator numbers of *Redset*.

3.2.2 Workload Component Preparation. To ensure a diverse set of queries and databases, we first generate benchmark databases with different sizes and data skewnesses. The database size is controlled by the *scale factor* parameter and the data skewness can be generated using five scales of skewness utilizing tools like TPC-H skewed generation tool [4] [23]. The data size primarily impacts the Scanned Bytes; while the skewness will influence the joining fan-out distributions, thereby affecting the intermediate result size and the CPU Time of the queries. To compensate for the difference between the customer's invisible database and the pre-prepared benchmark databases in real-time, we propose a workload component augmentation method (See Section 5).

Remarks. Current single-threaded profiling approach provides only a rough approximation of runtime behavior, especially in scenarios where queries are executed concurrently and interact through shared system resources. Looking forward, we plan to incorporate advanced modeling techniques such as graph neural networks (e.g., as in STAGE [36]) to better capture and generalize complex inter-query and subsystem interactions.

4 TWO-PHASE WORKLOAD SYNTHESIS

4.1 Workload Component Selection Phase

4.1.1 Optimization Objective. To synthesize a workload with real statistics, we combine multiple workload components within a time window, and each selected component can be repeated multiple times as shown in Figure 4. Therefore, the objective is to select and combine multiple workload components to build $\widetilde{W_C}$ such that the total relative error of the performance feature is minimized.

$$\arg\min_{x_j^{(i)}} \sum_{i=1}^{n} \Bigg(\sum_{h=1}^{n_h} \left| \frac{\sum_{j=1}^{v} x_j^{(i)} * m'_{hj} - M_h^{(i)}}{M_h^{(i)}} \right| + \sum_{u=1}^{n_u} \left| \frac{\sum_{j=1}^{v} x_j^{(i)} * o'_{uj} - O_u^{(i)}}{O_u^{(i)}} \right| \Bigg) \quad (1)$$

$$s.t. x_j^{(i)} \in \mathbb{Z}^*, x_j^{(i)} < y, \sum_{j=1}^{v} x_j^{(i)} < z, \sum_{j=1}^{v} x_j^{(i)} * T'_j \leq l$$

The optimization objective is denoted as the above equation, where $x_j^{(i)}$ is the repetition count of the j-th workload component in the i-th time window; $M_h^{(i)}$ and $O_u^{(i)}$ are the aggregated values of the h-th performance metric and the u-th operator distribution in the i-th time window, respectively. Given component C_j, m'_{hj} is the h-th performance metric, and o'_{uj} is the u-th operator distribution; T'_j is the duration of C_j; n_u, n_h, v are the upper bounds of u, h, and j, respectively; y, z, l are three constraint values.

4.1.2 Programming Constraints. To obtain a synthetic workload that is as realistic as possible, we impose the following constraints:

- **Duration constraint.** The total duration of all selected workload components, $x_j^{(i)} * T'_j$, must not exceed the time window length multiplied by the maximum concurrency of the system (usually set to the number of CPU cores). We represent this value as l.
- **Diversity constraint.** PBench needs to guide the workload synthesizer to combine different workload components for better diversity. This is achieved by limiting the number of repetitions of each workload component to a specified value y.
- **Total count constraint.** To avoid generating a workload consisting of massive short queries, PBench limits the total number of workload components used to z (when the count of queries in the customer's workload is known, z can be set accordingly).

4.1.3 Integer Linear Programming. Given the above-defined optimization objectives, constraints, and decision variables, workload component selection can be formulated as an Integer Linear Programming (ILP) problem. The problem is NP-hard, as it can be reduced from the well-known multi-objective multidimensional knapsack problem (MOMKP) [22].

By formulating the ILP problem, we can leverage standard ILP solvers to determine the optimal count of each workload component, thereby synthesizing a workload with realistic statistics.

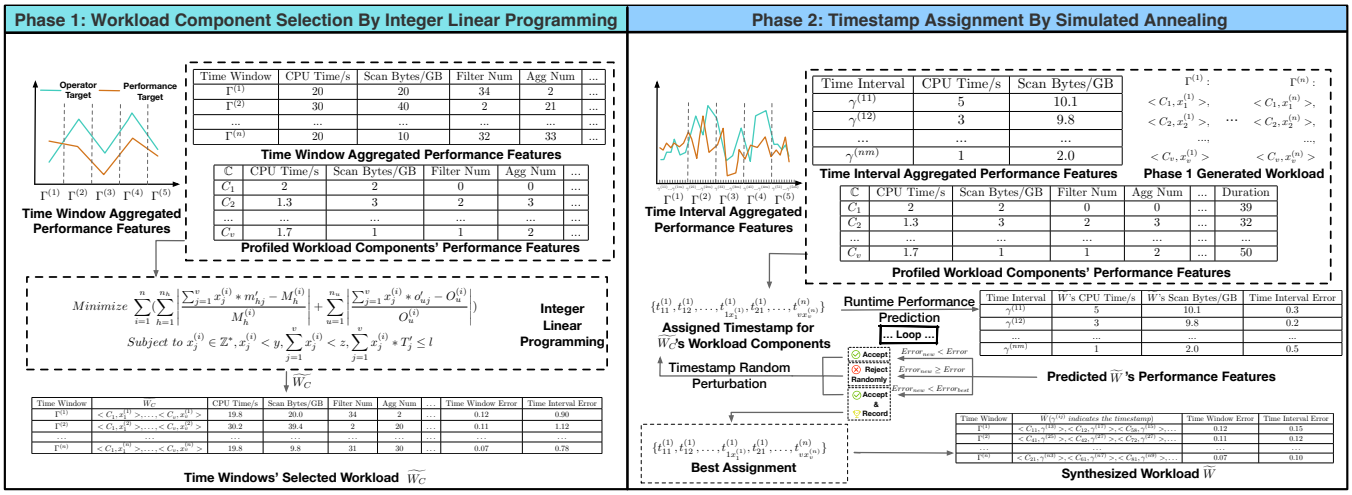


Figure 4: Two Phase Workload Synthesis

4.2 Timestamp Assignment Phase

After the first phase, PBench generates a workload that approximates the performance features within a coarse-grained time window. However, it still faces two key challenges as follows:

First, the ILP-based selection handles the small windows with longer queries. Specifically, the ILP solver operates on disjoint time windows and treats each window in isolation to avoid cross-window interference. This design, however, prevents the inclusion of long-running queries that span across multiple windows. As a result, while the ILP stage is effective for matching workload features at the window level, it lacks the flexibility to capture fine-grained temporal patterns, especially when windows are too small and long-running queries affect the resource usage across multiple windows.

