Towards instance-optimized data systems

Tim Kraska
Massachusetts Institute of Technology
Cambridge, MA
kraska@mit.edu

ABSTRACT

In recent years, we have seen increased interest in applying machine learning to system problems. For example, there has been work on applying machine learning to improve query optimization, indexing, storage layouts, scheduling, log-structured merge trees, sorting, compression, and sketches, among many other data management tasks. Arguably, the ideas behind these techniques are similar: machine learning is used to model the data and/or workload in order to derive a more efficient algorithm or data structure. Ultimately, these techniques will allow us to build "instance-optimized" systems: that is, systems that self-adjust to a given workload and data distribution to provide unprecedented performance without the need for tuning by an administrator. While many of these techniques promise orders-of-magnitude better performance in lab settings, there is still general skepticism about how practical the current techniques really are.

The following is intended as a progress report on ML for Systems and its readiness for real-world deployments, with a focus on our projects done as part of the Data Systems and AI Lab (DSAIL) at MIT. By no means is it a comprehensive overview of all existing work, which has been steadily growing over the past several years not only in the database community but also in the systems, networking, theory, PL, and many other adjacent communities.

PVLDB Reference Format:
Tim Kraska. Towards instance-optimized data systems. PVLDB, 14(12): 3222 - 3232, 2021.
doi:10.14778/3476311.3476392

1 INTRODUCTION

Database systems have a long history of carefully selecting efficient algorithms, e.g., a merge vs a hash-join, based on data statistics. Yet, most databases are general-purpose systems that are not purpose-built for a specific workload or data distribution. For this reason, the architects and developers of a database system have to make compromises when building the system. They have to decide the type of index structures to implement, what scheduling algorithms to use, how to organize data on disk as well as in-memory, and so on. All of these decisions come with their own trade-offs, which they try to balance. As a result, the final system will likely perform well but not achieve the best possible performance for any given application.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476392

In contrast, if they would design a system from scratch in C++ for just one particular application for one single customer and maybe even just one single situation, such as the transaction processing for Walmart's online shop during Black Friday, they would probably design the system very differently than a general purpose transactional database system. Unfortunately, building a system like a database is very time-consuming and expensive and involves hundreds of engineers, making it impossible to design a system for every application and user from scratch.

But what if the system could automatically self-adjust to the data and workload, to provide near-optimal performance? This is the core idea behind a range of recently proposed techniques to automatically adjust the system for a given application without human intervention. Here we refer to this idea as an instance-optimized system in analogy to instance-optimal algorithms [32]. Machine learning plays an important role in this area, as it provides well-known techniques to "learn" something about the (expected) data, workload, or hardware behavior, which can then be used to optimize the system. However, instance-optimizations might not require machine learning; often more traditional optimization techniques suffice. This article outlines three different degrees of instance-optimization and their readiness for industry.

2 KNOB AND DESIGN TUNING

The simplest form of instance-optimization automatically tunes system parameters. Many systems provide a range of parameters, i.e., knobs, to tune the system, mostly in order to achieve better performance for a particular application or hardware. For example, the buffer pool size is often an important parameter in database systems, which should be tuned depending on the available memory and workload. Similarly, database systems offer a range of design options, such as indexes or materialized views, which can significantly reduce the data-processing time.

Traditionally, an administrator would tune the parameters of a system and create indexes or materialized views. Unfortunately, this in itself can be a time-consuming task and not every organization can afford to pay a skilled database administrator. Moreover, workloads tend to change, making the tuning process a never-ending task.

Not surprisingly, there is a long history of research to automate this process [2, 3, 13–15, 20, 30, 40, 67, 98, 109] and several techniques have been successfully deployed into production by Microsoft [15] and Oracle [21], among many others. Traditionally, these techniques require a (representative) training workload provided by an administrator and address only one problem (e.g., index selection) at a time. More recent techniques in this area often use modern machine learning techniques, such as reinforcement learning, to be more proactive, make changes more quickly (online), and/or address the problem more holistically [4, 5, 22, 41, 60, 61, 61, 62, 94, 94, 101].

Yet, the goal of knob and design tuning is to assist, and sometimes replace, the database administrator while not making fundamental changes to the database system itself. Hence, the tuning potential is restricted to the knobs and design choices the systems have today. This leads to the next category of instance-optimized components, which go beyond the more traditional algorithms and often deeply embed a model in the algorithm itself. Thus, for the purpose of this paper, we consider a reinforcement-learning-based scheduling algorithm to be an instance-optimized component, whereas a reinforcement-leaning-based model to select the best indexes is considered a knob and design tuning technique.

Still, it should be pointed out that Pavlo et al. argue in [94] that in order to create "self-driving" database systems, one also needs to reconsider many of the traditional design choices. In particular, the authors argue that we require a new system design that allows configuration to be changed much more quickly without requiring a restart of the database and potentially also to provide more "knobs" for automization. This ultimately blurs the line between knob/design tuning and what we refer to as instance-optimized systems, as discussed further below.

3 INSTANCE-OPTIMIZED COMPONENTS

More recently a wide range of learning-enhanced algorithms and data structures to build instance-optimized components have been proposed. These techniques go far beyond knob tuning by deeply embedding models or optimization into the algorithms, data structures, or entire component designs. In "The Three Pillars of Machine Programming" [39] these techniques fall mainly under the invention and adaptation pillar. In the following, we highlight research results on only two instance-optimized database components, indexing and query optimization, with a focus on techniques out of DSAIL [26] at MIT. However, many other components have been "instance-optimized," including scheduling [72, 73, 103], sorting [58, 59], joins [38], and compression [12, 51] (see [28] for a more comprehensive list of papers).

