Towards a Benchmark for Learned Systems

Laurent Bindschaedler*, Andreas Kipf*, Tim Kraska*, Ryan Marcus*, and Umar Farooq Minhas†
*Data Systems and AI Lab, Massachusetts Institute of Technology
{bindscha, kipf, kraska, ryanmarcus}@mit.edu
†Database Group, Microsoft Research, Redmond
ufminhas@microsoft.com

***Abstract*—This paper aims to initiate a discussion around benchmarking data management systems with machine-learned components. Traditional benchmarks such as TPC or YCSB are insufficient to analyze and understand these *learned systems* because they evaluate the performance under a stable workload and data distribution. Learned systems automatically specialize and adapt database components to a changing workload, database, and execution environment, thereby making conventional metrics such as average throughput ill-suited to understand their performance fully. Moreover, the standard cost-per-performance metrics fail to account for essential trade-offs related to the training cost of models and the elimination of manual database tuning. We present several ideas for designing new benchmarks that are better suited to evaluate learned systems. The main challenges entail developing new metrics to capture the particularities of learned systems and ensuring that benchmark results remain comparable across many deployments with wide-ranging designs.**

***Index Terms*—benchmark, learned system, instance-optimized system, data management, machine learning, metrics, cost of ownership**

I. Introduction

Conventional benchmarks aim to evaluate the average performance of a data management system in a specific scenario with a fixed workload and data distribution, usually based on synthetic data generators [1]–[7]. Benchmark specifications generally mandate a fixed deployment environment and configuration for the system under test, including stable workloads, fixed datasets, and standard metrics. Accordingly, these benchmarks reflect the average performance and total ownership cost of a static system in a well-defined, predetermined scenario.

Recently, data management systems have undergone a paradigm shift where the use of machine learning models offers performance benefits by synthesizing optimized components [8]–[10] or tuning configuration "knobs" [11]–[13] for a given use case. For instance, reinforcement learning techniques can adaptively pick the best query execution plan under changing workloads or database conditions [14]–[16], optimized data structures can be automatically synthesized from first principles [10], and approximate search heuristics allow constructing efficient data storage layouts [17]. These new *learned* systems (sometimes also referred to as *instance-optimized*) allow specialization to a given workload and data distribution and adaptability to changing conditions unmatched by traditional systems designed for general-purpose usage [18]. These benefits come at the cost of learning (and maintaining) the models used by individual components and involve new trade-offs for conventional database administration practices.

In this paper, we argue that existing benchmarks lack flexibility and do not sufficiently represent real-world workloads and, therefore, are insufficient to analyze and compare learned systems properly. Fundamentally, learned systems are dynamic and adaptive in nature, and their performance characteristics cannot be summarized in a single average performance score. Consequently, we propose designing new benchmarks that better fit the characteristics of learned systems. Doing so entails new challenges to design scenarios that reflect the dynamic behavior of real-world applications and to make the benchmark results comparable across widely different learning techniques and system designs.

A first difference is that new benchmarks should not only use fixed workload and data distributions to evaluate learned systems. Instead, these distributions should better resemble evolving real-world data and be hard to predict, e.g., varying dataset size and lifetime, fluctuations in query load and concurrency, complex diurnal patterns, or growing data skew over time [19]–[21]. The benchmark should then measure and report on the ability of a system to specialize to a given workload and data distribution, as well as its adaptability to changes. Therefore, new metrics are required to complement existing ones. Such metrics should focus on descriptive throughput statistics and outliers, along with throughput and latency during transitions between distributions. It is also desirable to capture the time a system takes to adapt to a new workload.

A prevalent problem is overfitting a system to the benchmark, leading to significant performance improvements that do not translate to real-world scenarios. While this issue is already present in traditional systems and benchmarks, it becomes more severe in the case of learned systems due to their inherent ability to specialize in specific situations.

Additionally, new benchmarks should embrace model training as a first-class citizen, equally important as actual execution. Since better models usually result in better performance during execution, the training time and cost should be part of the reported benchmark results. Moreover, these new training metrics should be put into perspective and made comparable across several systems. As learned systems enable automation of manual tuning by database administrators, a benchmark should compare model training costs to manual optimizations.