Second, the ILP-based solution focuses only on "*which*" and "*how many*" queries to send in each time window, neglecting "*when*" they actually arrive. Sending queries at the very beginning or utilizing a Poisson distribution or a uniform distribution will all lead to differences compared to the distribution of actual query arrival time, thus limiting the effectiveness of synthetic workloads.

To deal with the issues above, PBench proposes a fine-grained method to assign workload components' timestamps at the interval level, thereby better simulating the temporal dynamics of real workloads. Compared to approximating performance metrics like CPU Time and Scanned Bytes, approximating the operator distribution for each time interval is more challenging. Hence, we do not pursue approximating the operator distribution at the interval level.

4.2.1 Concurrent Processing Assumption. Normally, processing multiple queries in parallel affects the queries' execution time. Therefore, we need to effectively estimate the execution time of each query when multiple queries are executed simultaneously. To address this issue, we employ a lightweight prediction model. Building on the approach in [29], we model the impact of adding a new query with profiled duration T. Specifically, if $\hat{T}$ represents the average duration of P existing queries in the target time interval, the updated average duration T' after incorporating the new query (yielding $P' = P + 1$ total queries) is given by: $T' = \frac{\frac{P'}{P}T + P\left(\hat{T} + \frac{1}{P}T\right)}{P'}$.

As OLAP systems evolve to become more complex and heterogeneous in resource usage, we believe more sophisticated modeling is necessary. Recent studies [36, 37] have proposed GNN-based approaches to better capture such interactions. We plan to integrate these promising directions into our framework in future work.

4.2.2 Simulated Annealing. To achieve a fine-grained generation, we model the timestamp assignment problem as a programming problem, and the optimization goal is to minimize the total error of all time intervals in each time window $\Gamma^{(i)}$. Consequently, PBench develops a simulated annealing-based algorithm to assign the timestamp to each query (i.e., the query's starting time). The algorithm performs in three steps. First, it initializes a timestamp assignment of workload components. Second, it randomly modifies the current assignment by adjusting one workload component's starting time to create a new solution. Third, it recursively checks if the new solution has a lower error. If this is the case, it accepts the current solution with a certain probability to avoid getting stuck in local optima. The probability of accepting a worse solution is controlled by a "temperature" parameter V that gradually decreases as the algorithm progresses. The process repeats for a few iterations, gradually lowering the temperature and refining the solution, until a stopping criterion is met (e.g., no significant improvement over a set number S of steps).

5 WORKLOAD AUGMENTATION WITH LLMS

The effectiveness of our two-phase workload synthesis is highly sensitive to the quality of candidate workload components. When user workloads and candidate components demonstrate similarities in overall resource consumption patterns, performance characteristic ratios, and operator distributions, the synthesis error remains minimal. However, substantial discrepancies in these aspects would result in significant synthesis errors. Specifically, such differences may fall into the following two aspects. (1) *Resource consumption differences*: User workloads demonstrate more diverse resource utilization patterns compared to the fixed resource profiles of benchmark-derived component candidates. (2) *Operator distribution differences*: The CAB benchmark study [32] reveals that

user workloads (Snowset) exhibit a higher prevalence of projection operators and fewer scan/filter operators when compared to TPC-H/TPC-DS benchmarks. Consequently, combining benchmark queries cannot effectively reproduce authentic user workload operator distributions. These discrepancies necessitate the *workload component augmentation* module in PBench.

5.1 Workload Component Augmentation

The augmentation task with new workload components can be formulated as a constraint-aware query generation problem. We formally define the problem as follows: Given target performance features F^G and an underlying database D, the objective is to synthesize a workload component (Q', D) whose execution features F' minimize the approximation error relative to F^G.

We employ large language models (LLMs) to generate queries that match target features F^G (comprising performance metrics m^G and operator requirements o^G). Our approach introduces a version-agnostic prompting strategy decoupled from specific LLM implementations and a modular framework designed for adaptation to future LLM advancements. The framework incorporates three core technical components:

5.1.1 Few-shot Prompting. Because LLMs cannot inherently understand query-performance relationships in zero-shot settings, we provide informative examples for each target F_j^G to enable effective query generation. The similarity between target F_j^G and existing component features F_i' is computed using Euclidean distance calculated as below:

$$Calc\_Sim(F^G, C_j) = 1/(\sqrt{\sum_{h=1}^{n_h}(m'_{hj} - m_h^G)^2 + \sum_{u=1}^{n_u}(o'_{uj} - o_u^G)^2} + 1) \quad (2)$$

We select the top-w most similar or dissimilar components as positive/negative examples E_P/E_N for generation guidance. These examples enable the LLM to learn query patterns from positive examples and avoid query patterns from negative examples. As illustrated in Figure 5 (for target F_1^G), green and brown markers identify positive and negative examples, respectively.

5.1.2 Iterative Refinement. In general, LLMs cannot reliably generate queries meeting target criteria in a single attempt due to their limited understanding of query-performance relationships. We therefore employ an iterative refinement strategy where the LLM progressively generates better-aligned queries based on differences between current outputs and target specifications. Specifically, for each generated query, PBench performs profiling to extract its performance features and quantify the discrepancy from the target specifications. All generated queries are incorporated into the workload component candidate set regardless of error magnitude, as even imperfect queries contribute beneficially to the ILP optimization process. The system iteratively requests regenerations from the LLM until either the generated query achieves the target accuracy or the maximum iteration threshold is reached.

5.1.3 Expert Knowledge Injection. Instead of issuing a generic regeneration request, PBench computes the feature vector differences between generated queries and the target and provides these differentials as explicit feedback to guide the LLM's regeneration.

Table 2: LLM Hints for Query and Data Augmentation

Scenarios	LLM Hints & Database Augmentor Actions
Generated query has lower CPU Time & lower Scanned Bytes	(i) Try to generate a query that performs computation on a larger table. (ii) Try to delete some predicates to scan more data. (iii) Try to add a dimensional table into the query.
Generated query has higher CPU Time & lower Scanned Bytes	(i) Scan more data while deleting some operators. (ii) Use more Inner Join operators to reduce the intermediate result size. (iii) Avoid accessing indexed columns.
Generated query has lower CPU Time & higher Scanned Bytes	(i) Perform arithmetic operations on some columns. (ii) Introduce more subqueries, joins (JOINs), set operations (UNION, INTERSECT, etc.), and complex aggregation operations (GROUP BY, HAVING) while accessing less tables. (iii) Add another dimensional table and do more joins between the fact table and dimensional tables.
CPU Time and Scanned Bytes are lower or higher	Use or generate a benchmark database with a higher or lower Scale Factor.
CPU Time/Scanned Bytes Ratio is lower or higher	Use a benchmark database of higher or lower skewness, or select a database with a more complex schema.