3.1 Building Instance-Optimized Components

Generally, we identify three different ways to develop instance-optimized data structures, algorithms, and components.

Design continuum: The first approach is to use a model to synthesize or configure a traditional algorithm. For example, in [48] Stratos et al. proposed a design continuum that unifies major distinct data structures/algorithms under the same model. The core idea is that every data structure consists of a few fundamental design concepts and that different optimization techniques can be used to compose several of these concepts into an instance-optimized data structure for a given data set and workload [50]. However, so far these design concepts consist entirely of traditional techniques. Moreover, making all the design concepts compatible with each other—including ways to transition between them—presents an immense research and engineering effort.

ML-enhanced algorithms: Another approach to building instance-optimized algorithms, data structures, and components is by means of an oracle. That is, the algorithm assumes an oracle, typically a machine learning model, that is capable of making a prediction that is imperfect but relevant to the problem; this prediction is then used to derive a more efficient algorithm. Key to this approach is the model's ability to capture something about the problem more efficiently than a data agnostic approach would. For example, a model that predicts the position of a key inside a sorted array can be used to improve the efficiency of a range index [56] and sorting algorithm [58]. In contrast, a model that predicts whether a key is in a given set can be used to improve Bloom filters [56], and a model that predicts the frequency of an item in a set can improve the estimation quality of counting sketches [29, 47], whereas a model that predicts the job execution time can be used to derive a more efficient scheduling algorithm [81, 82]. Here we can distinguish between two cases: oracles that predict something about the future (e.g., the job arrival or execution time) and oracles that "predict" (or estimate) something about known data (e.g., a empirical CDF model to predict the rank of an item). Interestingly, while it is generally important that prediction-based models generalize, overfitting can be a good thing if the model only needs to capture something about the known data more compactly.

The general category of ML-enhanced algorithms is also sometimes referred to as algorithms with predictions [82], oracle-based algorithms, learning-enhanced algorithms, or learned algorithms. Moreover, the idea of embedding models to improve existing algorithms opens up a whole new range of interesting opportunities and research challenges for a wide range of existing problems [82].

Full-model replacement: For some restricted problems it is possible to entirely replace an algorithm or data structure through a model. This is most often the case if the original algorithm is already a heuristic to solve an NP-hard problem and does not require strict guarantees (e.g., a guarantee to find the best solution). Examples include cardinality estimation and scheduling algorithms, among other problems.

Interestingly, these three approaches are not exclusive to each other. For example, one could consider oracle-based approaches as part of a design continuum. Similarly, oracle-based algorithms often already use code synthesis to build more efficient models. For example, in [56] the authors propose combining traditional B-trees with regression models to achieve better CDF estimates. It still remains to be shown whether it is better to include a design continuum as part of a single dominant oracle-based algorithm or if oracle-based ideas are better as a part of a design continuum.

3.2 Learned Indexes

In our SIGMOD 2018 paper "The Case for Learned Index Structures" [55, 56], we showed that traditional algorithms and data structures, particularly B-trees, hash-maps, and Bloom filters, can be enhanced by learned models with significant space and performance benefits while providing the exact same semantic guarantees. To make this possible, we invented ways to combine probabilistic models with traditional data structures and algorithms and a new type of model structure, called the recursive model index (RMI), which can have nano-second inference time.

While the results were promising, the paper also excited a lot of controversy because of its highly unintuitive result that machine learning can improve data structures with (sub-)linear complexity. Most notably, three individual blog posts by Prof. Neumann, Prof. Mitzenmacher, and a group at Stanford raised the question whether other more traditional approaches could outperform learned indexes. Moreover, we made several simplifying assumptions (e.g., we focused on read-only in-memory workloads) and the paper did not show how a learned index would impact an end-to-end system. This in turn raised the question of how significant the impact would be in real-world systems. The lack of an open-source evaluation further contributed to the uncertainty.

3.2.1 A progress report. Fortunately, many researchers around the globe started to expand on our initial results to address many of the open research questions and limitations (see [28] for a collection of these papers). For example, in a joint evaluation with TU Munich [27, 75], we showed that learned indexes are indeed much smaller while providing better or similar performance than traditional state-of-the-art indexes [27, 75]. Similar results were also found by other (independent) research groups [69][1] and several micro experiments in various follow-up papers (e.g., [24, 36, 107]). We open-sourced all our implementations and created a public index performance leaderboard [27], allowing for more fair comparisons going forward. Very exciting is also the theoretical result by Ferragina et al. [33, 34], which proves why learned indexes are more space efficient under certain data/workload assumptions.

Note that [27] also contains a comparison against newer index variants such as the radix-based binary search and the Compact HisTree (CHT) [16], a highly read-optimized and compressed data structure. While the author of [16] argues that the Compact HisTree is not a learned index structure, we do consider it one, as it explicitly models the histogram of the data rather than relying solely on binary decisions to navigate a tree structure. Regardless of its classification, [27] shows that RBS and CHT are only able to slightly outperform RMIs on very few datasets and that only at a very large index size, often surpassing the size of the original dataset, rendering its impact on real-world systems more questionable (see below). This is due to the fact that both approaches require a radix table to be efficient; in the extreme the radix table would store a pointer for each possible key in the domain, not just all keys in the data set.