There is no benchmark designed to evaluate data management systems with learned components in dynamic scenarios to the best of our knowledge. OLTP-Bench supports evolving workload mixing and access patterns but not varying data distributions [22]. Likewise, the Under Pressure Benchmark introduces shifting workloads in the context of detecting failures in distributed databases [23].

We make the following contributions in this paper:

- We provide an overview of learned systems and state their main characteristics (§II).
- We discuss various problems with existing benchmarks and explain why they are a bad fit (§III).
- We list new requirements for benchmarking learned systems (§IV).
- We present some initial ideas to build a new benchmark, including changes to the configuration and execution, as well as some new metrics to complement the results (§V).

II. Learned Systems

Traditional data management systems are engineered for general-purpose data storage and access. This generality, which results from the lack of runtime adaptation, limits these systems from leveraging specific characteristics of the workload or data distribution for optimization purposes. On the opposite end of the spectrum lie custom-built systems that are highly optimized for a given application and dataset.

Learned systems attempt to bridge the performance gap between general-purpose data management systems and specialized solutions through machine learning techniques. A learned system can automatically synthesize optimized database components such as indices or query optimizers by learning models for the data distribution or the workload. Moreover, such a system can automatically adapt to perceived changes.

This learning-based approach presents a radical paradigm shift from traditional optimization. We believe that the coming years will see the commercialization of several learned databases that instantiate all components and dynamically tune their interactions at runtime, thereby truly mimicking specialized systems. Today, the state-of-the-art focuses on learning models to synthesize specific database components such as query optimizers, query execution operators, and indices. In the remainder of this section, we provide a brief overview of these approaches. The interested reader may consult [9] for a more detailed description.

Query optimization is an excellent candidate for learned approaches due to the complexity of conventional optimizers [24]. Different approaches are possible: learning cardinality estimates [25]–[29], learning to tune an existing query optimizer [14], or even learning a complete optimizer [15]. A popular technique relies on deep reinforcement learning to generate optimized query execution plans on-the-fly, improving with each incoming query. Reinforcement learning can also help optimize scheduling policies in specific workloads [30].

Similarly, models can improve query execution performance. For example, a cumulative distribution function (CDF) model allows fast sorting by placing the data records in roughly sorted order and then running a quick touch-up pass to get the final correct order [31]. A similar CDF approach can be used for joins where the model allows to skip over data records that will not join.

Learned indices improve data access performance by fitting a model over the data to capture the distribution's characteristics. Such models are often arranged in a tree, with the prediction of a model being used to pick a more specialized model recursively until the leaf model makes a final prediction [8]. Learned indices have been shown to outperform B+ trees in various situations, and several improvements have recently been proposed to add support for compression, updates, and multi-indexing [32]–[35].

Many other potential opportunities for machine learning optimization are actively explored at the time of writing, including approximate query processing, predictive models training, learning-based caches, and insert/update optimization. Similarly, other approaches that focus on tuning existing components rather than synthesizing new components from scratch have shown promise [11]–[13].

Many of these techniques provide up to orders-of-magnitude better performance under laboratory conditions. However, it remains largely unclear how these numbers will hold up in more realistic production environments. At the same time, practitioners are wary about adapting and including these techniques into mainstream systems due to a lack of evidence of how they would perform under varying conditions. We believe the only way to address these concerns properly is through better benchmarking.

III. Problems with Existing Benchmarks

Traditional benchmarks for data management systems imitate the behavior of specific workloads or applications to assess the performance of the system under test, generally a key-value store or a relational database [1]–[5]. Most benchmarks make no assumptions about the technologies or software used by the system under test. They test an entire data management system end-to-end, as actual end-users will see it. However, these benchmarks are unsuitable for evaluating and understanding systems with learned components and may cause users to draw improper conclusions. Some benchmarks even forbid systems from leveraging workload knowledge (e.g., prematerializing data structures or indexes). In the remainder of this section, we outline several shortcomings of existing benchmarks.