Specifically, to help the LLM make more effective adjustments, we introduce **LLM hints** that are designed with the aid of external expert knowledge. These hints are used to construct high-quality prompts for regenerating queries that better match the desired performance characteristics. Several examples of such hints are illustrated in Table 2. For instance, if a generated query exhibits *low CPU Time and high Scanned Bytes*, PBench guides the LLM to incorporate more subqueries, join operations (e.g., JOIN), set operations (e.g., UNION and INTERSECT), and complex aggregations (e.g., GROUP BY and HAVING), while accessing fewer tables.

Due to the fact that a user's database is typically not visible, PBench may fail to generate suitable queries using the existing benchmark databases. To address this issue, PBench employs a *heuristic-driven database regeneration strategy* when multiple query generation attempts fail. Common error scenarios and corresponding remedies are summarized in Table 2. For example, when both CPU Time and Scanned Bytes are consistently too low or too high, this often indicates that the selected database's scale factor is either too small or too large, thus requiring adjustment. If a suitable pregenerated database with the desired scale factor is not available, a new one must be generated. In another example, when the CPU Time-to-Scanned Bytes ratio remains significantly lower than the target, this may suggest that the database lacks sufficient intermediate result complexity. To mitigate this, we can either use a database with higher skewness or switch to a more structurally complex benchmark, such as moving from TPC-H to TPC-DS, which features richer table relationships and query patterns.

5.2 Workload Augmentation Procedure

The overall workflow for workload augmentation, as shown in Figure 5, consists of the following seven steps.

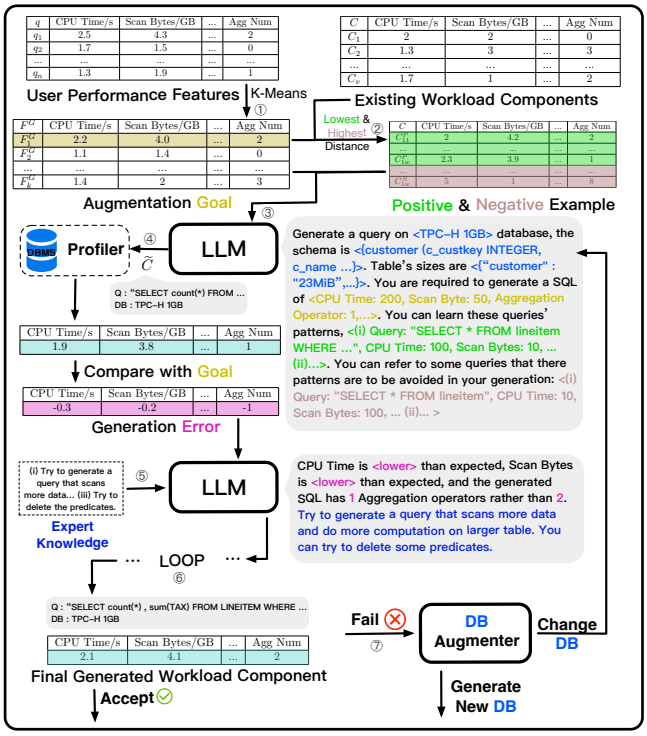


Figure 5: LLM-based Workload Component Augmentation

(1) **Target Generation**: The user workload is clustered to identify representative target feature vectors $\{F_1^G, \ldots, F_k^G\}$ that capture the diversity of workload characteristics.
(2) **Example Retrieval**: Based on the similarity function defined in Equation 2, relevant examples are retrieved from existing query pools, including both similar and dissimilar queries to aid LLM generalization.
(3) **Initial Prompting**: The LLM is prompted with database metadata, representative targets, and selected examples to guide the initial query generation process.
(4) **Real-time Measurement**: The generated queries are executed against the target database to collect performance metrics, enabling empirical evaluation of how closely the queries match the desired characteristics.
(5) **Expert Guidance**: When discrepancies between generated and target features arise, predefined expert rules are introduced as LLM hints to guide subsequent query generation.
(6) **Iterative Refinement**: A feedback loop refines the queries by incorporating real-time performance measurements and expert hints, improving alignment with the target features over multiple iterations.
(7) **Database Augmentation**: If query refinement fails to meet target metrics, the database is augmented by adjusting parameters such as scale factor and data skewness, or by regenerating the dataset to support the desired performance behaviors.

6 EXPERIMENTS

6.1 Experiment Setup

Platform and Implementation. We use the same experiment setting as our baselines. For each server node, we use a Databend [8] 8C16G setting and we leverage Prometheus [28] to collect the metrics. For Redset's query-to-query baseline experiment, we run our evaluation on a 4-node ra3.4xlarge Redshift provisioned cluster and collect metrics via the statistic tables including stl_wlm_query, stl_scan, and stl_explain. We use CBC Solver [14] as our ILP solver and Python SimAnneal Package [34] as our SA solver. We use a state-of-the-art LLM model for the Augmenter (Note that we cannot disclose the model name due to Amazon policy).

Workload Traces. For *Snowset*, we cluster all the traces into five query arrival time patterns based on CAB [32]'s summarization. We randomly select a one-hour trace from each pattern's traces. These selected traces contain both short-term and long-term peak loads, ensuring good representativeness for the entire *Snowset* dataset. All of the five traces are used for ablation experiments. For *Redset*, we randomly select two one-hour traces for window-level experiments, and three 24-hour traces on four nodes for query-level experiments.

Workload Components. We prepare TPC-H, TPC-DS, the Join Order Benchmark (JOB) [18], and YCSB [10] with various scale factors. JOB is a benchmark that contains 113 OLAP queries and a dataset collected from IMDB [17]. As for TPC-H, we use sizes of 500M, 1G, 2G, 5G, 9G. As for TPC-DS, we use sizes of 1G and 2G. As for Redset's 4-node experiments, we use sizes of 1G, 10G, 100G, 1T, and 3T for both TPC-H and TPC-DS.