Learned indexes have also been integrated into full disk-based systems. Interestingly, the evaluations of this work have shown that the advantage of learned indexes on disk-based systems often comes through their smaller index size, which can be orders-of-magnitude smaller, rather than the faster lookup time. For example, Google integrated learned index structures into their BigTable system, where it was able to improve the throughput by up to 2x [1]. This performance benefit can be attributed to the fact that the smaller index makes it possible to save an IO request and the associated de-compression cost. Similarly, research work at the University of Wisconsin showed how learned indexes can improve "lookup performance by 1.23x-1.78x as compared to state-of-the-art production LSMs" [18], in this case due to both the index size and faster lookup performance.

[1]Note that the evaluation in [69] uses the same traditional baselines as in [75], but re-implements the RMI-model-based index instead of using the one from [75], that seems less tuned than the other variants, with a training technique which is more sensitive to outliers.

Many of the other previous limitations, such as the lack of update support, have also been addressed and, to date, we have counted over 50 new variants of learned indexes (see [28] for a list of papers). For example, Alex [24], Bourbon [18], and PGM [36] extend the idea of learned index to support inserts. Interestingly, it has also been found that while overfitting is often a good property for read-only workloads, supporting writes might require models that better generalize and consider potential future updates [42]. Similarly, different learning approaches for the models and cost functions have been proposed. Whereas our original RMI structure was trained greedily top-down, FittingTree [37], RadixSpline (RS), and the PGM index [36] are all trained bottom-up. FittingTree [37] uses linear splines [89] to approximate the CDF of the data, whereas RS combines the idea of splines with an additional radix-based lookup table to achieve faster lookup performance. In [31] the authors show that the right cost function for training should be the log-error, because of the subsequent exponential or binary search process. There exists also several extension for learned indexes on string data [104, 110]. Finally, more recent projects explored how to adjust learned indexes for modern hardware, such as multi-core environments [107], NVM [66], and RDMA [111].

3.2.2 Learned Bloom filters and hash-maps. Similarly there has been exciting follow-up work on the idea of Learned Bloom Filters. In [80] Prof. Mitzenmacher proposed the Sandwiched Learned Bloom Filter (SBF) as an improved learned Bloom filter variant on our original work [56]. It combines two Bloom filters (not just one) with a model that predicts whether a key is in a given set. Later, Dai and Shrivastava proposed Adaptive Learned Bloom Filter (Ada-BF) [19], which leverages the certainty of the prediction to achieve smaller Bloom filter sizes. In contrast, our most recent work, Partitioned Learned Bloom Filters (PLBF) [108], can be regarded as a generalization with provable guarantees of Sandwiched Learned Bloom Filter (SBF) and Adaptive Learned Bloom Filter (Ada-BF) [19]. However, there has been other work on improving Learned Bloom Filters, including improvements using meta-learning techniques [97] or learned Bloom filters for data streams [64]. Moreover, we also theoretically and experimentally evaluated (RMI) models as potentially better hash functions for hash-maps [100].

We also addressed the questions raised around learned hash-maps in [100], which theoretically and experimentally evaluates when learned models can and cannot outperform hash functions to build more efficient hash-maps.

3.2.3 Multi-dimensional indexes and storage layouts. One of the most exciting directions for learned index structures are techniques to index multi-dimensional data to improve upon index structures such as R- or KD-trees. With the Flood project [84] we demonstrated how we can extend the idea of learned indexes to multi-dimensional indexes, which outperform alternative general-purpose data structures by orders of magnitude by automatically adjusting to the data distribution and workload through models. With Tsunami [25], we later extended the Flood technique to create the first self-optimizing in-memory storage manager, which also takes advantage of correlation within the data. Interestingly, in order to achieve the orders-of-magnitude improvement in lookup performance benefits, both techniques also require optimization of the storage layout; they both are designed to be clustered indexes rather than being used as secondary indexes. Hence, we consider Flood and Tsunami strongly related to techniques that help to automatically partition the data based on the workload, such as Schism [17], QD-tree [115], or MTO [23]. Yet, traditionally automatic data partition algorithms fall more into the knob tuning category, as they do not fundamentally change the components of the database, whereas Flood and Tsunami go further and introduce more opportunities for optimization (i.e., degrees of freedom) than a traditional system would. In fact, QD-tree [115] and MTO [23] were explicitly designed to be used with a traditional data-warehouse database.

3.3 Learned Query Optimization

Query optimization remains one of the most challenging problems in data management systems. In general, a query optimizer has four components: (1) a cost model, (2) a cardinality estimation model, (3) transformation rules, and (4) a search strategy. As the name implies, the cost model's goal is to estimate the cost (e.g., latency) for a given query plan. Here the cost model might depend on the hardware, the schema, and even the state of the database (e.g., what pages are in the cache), but is generally independent of the data itself.[2] The cardinality estimation model depends on the data but is otherwise independent of the workload. The transformation rules and search strategy are independent of either. The rules define what rewrites are possible and how they can be combined, whereas the search strategy determines the way the "best" plan out of all possible plans is found.

Given the importance and the complexity of the problem, there have been several attempts to build "learned" query optimizers, most often focusing on the cardinality estimation component. For example, one of the earliest applications of learning to improve the cardinality estimate model was Leo [105], which used successive runs of similar queries to adjust histogram estimators. More recent approaches [53, 63, 93, 106, 112] have used deep learning to learn cardinality estimations or query costs in a supervised fashion. Other work [7–9] present a query-driven approach to cardinality estimation, using techniques such as self-organizing maps. Unsupervised approaches, based on Monte Carlo integration, have also been proposed [116, 117]. In [43], the authors present a scheme called CRN for estimating cardinalities via query containment rates.

While all of these works demonstrate improved cardinality estimation accuracy (potentially useful in its own right, as in [7–9]), they do not provide evidence that these improvements lead to better query plans. In fact, it is possible that while the cardinality estimation accuracy (or Q-Error) improves overall, queries become slower [86, 87]. Ortiz et al. [91] showed that certain learned cardinality estimation techniques may improve mean performance on certain data sets, but tail latency is not evaluated.