A. *Fixed Workload and Database*

Benchmarks generally simulate well-defined scenarios with a stable and deterministic query workload on a database whose data distributions remain mostly fixed throughout execution. In contrast, many real-world deployments exhibit a wide range of behaviors such as evolving workloads, diurnal query patterns, temporary bursts in query load or concurrency, changing data distributions and dataset size, or varying degrees of skew that classical benchmarks rarely capture [19]–[21]. As a result, multiple separate benchmarks are often required to exercise a system under various situations and achieve sufficient coverage of possible cases. Since learned systems are, by definition, adaptable, it is highly desirable to benchmark them in a single execution run where the transitions between different situations can be captured and analyzed. Additionally, proper comparisons between learned and traditional systems should not rely on fixed workloads and databases, as their characteristics are easy to learn. For instance, a learned system may gain a significant performance edge when presented with predictable data distributions.

➥ **Lesson 1: Abstain from fixed workloads and databases as their characteristics are easy to learn.**

B. Average Performance

The primary metric used by most existing benchmarks is the throughput that the system under test can handle (number of queries handled per unit of time) during execution. In the context of a fixed workload, data distribution, and execution environment, the average throughput provides a reasonably good characterization of a system's performance. However, the average performance "hides" too much information to properly portray a system's performance in a dynamic execution environment and subject to an evolving workload and data distribution. Moreover, comparing two learned systems based on their average performance during execution is inadequate. For example, one system may perform better on average than the other but fail to respond to any queries during transition periods or suffer from excessive throughput variance on some workloads, making it potentially less adaptable.

➥ **Lesson 2: Average metrics do not capture adaptability.**

C. Model Training

Conventional benchmarks generally load data into the database and immediately execute for a fixed duration or number of queries, reporting the system's performance during the entire execution. This execution mode is incompatible with many learned systems that require model training either in a separate first phase or during the actual execution. The benchmark should not only support training phases, but it should also measure and report on the training as a first-class result to enable a more meaningful comparison between learned systems. Furthermore, existing benchmark results cannot easily be used to compare learned systems with traditional data management systems in terms of training time versus database administration work.

➥ **Lesson 3: Training must be a first-class result.**

D. Total Cost of Ownership

The second metric often used by traditional benchmarks is the cost-per-performance ratio derived from the average throughput and the system's total cost of ownership. The total cost of ownership (TCO) is a dollar value estimate of the total expenditures involved in running a particular system, including hardware, software, maintenance, and administration for a specific duration (typically three years). Several problems arise from using the total cost of ownership in the context of learned systems. First, similar to traditional benchmarks, the TCO is hard to estimate and is often disregarded by researchers. However, for learned systems, the TCO becomes even more relevant because it is a fundamental part of the value proposition of learned systems, i.e., reducing lead time and avoiding maintenance and administration. Second, as discussed previously, in the presence of an evolving workload or data distribution, the average throughput is an improper performance characterization. Therefore, the cost-per-performance ratio may be insufficiently informative.

➥ **Lesson 4: We cannot ignore the human cost anymore.**

IV. New Benchmark Requirements

As discussed in §III, data management systems with learned components are hard to benchmark. Furthermore, developers and operators of learned systems are confronted with new trade-offs regarding training and cost. Therefore, the purpose of a new benchmark for learned systems is to address these limitations. This benchmark should help developers compare systems (whether learned or traditional) and choose the right trade-offs in terms of models, configuration, hardware, and cost for their systems.

The key benefits of data management systems with learned components are specialization and adaptability (§II). These benefits are achieved differently for different database components and likely vary significantly across systems. For example, different systems may use different machine learning techniques to index data or use the same techniques but with different parameters. Similarly, some learned components may have limitations or perform much better in a limited subset of workloads or data distributions. Some learned components require a separate training phase to learn a model, while others can learn online during benchmark execution but possibly impacting the query response time or overall throughput. Training time may also vary significantly depending on available hardware resources. Finally, overfitting to a specific execution may provide significant performance benefits at the expense of generalizability, e.g., if the data distribution suddenly changes.

A new benchmark should support execution with varying workload and data distributions without imposing architectural, configuration, or runtime constraints. Doing so is challenging because, to date, there is no consensus on the type of interfaces provided by learned systems or constraints on their behavior during workload or database transitions. The benchmark should also be agnostic to the differences across systems yet capture enough relevant metrics to understand the qualities and shortcomings of a given system. Finally, it should also provide a factual basis for comparing several systems, whether they be learned systems or a mix of learned and traditional systems.