Evaluation Metrics. For window-level performance evaluation, we use Mean Absolute Error (MAE), Geometric Mean Absolute Percentage Error (GMAPE), and Geometric Mean Q-Error (GMQE) per time window as the evaluation metrics. We use the geometric mean of the relative error metrics instead of the arithmetic mean to avoid the influence of several extremely large relative error values. For window-level (or query-level) evaluation, n denotes the number of time windows (or user queries).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|F^{(i)} - \widetilde{F^{(i)}}\right| \tag{3}$$

$$\text{GMAPE} = \left(\prod_{i=1}^{n}\left|\frac{F^{(i)} - \widetilde{F^{(i)}}}{F^{(i)}}\right| + 1\right)^{\frac{1}{n}} - 1 \tag{4}$$

$$\text{GMQE} = \left(\prod_{i=1}^{n}\max\left(\frac{F^{(i)}}{\widetilde{F^{(i)}}}, \frac{\widetilde{F^{(i)}}}{F^{(i)}}\right)\right)^{\frac{1}{n}} \tag{5}$$

Baseline. We evaluate two baselines as follows: **(1) Stitcher [35]:** It synthesizes workloads by adjusting the sending concurrency and frequency of benchmark queries using Bayesian Optimization(BO) [15]. Since it is not open-sourced, we follow the instructions in [35] and collect training data of various configurations of benchmark suite combinations based on the workload components we use. We train a linear model for each "workload type" (e.g. TPC-H 5G & TPC-DS 1G), and use BO [16, 26]) to find the solution. For each time window, we run 400 iterations of the BO processes to achieve the best accuracy. **(2) CAB [32]:** It creates a query pool and selects queries from the query pool to simulate the user workload's CPU Time. We use all our prepared workload components

queries to construct the query pool for a fair comparison. Since it is open-sourced, we use their code to run the experiments.

Hyper parameter settings. Time window Γ is set to 5 minutes, and time interval γ is set to 30 seconds. For ILP, we set diversity constraint y to 10, and total count constraint z is equal to twice the original user query count. For TA, we set the terminating criterion's no-improvement step number $S = 100$ and we use a SA tool [34] to set the maximum and minimum temperatures.

6.2 Evaluation of PBench on Snowset

Overall Evaluation. Table 3 reports the evaluation results of workload synthesis at the time window level over *Snowset*, reporting the average performance across five workload traces. For simplicity, we omit the GMAPE measurement of approximation errors for operator distributions. The results indicate that PBench significantly outperforms the baselines, achieving up to 6× lower GMAPE for performance metrics and much lower errors for operator distributions. Specifically, PBench achieves a GMAPE of 17.37% for CPU Time and 12.43% for Scanned Bytes. The superior performance of PBench can be attributed to its workload synthesis approach, which effectively simultaneously approximate both performance metrics and operator distributions.

In summary, the results demonstrate that, for real workload traces from *Snowset*, PBench achieves superior approximation of both performance metrics and operator distributions, and significantly outperforming existing methods.

Time Interval Results. We also evaluate the performance of the approaches on time interval results. As shown in Table 4, PBench achieves more than 3× lower GMAPE compared to CAB and Stitcher for CPU Time and Scanned Bytes. Due to the small sample size within each time interval, errors can be influenced by singular values, leading to a higher mean error. Nevertheless, PBench effectively captures peak patterns. Figure 6 compares the original *Snowset* traces with the replayed performance traces of the synthetic workloads. Due to the space limit, we present results for three representative traces, illustrating how well PBench captures real workload dynamics across different workload patterns. PBench consistently achieves better approximation performance across different workload patterns, including high spikes, low peaks, and sustained peaks. In contrast, CAB and Stitcher struggle to approximate certain peaks and exhibit larger errors due to their coarse granularity and limited adaptability.

It is worth noting that PBench may occasionally exhibit slight deviations in peak positions. This is primarily due to inaccuracies in estimating execution times under concurrent query processing, which is a challenge that affects all methods, including the baselines. While we apply a linear slow-down model to approximate these variations, completely eliminating such mismatches is infeasible. Fortunately, as these deviations result only in minor peak shifts or slight peak flattening within a few seconds, it has insignificant impact on the overall performance.

Efficiency Evaluation. We also evaluate the generation time of each method to compare their efficiency. CAB synthesizes workloads within milliseconds, as it simply selects queries randomly without requiring training or prediction. Stitcher incurs minimal time for model training, as it only fits a few linear models, but requires an average of 122 minutes to assemble the workload. In contrast, PBench significantly improves efficiency, taking an average of 6 minutes for ILP and 8 minutes for TA, resulting in a total synthesis time of 14 minutes—making it 8× faster than Stitcher. It is worth noting that for real-time workload synthesis and replay, the synthesis time must remain within the length of the workload trace (60 minutes in this case) to ensure uninterrupted processing. Therfore, PBench meets this requirement, demonstrating its practicality for online workload synthesis and benchmarking.

6.3 Ablation Study of PBench

6.3.1 Evaluation on TA & ILP. Figure 7a and Table 6 shows the results of PBench's w/o Timestamp Assignment (TA) results. Since our baselines have no timestamp assignment step, we randomly send workload components within the time window. It can be seen that when we do not assign timestamps (i.e., performing the first phase only), the fitting result shows a significant mismatch. With timestamp assignment, the results for performance metrics are significantly improved (i.e., 2.4× and 1.6× better GMAPE on CPU Time and Scanned Bytes, respectively). Note that TA has no impact on the operator results as the ratio is fixed in a time window regardless of the queries' order or the length of the time window.

Moreover, to evaluate the effectiveness of the ILP stage, we implement a baseline Greedy that selects the most similar query per time window based on average feature similarity within each window (matching the original workload's query count). As shown in Table 5, the results clearly demonstrate that our ILP-based method achieves 3x-5x better performance across all measured features. This is because Greedy processes each window with a single query variant, while PBench dynamically combines multiple queries per window. This compositional flexibility enables PBench to better approximate the target feature distributions.

6.3.2 Evaluation on Workload Augmentation. Figure 7b shows the comparison results with and without the LLM-based workload components augmenter. It can be observed that the workload component added by LLM significantly improves the fitting accuracy. In particular, as shown in Table 5, compared with ILP+TA, PBench improves not only the accuracy of performance metric fitting but also significantly enhances the accuracy of operator ratio fitting. This improvement stems from the fact that traditional benchmark queries often contain a higher proportion of computational operators, which deviates from the operator distributions observed in real customer workloads. By augmenting the candidate query set with components generated through our LLM-based augmenter, PBench effectively reduces this mismatch, thereby achieving a lower fitting error across both performance and operator-level features.