More recent work has tried to address this shortcoming by training the cardinality estimation model while more fully considering its impact on the cost model. For example, Negi et al. [88] showed how prioritizing training on cardinality estimations that have a large impact on query performance can improve estimation models. Furthermore, [57, 77] showed that, with sufficient training, reinforcement learning-based approaches could find plans with lower costs (according to the PostgreSQL optimizer). In [86, 87], we proposed a new loss function, Flow-Loss, that approximates the optimizer's cost model, which it uses to optimize explicitly for better query plans. At the heart of Flow-Loss is a reduction of query optimization to a flow routing problem on a certain "plan graph" in which different paths correspond to different query plans. The result shows that, across different architectures and databases, a model trained with Flow-Loss improves the plan costs and query runtimes despite having worse estimation accuracy than a model trained with Q-Error.

With Neo [79], we took this even further and used reinforcement learning to directly improve the query latency by addressing cardinality estimation, cost models, and the search strategy all at once (Neo still assumes a given set of transformation rules). Thus, Neo is one if not the first end-to-end learned query optimizer. It uses a value network based on tree convolution, employs row vectors to represent query predicates, and applies learning from demonstration to shorten Neo's training time. As a result, Neo could learn optimization strategies that were competitive with commercial systems after 24 hours of training. However, none of these techniques are capable of handling changes in schema, data, or query workload, and database administrators expressed concerns about robustness and how much they could trust a completely learned query optimizer.

Motivated by these difficulties, we more recently introduced Bao (the Bandit optimizer) [76]. Bao takes advantage of the wisdom built into existing query optimizers by providing per-query optimization hints. The core idea behind Bao is that we do not try to learn an optimizer from scratch: instead, we take an existing optimizer (e.g., PostgreSQL's optimizer) and learn when to activate (or deactivate) some of its features on a query-by-query basis. In other words, Bao seeks to build learned components on top of existing query optimizers in order to enhance query optimization, rather than replacing or discarding traditional query optimizers altogether. This also allows Bao to be used as an advisor for a database administrator. In [76], we showed that Bao can offer both reduced costs and better performance compared with commercial systems deployed in the cloud. We later informally extended the evaluation in [74] to include Vertica, Azure Synapse, and Redshift in the evaluation, showing cost reductions of up to 25%. Finally, in [85], together with Microsoft, we evaluated the benefits of Bao for Big Data workloads, showing up to 90% runtime latency savings for complex queries.

3.4 Other Algorithms

Machine learning techniques have been applied to a wide variety of other problems relevant for database systems. For example, reinforcement-learning-based techniques were proposed for (job) scheduling [72, 73, 103], garbage collection [52], and managing elastic clusters [65, 92] (see also [71, 102]). Similarly, oracle-based algorithms were suggested for sorting [58], caching [68], count-min sketching [47], data compression [12, 51], and optimizing the transaction execution policy [95], among many other problems. While there are already too many learning-enhanced algorithms to list here, the whole area is also evolving quickly. Just to mention two interesting projects out of DSAIL: Vaidya et al. recently proposed the use of a CDF model to improve the space efficiency of range filters

[2]This might not be true in cases where the data has a lot of variable length attributes and/or requires complex UDFs.

(currently under submission), and Kristo et al. recently proposed a new LearnedSort algorithm [59], which is more robust against duplicates and outperforms the best sorting algorithms we were able to find by up to 30% in sorting throughput on a wide range of data sets, even when the model training time is included.

3.5 Future Research Opportunities

The entire field of instance-optimized or learning-enhanced algorithms is evolving extremely rapidly across various communities. For example, the idea of the learned index alone has also been applied to other areas, including network package classification [99], DNA sequence search [46], longest prefix matches [44], and inverted indexes [90, 114], and follow-on has appeared in database (e.g., VLDB[113]), systems (e.g., OSDI [111]), machine learning (e.g., NeurIPS [19]), networking (e.g., SIGCOMM [99]), and theory (e.g., Theor. Comput. Sci. [34]) conferences. For an overview the reader is referred to a recent tutorials [6, 49, 70], surveys [35], and our collection of papers [28]. However, a lot of interesting research questions and opportunities remain and it will be exciting to see how the entire field of ML for Systems will develop.

In the case of learned indexes, there is a range of open opportunities for study. For example, while [1, 18] show that learned indexes can improve the performance of large-scale disk-based systems, it would be exciting to see to what degree learned indexes would impact an in-memory OLTP and HTAP system or embedded database, like DuckDB [96]. Integrating a learned index into those systems would also require investigation into more advanced recovery techniques (e.g., is it possible to simplify ARIES [83]) and techniques to efficiently support range-locks using learned indexes).

As [75] shows, learned indexes are not beneficial for all data sets. In particular, if the data set contains a lot of duplicates, traditional indexes might be better. Similarly, some models and training methods are not robust against outliers. Hence, an interesting research direction is to make learned indexes more robust by considering outliers and duplicates more closely. Here, CDFShop [78] already takes a step in the right direction, but many more improvements are possible. Especially interesting is how to better combine models with the traditional techniques used in Tries and B-trees and explore the implications. Similarly, it would be interesting to explore new end-to-end learning techniques. Current approaches either build an index bottom-up or top-down, often considering the placement of data as a secondary optimization goal. However, if we could consider it as a holistic optimization problem, that co-optimizes the model structure, model types, and data placement, it can be assumed that we could achieve faster lookup times while further reducing the index size.