A significant effort may be required for designers or users of learned components to collect and curate data labels for training. For instance, supervised learned cardinality estimation requires collecting the real cardinalities to build a regression model [36]. Ground truth data can be obtained either during query execution (e.g., collecting all cardinalities of the executed sub-plans) or in a separate training phase. The benchmark should attempt to measure these costs and include them in the results.

A key requirement for a learned system benchmark is the availability of datasets and workloads that are as representative as possible of real-world scenarios. This requirement may be stronger than for traditional benchmarks due to the increased focus on specialization and adaptability. Ideally, the new benchmark should include several high-quality datasets and workloads to allow for a thorough evaluation of systems.

While the conventional metrics that make up the results of existing benchmarks remain relevant, a new benchmark should combine these metrics with new metrics that incorporate learning-specific features and behaviors. Similarly, the new benchmark should also support execution on fixed databases as a baseline. However, it is necessary to provide more flexibility to assess the system in different, complementary areas to understand its performance better. This task is arduous because it involves summarizing performance characteristics that are intrinsically dynamic in a single number. As a result, a new benchmark should strive to augment traditional average metrics with additional dimensions that capture dynamic behaviors.

V. Ideas for a New Benchmark

This section presents some initial ideas towards designing new benchmarks that are better suited for learned systems. We first provide an overview of what such benchmarks should look like in this context. Then, we discuss some necessary changes in benchmark configuration and execution to support learned systems. We also briefly address the problem of obtaining high-quality datasets for benchmarking. Finally, we propose new metrics that are relevant in this context.

A. Overview

A new benchmark for data management systems with learned components should not be based on a specific real-world scenario but rather provide a common framework for executing different scenarios, each with its own workload and data distributions. This common framework should test the complete end-to-end system using a benchmark driver to generate load and measure performance. The benchmark driver should ideally run on a separate machine and connect to the system under test over a fast network connection.

A fundamental difference of this new benchmark compared to traditional benchmarks is the introduction of new metrics. While traditional benchmarks focus on average throughput, a benchmark for learned systems should also analyze a system's ability to adjust to varying workload and data distributions, its capacity to adjust quickly and without significant performance degradation, and the potential performance benefits of longer model training time. Accordingly, the benchmark should make it possible to vary the workload and data distribution during a single benchmark run based on user-defined parameters.

Learned systems are highly effective at specializing components to a given workload and data distribution or to a sequence of different workload and data distributions. However, it is desirable to avoid overfitting models to the benchmark. Therefore, we propose to include hold-out workload and data distributions that the system is only allowed to execute once. In doing so, the benchmark could measure out-of-sample performance. A possible approach to achieve this is to deploy the benchmark as a cloud service and evaluate systems on behalf of users. The use of this benchmark-as-a-service could be a requirement for inclusion in official benchmark results.

B. Configuration and Execution

In addition to the typical benchmark configuration options (e.g., database scale factor, mixes of query streams), we suggest that the benchmark include settings for configuring execution with different workload and data distributions as well as setting the training time and associated resource overhead.

A benchmark can vary workload and data distributions in different ways that should be configurable. First, a workload can slowly transition to another or transition abruptly. The type of transition can impact performance and adaptability in non-obvious ways. Second, such transitions may occur as part of a single execution phase or as two separate execution phases with possible retraining of the models in-between. Finally, the benchmark must make it possible to define how many different workload and data distributions to use and in which order they should be executed.

The benchmark should include configuration options for model training. Learned systems generally offer better performance with more extended training, making the training time or the training data important components of the benchmark. Similarly, users should be allowed to configure whether to use specialized hardware or the fraction of system resources to dedicate for online training.

C. Data and Workload

A learned system benchmark has a clear need for high-quality datasets and workloads representing real-world conditions. At a minimum, we should provide guidelines for selecting databases and workloads along with the benchmark. The benchmark could also include a software tool that evaluates the quality and relevance of a given dataset for the benchmark. For example, this tool could attribute low marks to uniform data distributions and workloads while favoring datasets exhibiting skew or varying query load.