We also compare PBench with a traditional query generator, SQLSmith [1], which generates queries and then selects those that satisfy the execution constraints. As shown in Table 5, after incorporating these queries into the workload component candidate pool, we find that PBench outperforms SQLSmith in fitting operator ratios by up to 28%, while also achieving better mean absolute error (MAE) performance on CPU Time and Scanned Bytes. In terms of efficiency, we observe that SQLSmith requires over two hours to generate and verify 1,000 queries in order to find 82 executable ones, whereas PBench completes this task in just 43 minutes. These

Table 3: Evaluation of PBench on Snowset

Methods	CPU Time (s)			Scanned Bytes (GB)			Filter Ratio		Aggregate Ratio		Join Ratio		Sort Ratio	
	MAE	GMAPE(%)	GMQE	MAE	GMAPE(%)	GMQE	MAE	GMQE	MAE	GMQE	MAE	GMQE	MAE	GMQE
Stitcher	64.89	110.16	2.27	71.10	43.61	2.71	0.12	1.35	0.12	1.39	0.16	1.02	0.02	1.68
CAB	41.01	25.24	1.25	24.83	45.43	1.86	0.73	5.41	0.73	5.25	0.66	4.26	0.56	5.05
PBench	**22.41**	**17.37**	**1.15**	**3.70**	**12.43**	**1.25**	**0.00**	**1.03**	**0.00**	**1.03**	**0.00**	**1.09**	**0.00**	**1.00**

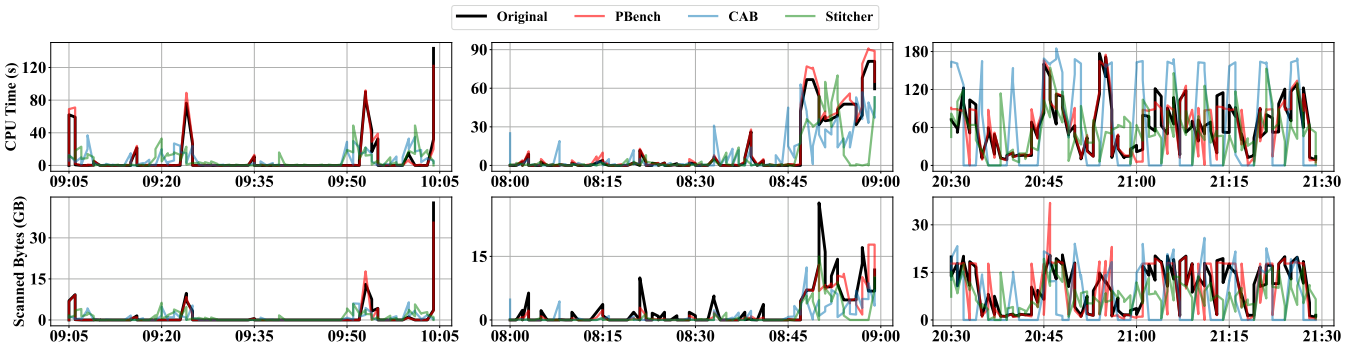
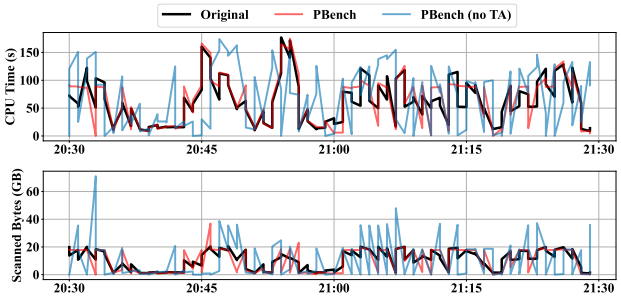


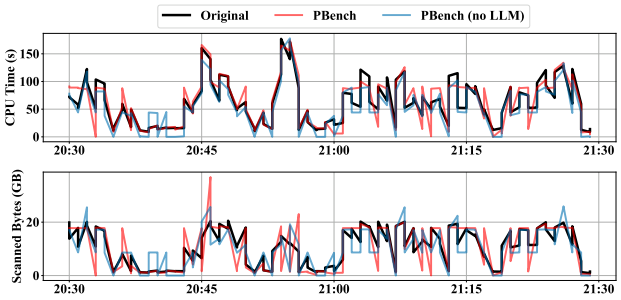
Figure 6: Replayed Traces on Snowset with Different Methods

Table 4: Results on Snowset (Time Interval)

Methods	CPU Time			Scanned Bytes		
	MAE (s)	GMAPE (%)	GMQE	MAE (GB)	GMAPE (%)	GMQE
Stitcher	20.29	163.10	4.41	3.37	82.85	2.71
CAB	32.13	154.18	10.57	4.52	80.57	4.73
PBench	**7.75**	**36.05**	**1.49**	**2.32**	**25.23**	**1.56**



(a) Replayed Traces of w/o Timestamp Assignment on Snowset



(b) Replayed Traces of w/o LLM Augmenter on Snowset

Figure 7: Ablation Study Replayed Traces on Snowset

results highlight the advantages of our hint-guided, LLM-powered query generation approach over traditional query generation methods in both efficiency and fidelity to real-world workloads.

We also provide a more detailed observation that, on average, 88.1% of the queries in the synthesized Snowset workloads across five traces were generated by LLMs. This high proportion is mainly influenced by two factors. First, as more LLM-generated queries are added to the candidate pool, the ILP-based selector has a higher chance of finding suitable matches with the desired performance profiles. Second, when benchmark queries diverge significantly from the target workload in terms of performance characteristics, the ILP solver tends to favor LLM-generated queries, which are explicitly tailored to better reflect the statistical and behavioral features of the original trace.

6.4 Evaluation of PBench on Redset

Figure 8 and Table 7 show the results of PBench and baselines on three Redset traces. Table 7 shows that PBench outperforms baselines as well. For CPU Time, PBench achieves better GMAPE than Stitcher and CAB. As for Scanned Bytes, PBench outperforms Stitcher and CAB in terms of MAE. As for operator numbers like Aggregation and Join number, PBench excels due to its multi-objective optimization and workload component augmenter, thus it outperforms all the baselines and achieves a GMQE of 1.00 for the aggregation number. Figure 8 shows that PBench can fit almost every peak of the customer trace, indicating that the timestamp assignment method works well on Redset. We also evaluate the performance metrics matching accuracy in the interval level. As shown in Table 9, we achieve more than 2.5× higher accuracy in GMAPE than the baselines for CPU Time.