Yet, to us the most exciting directions are applications of learned indexes in related fields and for multi-dimensional data. For example, our latest results on LISA [46] build on and extend FM-index, which is the state-of-the-art technique widely deployed in genomics tools. Experiments with human, animal, and plant genome data sets indicate that LISA achieves up to $2.2x$ and $10.8x$ speedups over the state-of-the-art FM-index-based implementations for exact search and super-maximal exact match (SMEM) search, respectively. Similarly, we believe that we are just at the beginning of rethinking the way we build multi-dimensional indexes and co-optimize index structures with storage layouts for an entire workload with complex access paths.

In the same respect, there are myriads of open research challenges for learned query optimizers. While Bao [76] provides a practical approach to improving existing query optimizers, it adds significant query optimization overhead. For long-running analytical queries the overhead often plays no important role, but it certainly would for short-running queries, as found in transactional workloads. On the other hand, end-to-end learned query optimizers, such as Neo, still suffer from the long training time and even small changes in the schema (or index configuration) might require the entire model to be retrained. A promising recent result by Hilprecht and Binnig [45] proposes the use of zero-shot learning to improve upon Neo [79]. This represents a first step toward making it possible to transfer a learned model from one database configuration to another. However, the approach still has severe limitations, as the cost model might still remember cardinality corrections and/or might not be able to correct wrong cardinalities, depending on the amount of training data and setup. Moreover, database vendors usually require that query optimizers are robust rather than having the best possible performance. For example, a query that ran fast and suddenly runs significantly slower might result in customer complaints, regardless of performance. One interesting idea in that regard is to develop techniques that keep the query latency anchored at some reference point, while passing on resource savings to the cloud provider. Many other related research challenges remain, including how to reduce query optimization times or how to better balance exploration and exploitation in a production setting, model management, explainability, etc.

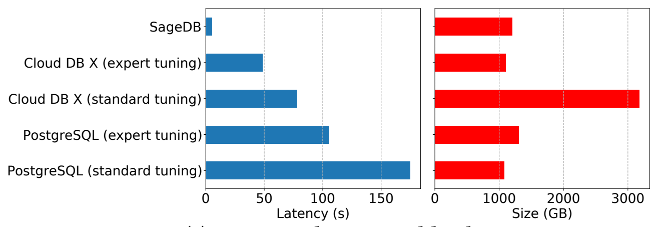
Obviously, this only outlines a few selected remaining research challenges and even does not touch upon other components, such as scheduling, distributed query processing, among other areas.

4 INSTANCE-OPTIMIZED SYSTEMS

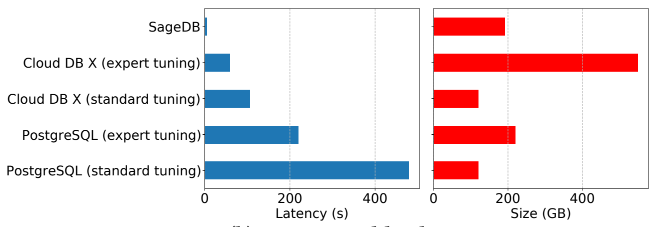
The previous section described how different components of a database management system can be instance-optimized to ultimately provide better performance. However, it remains an open question how different learned components can actually be combined to build an entire instance-optimized system. This is the question we are exploring with SageDB [54]. While it is still too early to draw any conclusions about the expected performance benefits on actual applications and analytic workloads, initial results using real-world workload traces on our current prototype are promising. Moreover, they have already revealed some interesting trade-offs and insights into how different learned components might affect each other. The following is a brief update on the development of SageDB, as well as a selected set of early lessons learned and future research opportunities.

4.1 SageDB

When we started to develop SageDB, the first question we had to address was if we really wanted to build a database system from scratch. While building an entire system from the ground up would give us the greatest flexibility, it would also increase the time before the system would become stable and feature complete enough that



(a) Data Warehouse Workload



(b) Bitcoin workload

Figure 1: Preliminary SageDB results on two workloads. Results include block-layout optimization and automatic replication, encoding, and partial materialization. However, the workload does NOT include any joins, nor does the system support (yet) learned query optimization, multi-dimensional indexes, or memory hierarchy optimizations. Hence, there exists a lot of opportunity for further optimizations.

other users would be willing to try it out. At the same time, we considered the prospect of early deployments essential to this line of work.

We thus decided to build SageDB as an accelerator for PostgreSQL. The core idea is that we leverage the concept of foreign data wrappers in PostgreSQL, which allows SageDB to take control of managing the data of one or more tables. Further, external data wrappers allow us to push down queries, even those containing joins, to SageDB without changing the code of PostgreSQL. That way we were able to build a system that not only is fully compatible to PostgreSQL but also provides an easier migration path for existing PostgreSQL deployments. Users are able to explicitly define which tables should be managed by SageDB vs natively by PostgreSQL. At the same time, the loose connection between PostgreSQL and SageDB allows us to build SageDB over time as part of PostgreSQL until it is ready to stand on its own. The main downside is that we will have the PostgreSQL and foreign data wrapper overhead, forcing us to focus on long-running analytical workloads for the moment.

As of July 2021, we have a first prototype of PostgreSQL with the SageDB accelerator running, which is able to instance-optimize the data layout of a single table in a multi-dimensional fashion, including partial materialization, replication, and encoding selection. Moreoever, SageDB uses the separation of storage and compute that we proposed in 2008 [11] for cloud database systems and is now commonly found in commercial offerings like Snowflake. That is, the data is stored in large blocks on cloud storage services like S3 and brought into a SageDB instance on demand. This allows SageDB to actually work with data, which goes beyond the capacity of a single compute node, and it is possible to have several SageDB instances running on the same data for scalability.[3] However, in contrast to current traditional cloud database systems, SageDB automatically adjusts how the data is stored, indexed, and replicated in blocks based on the workload and data.