Ideally, the new benchmark should incorporate data and workloads from a variety of production environments. Unfortunately, most companies are not willing or allowed to share such data freely, and, therefore, it may prove impossible to include such datasets in a public benchmark. Privacy-preserving techniques [37]–[39] may provide a path forward but have so far failed to convince companies to share large subsets of their data with the research community. While there is no substitute for the richness of real-world data, synthetic databases and workloads may be sufficient to evaluate learned systems. For instance, a table column containing email addresses could be replaced by a synthetic email address generator that provides a similar data distribution without adversely affecting the outcome in many cases. Therefore, an interesting avenue for a new benchmark involves automatically generating synthetic datasets and workloads from real-world deployments, e.g., using machine learning techniques. Such a synthesizer would have to extend existing approaches [40], [41] to achieve a high-fidelity reproduction of real-world conditions.

D. Metrics

We propose several new metrics that focus on the specific characteristics of learned systems: component specialization, adaptability, and training cost. These metrics are intended to complement traditional metrics such as average performance and cost-per-performance to evaluate the properties of learned systems better. Figure 1 sketches out the different metrics by providing example results. In the remainder of this section, we present and discuss these metrics in more detail.

1) Specialization: Learned systems can automatically synthesize optimized components based on the workload and data distribution. Therefore, a key metric of the new benchmark should provide a breakdown of performance for different workload and data distributions.

We propose to report throughput for each combination of workload and data distribution. However, instead of only reporting the average throughput, the benchmark should report descriptive statistics (e.g., using a box plot) to adequately capture the specialization and adaptation capabilities of the system under test (SUT). Moreover, we suggest providing an estimate of how *far* workload and data distributions differ from each other. Similarity across workloads can be estimated, for example, using the Jaccard similarity [42] between the sets of all subtrees of the query tree for all queries in the workload. Likewise, similarity across data distributions can be evaluated using, e.g., the Kolmogorov–Smirnov test [43] or the Maximum Mean Discrepancy [44]. Figure 1a shows an example where we select the first workload or data distribution as a baseline. Importantly, the similarity values, represented by the function Φ, across the X-axis need not be precise, and it should be sufficient to sort the results by Φ value.

Plots such as the one in Figure 1a can be used to answer several questions about a learned system. When comparing two or more workload and data distributions, the box plots provide a good overview of the dispersion, skewness, and outliers in each case. Therefore, it illustrates how well the SUT performs in a given situation and whether it manages to maintain a stable throughput. This type of plot also supports evaluating out-of-sample performance by plotting hold-out distributions alongside a baseline distribution.

2) Adaptability: In addition to measuring the performance of a SUT in a given situation, it is desirable to report how well and how fast the SUT can adapt to changes in the workload and data. Adaptability concerns both the throughput variations and the delays in query response time resulting from a distribution change. For example, throughput could temporarily decrease due to the CPU overheads of retraining a model. Similarly, query latency could increase due to suboptimal decisions taken by a learned query optimizer.

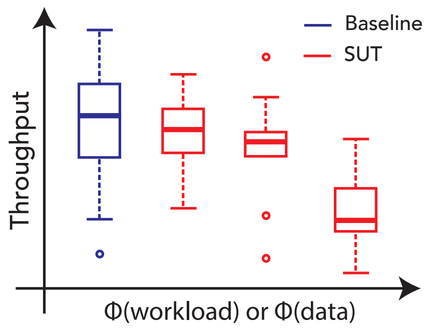
We suggest reporting throughput variations by plotting the cumulative queries completed over time. In this case, the slope of the curve is the throughput, and it is easy to see the impact of a change. Figure 1b shows an example where the SUT starts slow and later catches up. We can derive a single-value result from this plot by computing the area difference between an ideal system with a constant throughput. Similarly, we can easily compare two systems by plotting them on the same chart. When comparing two systems, the area difference between the two systems provides a single-value result.

We also propose to report query latency bands at, e.g., 1-second or 10-second intervals throughout execution. Each query latency band represents the number of completed queries within the interval (throughput), split into two categories depending on whether the query finished within the allotted Service-Level Agreement (SLA) time. An example is provided in Figure 1c. A low number of completed queries or a high number of queries with an SLA violation (red) following a distribution change indicates slow adjustment speed. This metric is highly dependent on the choice of the SLA threshold. Therefore, the SLA threshold should ideally be determined based on a baseline system's query latency statistics on the same hardware and workload distribution.