6.5 Evaluation on Extensibility

To demonstrate the extensibility of PBench to new metrics, we conduct an experiment in PostgreSQL 16 to evaluate its ability to match subquery plan structures, which are critical for query optimization. Specifically, we focus on two-way join substructures, capturing both the physical join type and the base tables involved (e.g., a Hash Join between the *orders* and *customer* tables). Since such fine-grained details are absent in Redset and Snowset, we simulate a "user workload" using a random query generator that

Table 5: Ablation Study Results on Snowset

Methods	CPU Time			Scanned Bytes			Filter Ratio		Aggregate Ratio		Join Ratio		Sort Ratio	
	MAE (s)	GMAPE (%)	GMQE	MAE (GB)	GMAPE (%)	GMQE	MAE	GMQE	MAE	GMQE	MAE	GMQE	MAE	GMQE
ILP+TA	**5.51**	31.58	1.90	**2.28**	29.61	1.65	0.65	4.47	0.65	4.35	0.24	1.95	0.28	2.26
SQLSmith	107.41	24.33	1.28	12.01	20.55	1.51	0.08	1.31	0.09	1.28	0.03	1.11	0.01	1.01
Greedy	68.62	41.13	1.48	19.41	53.16	1.71	0.74	5.42	0.74	5.27	0.06	1.10	0.38	2.33
PBench	22.41	**17.37**	**1.15**	3.70	**12.43**	**1.25**	**0.00**	**1.03**	**0.00**	**1.03**	**0.00**	**1.09**	**0.00**	**1.00**

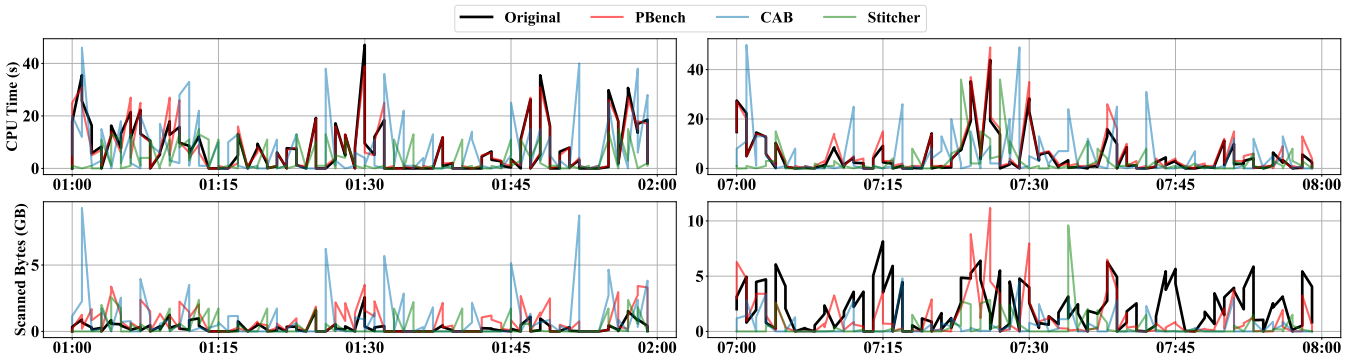


Figure 8: Replayed Traces on Redset

Table 6: Ablation Study of TA Phase (Time Interval)

Methods	CPU Time (s)			Scanned Bytes (GB)		
	MAE	GMAPE (%)	GMQE	MAE	GMAPE (%)	GMQE
ILP+LLM	26.41	175.60	4.63	5.67	98.91	4.22
PBench	**7.75**	**36.05**	**1.50**	**2.32**	**25.24**	**1.57**

samples from the TPC-H query pool. The resulting workload spans one hour, contains 880 SQL queries, and exhibits concurrency levels from 1 to 8. We extract 18 distinct types of two-way joins from this workload. In our setting, we assume access only to aggregated statistics over the 18 join types, without the underlying subquery structures. PBench then serves as an intermediate tool that transforms these coarse-grained summaries into a concrete synthetic workload with rich structural fidelity. As shown in Table 8, the synthetic workload closely matches the distribution of join types in the original, including Hash Join (HJ), Sort-Merge Join (SMJ), and Nested Loop Join (NLJ).

We further demonstrate the utility of the synthesized workload in facilitating query optimization. By enumerating scan and join type hints, we are able to systematically explore alternative execution strategies for subqueries. Our findings reveal that the default physical operators chosen by the optimizer are not always the most efficient. For example, in TPC-H Q3, a Hash Join can be replaced with a Nested Loop Join to achieve better performance under specific hint configurations. This experiment highlights how PBench can reproduce meaningful subquery plans and enable controlled plan variations, thereby supporting physical operator tuning and optimization testing.

6.6 Evaluation on Downstream Task

To evaluate the effectiveness of PBench on database benchmarking, we synthesize workloads from two traces, namely Snowset Trace 4 (The third trace in Figure 6) and Redset Trace 2. We then replay the synthetic workloads on Snowflake and Redshift. Both systems are provisioned with 8 vCPUs to ensure a fair comparison.

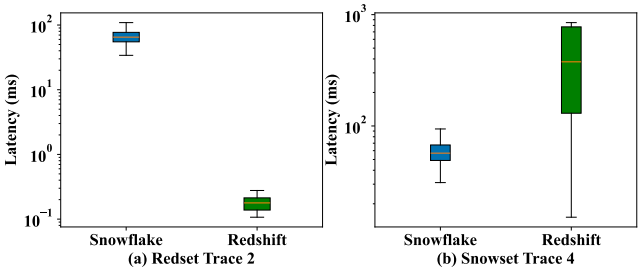


Figure 9: Database Benchmarking using PBench

Snowset Trace 4 is both read- and compute-intensive, and Snowflake is well suited to this workload due to its decoupled storage and compute architecture, which allows independent scaling under heavy load. In contrast, Redset Trace 2 consists of highly concurrent short queries, where Redshift excels due to its AutoWLM [29] that can manage such workloads efficiently.

As shown in Figure 9, the latency distributions on the synthetic workloads derived from Snowset and Redset respectively match these expectations: Snowflake outperforms Redshift on the synthetic workload derived from Snowset Trace 4, while Redshift surpasses Snowflake on the synthetic workload derived from Redset Trace 2. These results indicate that PBench successfully preserves the key performance characteristics of the original traces in the generated workloads, which demonstrates its practical value in database benchmarking and selection, especially when direct access to production workloads is unavailable.