Moreover, the system uses automatically partial materialization, a concept similar to materialized views but on a finer granularity. While materialized views "cache" the entire query result, SageDB's partial materialization technique is able to build pre-aggregates, such as running sums, aggregations, and counts, on a block or at an even more fine-grained level. Then at run-time SageDB determines for each query what partial materialized results can be used to avoid scanning certain parts of the data to speed up the query processing time. Obviously, the benefits of partial materialization care highly dependent on the workload, available amount of memory, and the amount of work required to keep them up to date. This is exactly where instance-optimization comes into play.

Similarly, SageDB provides support to (partial) replicate and reorganize each replica for performance and select the most appropriate compression technique for the data based on the data and workload. However, the current prototype does not yet integrate a learned query optimizer; neither does it currently support joins, nor does it perform automatic optimization for the entire storage hierarchy (Cloud storage, SSD, NVM, vs Memory): all these are features on our roadmap.

Figure 1 shows some preliminary results of SageDB against PostgreSQL and a commercial cloud datawarehouse (Cloud DB X) for two workloads. The workloads are derived from real-world data and traces, but focus entirely on queries over a single table. The Bitcoin dataset had a size of 500GB; the datewarehouse dataset was 1TB in size. The server in this experiment had 32 vCPUs and 244 GB of RAM. All SageDB data was stored on S3, whereas the Cloud DB used their own storage, and PostgreSQL local storage. We also tuned both systems: first, following standard best-practice tuning technique (standard tuning), and second, more aggressively, using the workload including creating materialized views and aggressive indexing. In the case of the commercial cloud database we also had a professional tuning expert from the cloud vendor help us tune the system. While our latest prototype already integrates multi-dimensional indexes and storage, the results here do not include those optimizations yet.

As Figure 1 shows, SageDB provides speed-ups (latency) of up to 8x compared to the expert tuned cloud-database and up to two orders of magnitude improvement compared to PostgreSQL standard tuned. SageDB achieves these speed-ups by using significantly less storage overhead compared to the expert-tuned configurations. Moreover, in the case of SageDB, no administrator was involved and the system self-optimized the layout, encoding, partial materialization, and replication based on the workload, saving significant human effort. Finally, we believe these numbers are only a glance into what might be possible and we actually expect bigger speed-ups as soon as SageDB fully supports multi-dimensional indexes and storage layouts as well as joins and partial materialization over joins.

[3]Note that our current prototype does not yet allow a query to be processed in parallel by several machines. We do plan to add distributed query processing in the future.

4.2 Lessons Learned

While we are still in the process of fully building SageDB, we have already learned several valuable lessons from our journey. In the following, we highlight some of them.

Lesson 1: Taking a holistic approach is important. Traditionally, systems components are developed in isolation. This compartmentalization helps to reduce the complexity and allows smaller teams to work in parallel. However, it often also yields suboptimal performance, as the components tend to make local decisions rather than global ones. For example, it is tempting to implement a component to store and compress the data and another to index the data. However, from our work on multi-dimensional indexes [25, 84] we already know that it is best to co-optimize both together.

Lesson 2: There are exciting new opportunities for cross-optimization. Interestingly, we found several new optimizations we hadn't considered before we started the SageDB project. For example, traditionally a database would use some partitioning strategy to split the data into blocks or pages and then build a caching layer on top of it. However, if the data and workload can be forecasted, it is also possible to determine which subsets of the data should be always in the cache. This in turn allows us to optimize the storage layout in a way that is more beneficial for caching. In fact, it even allows us to implement the cache entirely differently. Especially if combined with other optimizations such as partial materialization and multi-dimensional in-memory indexes, this can lead to entirely new data layouts and significant speed-ups.

Lesson 3: More knobs are often better. In general, database architects tend to minimize the number of tuning parameters. Every tuning parameter introduces complexity and database administrators already have a hard time tuning them correctly. Hence, the trend towards "knobless" systems. In many cases "knobless" systems are actually just systems where things that obviously should be knobs have a fixed value that cannot be modified by the user. It is easy to equate a system that is "knobless" to one that is autonomous, but we should not take the "easy route" of simply gluing and hiding knobs—this might make the administration easier, but it won't improve system performance.

However, if this complexity doesn't pose an additional burden on the administrator, increasing the number of knobs can actually be a good thing. Additional knobs provide additional degrees of freedom and more ways to better tune a system for a given workload.

Lesson 4: Start small and revisit often. The complexity of building an instance-optimized systems is significantly higher than building a traditional database system. On one hand, we want to have more knobs; on the other hand, we also don't want to have abstracted components, which cannot be co-optimized. In addition, optimizing the system in a timely fashion using techniques like RL also gets significantly harder as the space of potential optimization increases. Thus, our approach is to not try to address all possible optimizations at once, but rather make simplifying assumptions that we will revisit later. For example, we decided to treat the block-level data layout optimization separately from the in-memory data layout optimization, even though ideally we would co-optimize them.