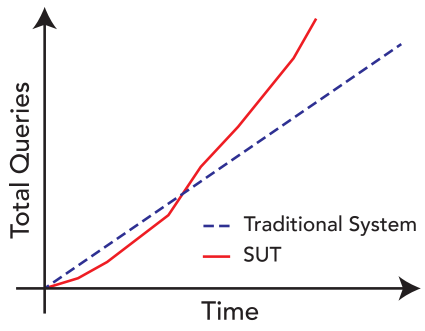
Increasing the number of bands and color-coding them appropriately (e.g., green-yellow-orange-red) could provide additional visual insight into the SUT's behavior. A single-value metric for the adjustment speed can also be obtained as the sum of query times above the SLA threshold over the first N queries after a distribution change.

3) Cost: One of the main advantages of learned systems is that they reduce the total cost of ownership (TCO) by saving on database administration. Therefore, unlike traditional benchmarks where the TCO is often ignored, the new benchmark must include metrics to provide guidance regarding the potential savings from using a given system.

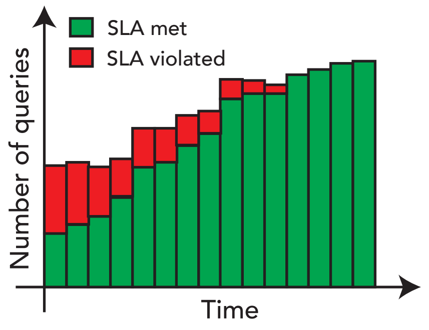
We propose to break down the cost-per-performance metrics into training and execution time. Therefore, while the cost per execution time remains similar to the classical setting, the cost per training time provides new insights into the SUT's behavior. In learned systems with separate training and execution phases, we should evaluate the cost of training on different hardware (CPU, GPU, or TPU). Model training could also run longer or use more data for a higher training cost. The performance achieved for a given training cost reveals an interesting trade-off compared to a traditional system that is manually optimized. Figure 1d illustrates such a trade-off by comparing the throughput achieved by the SUT for different training costs with the throughput achieved by a traditional baseline system that is manually tuned by a database administrator. In this case, the traditional system cost is a step function representing different optimization efforts. In practice, drawing such a plot would require collecting statistics on database administrators and manual optimization costs. This plot allows us to define a new metric: the training cost to outperform a



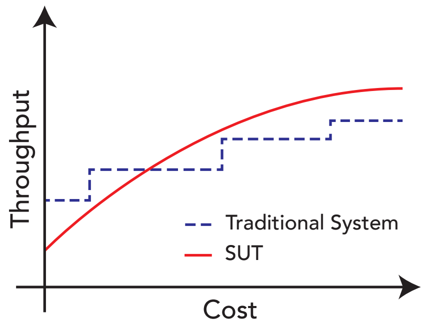
(a) Throughput per workload or data distribution



(b) Cumulative queries over time



(c) Service-Level Agreement (SLA) violations



(d) Throughput per cost

Fig. 1: Illustration of various metrics for a new benchmark for learned systems.

traditional system. Comparing the training cost and execution cost to the total cost of a traditional system allows users to draw conclusions regarding the potential savings of using a learned system. For example, it could be more profitable to use a learned system with a GPU rather than rely on a manually optimized system and its associated database administration costs. It may be difficult to measure the training cost in learned systems with online learning (e.g., reinforcement learning). In this case, we propose to measure the system metrics (e.g., CPU, memory, GPU) corresponding to the training overhead to estimate the training cost.

VI. Conclusions and Future Work

We believe the community should start exploring ways to benchmark data management systems with learned components. In this paper, we first described how learned systems fundamentally change the data management game and argued that existing benchmarks are insufficient to properly understand and compare the performance of these new systems. We then presented additional requirements for a new generation of benchmarking tools. Finally, we outlined how such benchmarks should work and provided new metrics to complement existing ones. We leave for future work the design and proper specification of a new benchmark. We plan to implement an initial version of this benchmark and evaluate different learned systems. We welcome community feedback on our initial ideas.