7 RELATED WORK

In the following, we introduce the related work in benchmark-based workload synthesis and constraint-aware query generation.

Benchmark-Based Workload Synthesis. There have emerged several works aiming to synthesize new workloads based on existing benchmarks. Particularly, Wan et al. developed Stitcher [35] to combine entire database benchmarks such as TPC-H's 22 queries

Table 7: Results on Redset

Methods	CPU Time (s)			Scanned Bytes (GB)			Join Num			Aggregate Num		
	MAE	GMAPE (%)	GMQE	MAE	GMAPE (%)	GMQE	MAE	GMAPE	GMQE	MAE	GMAPE	GMQE
Stitcher	18.17	27.09	1.50	8.50	**61.65**	3.42	56.45	337.82	5.28	135.21	74.56	5.81
CAB	8.29	25.70	1.26	10.89	138.75	4.17	187.45	95.52	36.29	55.48	188.43	10.72
PBench	**7.52**	**24.49**	**1.24**	**6.83**	97.16	**2.53**	**0.59**	**6.56**	**1.06**	**0.47**	**0.46**	**1.00**

Table 8: Fitting Results for Two-Way Join Substructures (Total Counts) HJ, SMJ, and NLJ denote Hash Join, Sort-Merge Join, and Nested Loop Join, respectively; pa, li, re, na , su, ps, cu, or are table abbreviations

Workload \ Substucture	HJ (pa & li)	HJ (re & na)	NLJ (re & na)	NLJ (ps & su)	HJ (ps & pa)	HJ (ps & su)	HJ (su & na)	NLJ (su & na)	NLJ (ps & pa)
Original	34	105	39	105	77	32	70	78	39
Synthesized	34	105	39	105	76	32	69	78	39
Workload \ Substructure	**HJ (or & li)**	**NLJ (pa & li)**	**NLJ (cu & or)**	**NLJ (su & li)**	**NLJ (li & li)**	**HJ (cu & or)**	**NLJ (ps & li)**	**SMJ (li & pa)**	**NLJ (or & li)**
Original	174	75	80	75	78	81	39	47	83
Synthesized	174	75	80	72	78	81	39	47	83

Table 9: Results on Redset (Time Interval)

Methods	CPU Time (s)			Scanned Bytes (GB)		
	MAE	GMAPE (%)	GMQE	MAE	GMAPE (%)	GMQE
Stitcher	6.03	197.68	8.24	1.21	90.94	4.09
CAB	7.41	230.68	7.41	1.39	128.87	4.67
PBench	**1.43**	**77.07**	**1.82**	**1.06**	**85.87**	**2.64**

and YCSB [6]'s workload. The system comprises a predictor, a generator, and an integrator, with linear regression models trained for each benchmark combination to predict performance metrics. Renen et al. [32] introduced the CAB tool for synthesizing analytical workloads, which sets database scales based on log-normal distributions. CAB allocates CPU time budgets proportional to the actual sizes of the databases. It schedules query arrivals by segmenting the benchmark execution into time periods, where CPU time consumption is proportional to the number of query requests. Unfortunately, these works cannot achieve high accuracy as they employ a coarse-grained fitting strategy and neglect the operator ratio, while PBench develops a fine-grained method to consider both performance metrics and operator distributions.

Constraint-Aware Query Generation. This line of research focuses on generating SQL queries that satisfy specified cardinality constraints over given database tables. A widely adopted approach involves using traditional query generators, such as SQLSmith [1], to produce a large pool of queries, from which valid candidates are subsequently selected. While this method is straightforward, it suffers from a low success rate and incurs significant computational overhead in verifying constraint satisfaction. Zhang et al. [42] introduce a cardinality-aware query generation framework that models the task as a sequential decision-making process and applies reinforcement learning (RL) techniques to optimize it. Mueller et al. [24] further contribute by developing novel query representations and predicate selection mechanisms based on feature vectors. However, RL-base and DL-based methods remain computationally expensive and challenging to train [21], and they often overlook system-level performance metrics such as CPU time or scanned bytes.

8 CONCLUSION AND FUTURE WORK

In this paper, we proposed PBench, a workload synthesizer that constructs synthetic workloads by strategically selecting and combining workload components from existing benchmarks. Particularly, PBench incorporates effective techniques for component selection and timestamp assignment, and leverages an LLM-enhanced approach for component augmentation. Extensive experiments on real workload traces demonstrate that PBench significantly reduces approximation errors compared to state-of-the-art methods.

In the future, we would like to deal with several open problems:

(1) Environment Adaptation. The current version of PBench requires that workload synthesis is performed in an environment similar to that of the target user. To relax this constraint, we plan to explore environment adaptation techniques that enable synthesis even under mismatched conditions. One promising direction is to develop a performance predictor that estimates a query's performance features in a new execution environment, thereby bridging the gap between environments.

(2) OLTP Extension. This work primarily focuses on OLAP-oriented workloads. In future work, we aim to investigate the applicability of our approach to OLTP-oriented workloads, which pose unique challenges due to stronger inter-query dependencies (e.g., locks, transactions, and frequent updates). Thus, it is challenging to perfectly fit the performance metrics in such cases. One promising direction is to incorporate a more sophisticated inter-query interaction module for capturing complex runtime dynamics.

(3) Distribution-Aware Data Generation. Currently, PBench relies heavily on existing benchmark databases. However, when user workloads operate on unseen databases with significantly different data distributions, synthesis accuracy can drastically degrade. To address this limitation, we plan to develop distribution-aware data generation techniques that can construct synthetic databases matching the statistical properties of the target workload's data, even in the absence of the original dataset.

ACKNOWLEDGMENTS

This work was supported by the NSF of China (62436010 and 62441230), the DARPA ASKEM program (award HR00112220042), the ARPA-H Biomedical Data Fabric project, the MIT DSAIL Project, grants from Liberty Mutual and Google, the United States Air Force Research Laboratory, the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000.

REFERENCES

[1] Seltenreich Andreas, Tang Bo, and Mullender Sjoerd. 2022. SQLsmith. https://github.com/anse1/sqlsmith. 2025/04/10.

[2] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J Green, Monish Gupta, Sebastian Hillig, et al. 2022. Amazon Redshift re-invented. In *Proceedings of the 2022 International Conference on Management of Data*. 2205–2217.

[3] Jinsheng Ba and Manuel Rigger. 2024. Keep It Simple: Testing Databases via Differential Query Plans. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.