Lesson 5: Evaluate with real-world workloads. We already knew from our experience with building instance-optimized components that the standard benchmarks, like TPC-H, are actually a bad way to evaluate any of these components (see also [10]). For example, for our Bao and Neo work on query optimization, we observed that we were not able to achieve meaningful improvements for TPC-H for any of the commercial database systems. We speculate that the reason for this is that database vendors make sure that their query plans for these common workloads are "perfect." In other words, the database vendors (manually?) "overfit" their systems to the benchmarks. Moreover, benchmarks tend to use synthetic data and workload generators, which do not contain any of the interesting patterns found in many real-world applications and are either too easy or sometimes too hard to learn, as they contain too much randomness. Those are just a few reasons, as outlined in [10], why standard database benchmarks are not suitable for evaluating instance-optimized systems. At the same time, getting real-world data is hard and, to date, we do not have a good solution to the problem, except our ongoing attempts to get real-world data and workload traces from industry.

Lesson 6: Robustness will be an issue. Over the course of the past six months, we had several conversations with cloud vendors about what it would take to operationalize an instance-optimized database. One, if not the biggest, concern is robustness and predictability. In other words, cloud vendors are particularly afraid that a query will run fast and suddenly slow down. Even more interesting, let's assume a traditional database runs a complex query in $100s$ and an instance-optimized system runs the same query first in $5s$ but because of re-optimization and the prioritization of other queries later at $60s$. Even though the entire workload might run faster after the re-optimization, some cloud vendors would prefer to have the predictable slower $100s$ rather than the variation in latency. One potential solution to the problem might be to artificially slow down queries while passing on the resource savings to the cloud provider. For example, if the reference system has a latency of $100s$, the cloud provider might fix the latency at $80s$ even though the query takes only $5s$. However, arguably a $5s$ query requires fewer resources, which would still be beneficial for the cloud provider, while the user gets a decent but not overwhelming performance increase. Another alternative technique might be to optimize in a way that performance degressions are minimized. For example, for Neo [79] we proposed a weighted loss function, which penalized performance degressions compared to PostgreSQL.

4.3 Future Work

We are just at the beginning of understanding how instance-optimized data management systems should be built and what techniques are required to do so. Open research questions in that space range from the right development tools (e.g., how do we efficiently develop systems that have models at the core), debugging (e.g., how do we debug a system that changes its behavior based on the observed workload and data), design principles (e.g., how do we reduce the complexity of building such a system while still being able to do holistic optimization), benchmarking (e.g., can we build standardized benchmarks for instance-optimized systems [10]), to entirely new techniques (e.g., the above-mentioned storage optimizations for caching). However, if successful, these techniques will provide an entirely new way to build (database) systems and unleash unprecedented performance while significantly reducing the cost in compute and human resources to keep the systems performant through administrators.

ACKNOWLEDGMENTS

Our work in the area of ML for Systems would have not been possible without my excellent students, PostDocs, and numerous collaborators from academia and industry. In particular, Jialin Ding and Ryan Marcus were so far instrumental in moving the SageDB project forward. In addition, many of the results shown here wouldn't have been possible without Mohammad Alizadeh, Alex Beutel, Laurent Bindschaedler, Ed Chi, Jeff Dean, Andreas Kipf, Donald Kossmann, Eric Knorr, Ani Kristo, Markos Markakis, Umar Farooq Minhas, Michael Mitzenmacher, Parimarjan Negi, Amadou Ngom, Ibrahim Sabek, Nesime Tatbul, Kapil Vaidya, and Geoffrey Xiangyu Yu.

Some of the research out of MIT featured here was supported by Google, Intel, and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL) at MIT, and NSF IIS 1900933. Part of the research out of DSAIL 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] Hussam Abu-Libdeh, Deniz Altinbüken, Alex Beutel, Ed H. Chi, Lyric Doshi, Tim Kraska, Xiaozhou Li, Andy Ly, and Christopher Olston. 2020. Learned Indexes for a Google-scale Disk-based Database. In *Proceedings of the Workshop on ML for Systems at NeurIPS*.

[2] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. 2000. Automated Selection of Materialized Views and Indexes in SQL Databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*. 496–505.

[3] Sanjay Agrawal, Vivek R. Narasayya, and Beverly Yang. 2004. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*. ACM, 359–370. https://doi.org/10.1145/1007568.1007609

[4] 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, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM, 1009–1024. https://doi.org/10.1145/3035918.3064029

[5] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Billian, and Andrew Pavlo. 2021. An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. *Proc. VLDB Endow.* 14, 7 (2021), 1241–1253.

[6] Abdullah Al-Mamun, Hao Wu, and Walid G. Aref. 2020. A Tutorial on Learned Multi-dimensional Indexes. In *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020*. ACM, 1–4. https://doi.org/10.1145/3397536.3426358

[7] Christos Anagnostopoulos and Peter Triantafillou. 2015. Learning Set Cardinality in Distance Nearest Neighbours. In *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM) (ICDM '15)*. IEEE Computer Society, USA, 691–696. https://doi.org/10.1109/ICDM.2015.17

[8] C. Anagnostopoulos and P. Triantafillou. 2015. Learning to Accurately COUNT with Query-Driven Predictive Analytics. In *2015 IEEE International Conference on Big Data (Big Data) (Big Data '15)*. 14–23. https://doi.org/10.1109/BigData.2015.7363736

[9] Christos Anagnostopoulos and Peter Triantafillou. 2017. Query-Driven Learning for Predictive Analytics of Data Subspace Cardinality. *ACM Trans. Knowl. Discov. Data* 11, 4 (June 2017), 47:1–47:46. https://doi.org/10.1145/3059177

[10] Laurent Bindschaedler, Andreas Kipf, Tim Kraska, Ryan Marcus, and Umar Farooq Minhas. 2021. Towards a Benchmark for Learned Systems. In *37th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2021, Chania, Greece, April 19-22, 2021*. IEEE, 127–133. https://doi.org/10.1109/ICDEW53142.2021.00029

[11] Matthias Brantner, Daniela Florescu, David A. Graf, Donald Kossmann, and Tim Kraska. 2008. Building a database on S3. In *Proceedings of the SIGMOD*. ACM, 251–264.