Acknowledgment

We would like to thank our anonymous reviewers for their feedback that improved this work. This research is supported by the Swiss National Science Foundation Early Postdoc Mobility Fellowship Grant No. P2ELP2_195136, Google, Intel, and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL) at MIT, and NSF IIS 1900933. This research was also sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] R. O. Nambiar and M. Poess, “The making of tpc-ds.” in *VLDB*, vol. 6, 2006, pp. 1049–1058.
[2] M. Poess and C. Floyd, “New tpc benchmarks for decision support and web commerce,” *ACM Sigmod Record*, vol. 29, no. 4, pp. 64–71, 2000.
[3] P. Boncz, T. Neumann, and O. Erling, “Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark,” in *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 2013, pp. 61–76.
[4] S. Chen, A. Ailamaki, M. Athanassoulis, P. B. Gibbons, R. Johnson, I. Pandis, and R. Stoica, “Tpc-e vs. tpc-c: characterizing the new tpc-e benchmark via an i/o comparison study,” *ACM Sigmod Record*, vol. 39, no. 3, pp. 5–10, 2011.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb,” in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[6] A. Wolski, “Tatp benchmark description (version 1.0),” 2009.

[7] https://github.com/memsql/dbbench, 2020.

[8] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, 2018, pp. 489–504. [Online]. Available: https://doi.org/10.1145/3183713.3196909

[9] T. Kraska, M. Alizadeh, A. Beutel, H. Chi, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan, “Sagedb: A learned database system,” in *CIDR*, 2019.

[10] S. Idreos, K. Zoumpatianos, S. Chatterjee, W. Qin, A. Wasay, B. Hentschel, M. Kester, N. Dayan, D. Guo, M. Kang *et al.*, “Learning data structure alchemy,” *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 42, no. 2, 2019.

[11] B. Zhang, D. V. Aken, J. Wang, T. Dai, S. Jiang, J. Lao, S. Sheng, A. Pavlo, and G. J. Gordon, “A demonstration of the ottertune automatic database management system tuning service,” *PVLDB*, vol. 11, no. 12, pp. 1910–1913, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p1910-zhang.pdf

[12] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, “Automatic database management system tuning through large-scale machine learning,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD ’17, 2017, pp. 1009–1024. [Online]. Available: https://db.cs.cmu.edu/papers/2017/p1009-van-aken.pdf

[13] K. Kanellis, R. Alagappan, and S. Venkataraman, “Too many knobs to tune? towards faster database tuning by pre-selecting important knobs,” in *12th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*, 2020.

[14] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, “Bao: Learning to steer query optimizers,” 2020.

[15] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, “Neo: A learned query optimizer,” Tech. Rep., 2019. [Online]. Available: http://arxiv.org/abs/1904.03711

[16] S. Krishnan, Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica, “Learning to optimize join queries with deep reinforcement learning,” *CoRR*, vol. abs/1808.03196, 2018. [Online]. Available: http://arxiv.org/abs/1808.03196

[17] Z. Yang, B. Chandramouli, C. Wang, J. Gehrke, Y. Li, U. F. Minhas, P.-Å. Larson, D. Kossmann, and R. Acharya, “Qd-tree: Learning data layouts for big data analytics,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 193–208.

[18] V. Jain, J. Lennon, and H. Gupta, “Lsm-trees and b-trees: The best of both worlds,” in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1829–1831.

[19] Z. Cao, S. Dong, S. Vemuri, and D. H. Du, “Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook,” in *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*, 2020, pp. 209–223.

[20] A. Vogelsgesang, M. Haubenschild, J. Finis, A. Kemper, V. Leis, T. Mühlbauer, T. Neumann, and M. Then, “Get real: How benchmarks fail to represent the real world,” in *Proceedings of the Workshop on Testing Database Systems*, 2018, pp. 1–6.

[21] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska, “Sqlshare: Results from a multi-year sql-as-a-service experiment,” in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 281–293.

[22] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux, “Oltp-bench: An extensible testbed for benchmarking relational databases,” *Proceedings of the VLDB Endowment*, vol. 7, no. 4, pp. 277–288, 2013.