[4] Peter Boncz, Angelos-Christos Anatiotis, and Steffen Kläbe. 2018. JCC-H: Adding Join Crossing Correlations with Skew to TPC-H. In *Performance Evaluation and Benchmarking for the Analytics Era*, Raghunath Nambiar and Meikel Poess (Eds.). Springer International Publishing, Cham, 103–119.

[5] Peter Boncz, Thomas Neumann, and Orri Erling. 2013. TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark. In *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 61–76.

[6] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[7] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. 2016. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*. 215–226.

[8] Databend. 2023. *TPC-H Benchmark: Databend Cloud vs. Snowflake*. Technical Report. Databend. https://docs.databend.com/guides/benchmark/tpch Accessed: 2023-11-20.

[9] Shaleen Deep, Anja Gruenheid, Kruthi Nagaraj, Hiro Naito, Jeff Naughton, and Stratis Viglas. 2021. Diametrics: benchmarking query engines at scale. *ACM SIGMOD Record* 50, 1 (2021), 24–31.

[10] Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Röhm. 2014. YCSB+ T: Benchmarking web-scale transactional databases. In *2014 IEEE 30th International Conference on Data Engineering Workshops*. IEEE, 223–230.

[11] Haowen Dong, Chao Zhang, Guoliang Li, and Huanchen Zhang. 2024. Cloud-Native Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[12] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H choke points and their optimizations. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1206–1220.

[13] Ju Fan, Tongyu Liu, Guoliang Li, Junyou Chen, Yuwei Shen, and Xiaoyong Du. 2020. Relational Data Synthesis using Generative Adversarial Networks: A Design Space Exploration. *Proc. VLDB Endow.* 13, 11 (2020), 1962–1975.

[14] John Forrest and Robin Lougee-Heimer. [n. d.]. CBC User Guide. https://www.coin-or.org/Cbc/. 2024/12/13.

[15] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[16] Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. 2014. Bayesian optimization with inequality constraints.. In *ICML*, Vol. 2014. 937–945.

[17] IMDb. 2024. IMDb Non-Commercial Datasets. https://developer.imdb.com/non-commercial-datasets/. 2024/12/13.

[18] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[19] Viktor Leis and Maximilian Kuschewski. 2021. Towards cost-optimal query processing in the cloud. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1606–1612.

[20] Guoliang Li, Haowen Dong, and Chao Zhang. 2022. Cloud databases: New techniques, challenges, and opportunities. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3758–3761.

[21] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. Machine learning for databases. In *Proceedings of the First International Conference on AI-ML Systems*. 1–2.

[22] Thibaut Lust and Jacques Teghem. 2010. The multiobjective multidimensional knapsack problem: a survey and a new approach. arXiv:1007.4063 [cs.DM] https://arxiv.org/abs/1007.4063

[23] Microsoft. 2024. DSB benchmark. https://github.com/microsoft/dsb. 2024/12/13.

[24] Magnus Müller, Lucas Woltmann, and Wolfgang Lehner. 2023. Enhanced Featurization of Queries with Mixed Combinations of Predicates for ML-based Cardinality Estimation.. In *EDBT*. 273–284.

[25] Vikram Nathan, Vikramank Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan (Murali) Narayanaswamy, and Tim Kraska. 2024. Intelligent scaling in Amazon Redshift. (2024). https://www.amazon.science/publications/intelligent-scaling-in-amazon-redshift

[26] Fernando Nogueira. 2014–. Bayesian Optimization: Open source constrained global optimization tool for Python. https://github.com/bayesian-optimization/BayesianOptimization

[27] Meikel Poess, Raghunath Othayoth Nambiar, and David Walrath. 2007. Why You Should Run TPC-DS: A Workload Analysis.. In *VLDB*, Vol. 7. 1138–1149.

[28] Prometheus. 2024. Prometheus Github Page. https://github.com/prometheus. 2024/12/13.

[29] Gaurav Saxena, Mohammad Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan (Murali) Narayanaswamy. 2023. Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift. In *Companion of the 2023 International Conference on Management of Data* (Seattle, WA, USA) *(SIGMOD '23)*. Association for Computing Machinery, New York, NY, USA, 225–237. https://doi.org/10.1145/3555041.3589677

[30] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.

[31] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC is not enough: An analysis of the Amazon Redshift fleet. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3694–3706.

[32] Alexander Van Renen and Viktor Leis. 2023. Cloud analytics benchmark. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1413–1425.

[33] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building an elastic query engine on disaggregated storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 449–462.

[34] Richard J. Wagner. 2024. Python module for simulated annealing. https://github.com/perrygeo/simanneal. 2024/12/13.

[35] Chengcheng Wan, Yiwen Zhu, Joyce Cahoon, Wenjing Wang, Katherine Lin, Sean Liu, Raymond Truong, Neetu Singh, Alexandra M. Ciortea, Konstantinos Karanasos, and Subru Krishnan. 2023. Stitcher: Learned Workload Synthesis from Historical Performance Footprints. In *EDBT*. OpenProceedings.org, 417–423. https://doi.org/10.48786/EDBT.2023.33

[36] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. arXiv:2403.02286 [cs.DB] https://arxiv.org/abs/2403.02286

[37] Ziniu Wu, Markos Markakis, Chunwei Liu, Peter Baile Chen, Balakrishnan Narayanaswamy, Tim Kraska, and Samuel Madden. 2025. Improving DBMS Scheduling Decisions with Fine-grained Performance Prediction on Concurrent Queries–Extended. *arXiv preprint arXiv:2501.16256* (2025).

[38] Chao Zhang, Guoliang Li, Leyao Liu, Tao Lv, and Ju Fan. 2025. CloudyBench: A Testbed for A Comprehensive Evaluation of Cloud-Native Databases. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2535–2547.

[39] Chao Zhang, Guoliang Li, and Tao Lv. 2024. HyBench: A New Benchmark for HTAP Databases. *Proceedings of the VLDB Endowment* 17, 5 (2024), 939–951.

[40] Chao Zhang, Guoliang Li, Jintao Zhang, Xinning Zhang, and Jianhua Feng. 2024. HTAP Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[41] Huanchen Zhang, Yihao Liu, and Jiaqi Yan. 2023. Cost-Intelligent Data Analytics in the Cloud. *arXiv preprint arXiv:2308.09569* (2023).

[42] Lixi Zhang, Chengliang Chai, Xuanhe Zhou, and Guoliang Li. 2022. Learnedsqlgen: Constraint-aware sql generation using reinforcement learning. In *Proceedings of the 2022 International Conference on Management of Data*. 945–958.