[23] A. G. Fior, J. A. Meira, E. C. de Almeida, R. G. Coelho, M. D. Del Fabro, and Y. Le Traon, “Under pressure benchmark for ddbms availability,” *Journal of Information and Data Management*, vol. 4, no. 3, pp. 266–266, 2013.

[24] R. Marcus and O. Papaemmanouil, “Towards a hands-free query optimizer through deep learning,” *arXiv preprint arXiv:1809.10212*, 2018.

[25] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper, “Learned cardinalities: Estimating correlated joins with deep learning,” in *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. [Online]. Available: http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[26] A. Kipf, M. Freitag, D. Vorona, P. Boncz, T. Neumann, and A. Kemper, “Estimating filtered group-by queries is hard: Deep learning to the rescue,” *1st International Workshop on Applied AI for Database Systems and Applications*, 2019.

[27] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, P. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica, “Deep unsupervised cardinality estimation,” *PVLDB*, vol. 13, no. 3, pp. 279–292, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol13/p279-yang.pdf

[28] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, “Deepdb: Learn from data, not from queries!” *Proc. VLDB Endow.*, vol. 13, no. 7, pp. 992–1005, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p992-hilprecht.pdf

[29] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri, “Selectivity estimation for range predicates using lightweight models,” *PVLDB*, vol. 12, no. 9, pp. 1044–1057, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol12/p1044-dutt.pdf

[30] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[31] A. Kristo, K. Vaidya, U. Çetintemel, S. Misra, and T. Kraska, “The case for a learned sorting algorithm,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1001–1016. [Online]. Available: https://doi.org/10.1145/3318464.3389752

[32] J. Ding, V. Nathan, M. Alizadeh, and T. Kraska, “Tsunami: A learned multi-dimensional index for correlated data and skewed workloads,” *Proc. VLDB Endow.*, vol. -, pp. –, 2020.

[33] J. Ding, U. F. Minhas, J. Yu, C. Wang, J. Do, Y. Li, H. Zhang, B. Chandramouli, J. Gehrke, D. Kossmann, D. Lomet, and T. Kraska, “ALEX: An updatable adaptive learned index,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 969–984. [Online]. Available: https://doi.org/10.1145/3318464.3389711

[34] A. Kipf, R. Marcus, A. van Renen, M. Stoian, A. Kemper, T. Kraska, and T. Neumann, “SOSD: A benchmark for learned indexes,” Tech. Rep., 2019. [Online]. Available: http://arxiv.org/abs/1911.13014

[35] R. Marcus, A. Kipf, A. van Renen, M. Stoian, S. Misra, A. Kemper, T. Neumann, and T. Kraska, “Benchmarking learned indexes,” *Proceedings of the VLDB Endowment*, vol. 14, no. 1, pp. 1–13, 2020.

[36] P. Negi, R. Marcus, H. Mao, N. Tatbul, T. Kraska, and M. Alizadeh, “Cost-guided cardinality estimation: Focus where it matters,” in *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2020, pp. 154–157.

[37] C. Dwork, “Differential privacy: A survey of results,” in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[38] A. Blum, K. Ligett, and A. Roth, “A learning theory approach to noninteractive database privacy,” *Journal of the ACM (JACM)*, vol. 60, no. 2, pp. 1–25, 2013.

[39] V. Gupta, G. Miklau, and N. Polyzotis, “Private database synthesis for outsourced system evaluation.” in *AMW*, 2011.

[40] N. Patki, R. Wedge, and K. Veeramachaneni, “The synthetic data vault,” in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 399–410.

[41] S. Deep, A. Gruenheid, K. Nagaraj, H. Naito, J. Naughton, and S. Viglas, “Diametrics: benchmarking query engines at scale,” *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3285–3298, 2020.

[42] P. Jaccard, “Étude comparative de la distribution florale dans une portion des alpes et des jura,” *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[43] F. J. Massey Jr, “The kolmogorov-smirnov test for goodness of fit,” *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[44] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola, “A kernel method for the two-sample-problem,” *Advances in neural information processing systems*, vol. 19, pp. 513–520, 2006.