[12] Lujing Cen, Andreas Kipf, Ryan Marcus, and Tim Kraska. 2021. LEA: A Learned Encoding Advisor for Column Stores. In *aiDM '21: Fourth Workshop in Exploiting AI Techniques for Data Management, Virtual Event, China, 25 June, 2021*. ACM, 32–35. https://doi.org/10.1145/3464509.3464885

[13] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*. Morgan Kaufmann, 146–155. http://www.vldb.org/conf/1997/P146.PDF

[14] Surajit Chaudhuri and Vivek R. Narasayya. 1998. AutoAdmin 'What-if' Index Analysis Utility. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*. ACM Press, 367–378. https://doi.org/10.1145/276304.276337

[15] Surajit Chaudhuri and Vivek R. Narasayya. 2007. Self-Tuning Database Systems: A Decade of Progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*. ACM, 3–14. http://www.vldb.org/conf/2007/papers/special/p3-chaudhuri.pdf

[16] Andrew Crotty. 2021. Hist-Tree: Those Who Ignore It Are Doomed to Learn. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2021/papers/cidr2021_paper20.pdf

[17] Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. 2010. Schism: A Workload-Driven Approach to Database Replication and Partitioning. *PVLDB* 3, 1 (2010), 48–57. https://doi.org/10.14778/1920841.1920853

[18] Yifan Dai, Yien Xu, Aishwarya Ganesan, Ramnatthan Alagappan, Brian Kroth, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2020. From WiscKey to Bourbon: A Learned Index for Log-Structured Merge Trees. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 155–171. https://www.usenix.org/conference/osdi20/presentation/dai

[19] Zhenwei Dai and Anshumali Shrivastava. 2020. Adaptive Learned Bloom Filter (Ada-BF): Efficient Utilization of the Classifier with Application to Real-Time Information Filtering on the Web. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/86b94dae7c6517ec1ac767fd2c136580-Abstract.html

[20] Biplob K. Debnath, David J. Lilja, and Mohamed F. Mokbel. 2008. SARD: A statistical approach for ranking database tuning parameters. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, Mexico*. IEEE Computer Society, 11–18. https://doi.org/10.1109/ICDEW.2008.4498279

[21] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. 2005. Automatic Performance Diagnosis and Tuning in Oracle. In *Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*. www.cidrdb.org, 84–94. http://cidrdb.org/cidr2005/papers/P07.pdf

[22] Bailu Ding, Sudipto Das, Ryan Marcus, Wentao Wu, Surajit Chaudhuri, and Vivek R. Narasayya. 2019. AI Meets AI: Leveraging Query Executions to Improve Index Recommendations. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 1241–1258. https://doi.org/10.1145/3299869.3324957

[23] Jialin Ding, Umar Farooq Minhas, Badrish Chandramouli, Chi Wang, Yinan Li, Ying Li, Donald Kossmann, Johannes Gehrke, and Tim Kraska. 2021. Instance-Optimized Data Layouts for Cloud Analytics Workloads. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 418–431. https://doi.org/10.1145/3448016.3457270

[24] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 969–984. https://doi.org/10.1145/3318464.3389711

[25] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed Workloads. *Proc. VLDB Endow.* 14, 2 (2020), 74–86. https://doi.org/10.14778/3425879.3425880

[26] DSAIL. 2021. Data System and AI Lab. http://dsail.csail.mit.edu/.

[27] DSAIL. 2021. (Learned Index Leaderboard. https://learnedsystems.github.io/SOSDLeaderboard/leaderboard/. [Online; accessed 7-July-2021].

[28] DSAIL. 2021. ML for Systems Papers. http://dsg.csail.mit.edu/mlforsystems/papers/.

[29] Elbert Du, Franklyn Wang, and Michael Mitzenmacher. 2021. Putting the "Learning" into Learning-Augmented Algorithms for Frequency Estimation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 2860–2869. http://proceedings.mlr.press/v139/du21d.html

[30] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with iTuned. *Proc. VLDB Endow.* 2, 1 (2009), 1246–1257. https://doi.org/10.14778/1687627.1687767

[31] Martin Eppert, Philipp Fent, and Thomas Neumann. 2021. A Tailored Regression for Learned Indexes: Logarithmic Error Regression. In *aiDM '21: Fourth Workshop in Exploiting AI Techniques for Data Management, Virtual Event, China, 25 June, 2021*. ACM, 9–15. https://doi.org/10.1145/3464509.3464891

[32] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.* 66, 4 (2003), 614–656. https://doi.org/10.1016/S0022-0000(03)00026-6

[33] Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. 2020. Why Are Learned Indexes So Effective?. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 3123–3132. http://proceedings.mlr.press/v119/ferragina20a.html

[34] Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. 2021. On the performance of learned data structures. *Theor. Comput. Sci.* 871 (2021), 107–120. https://doi.org/10.1016/j.tcs.2021.04.015

[35] Paolo Ferragina and Giorgio Vinciguerra. 2020. Learned Data Structures. In *Recent Trends in Learning From Data*, . Springer International Publishing, 5–41. https://doi.org/10.1007/978-3-030-43883-8_2

[36] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-Index: A Fully-Dynamic Compressed Learned Index with Provable Worst-Case Bounds. *Proceedings of the VLDB Endowment* 13, 8 (April 2020), 1162–1175. https://doi.org/10.14778/3389133.3389135

[37] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. 2019. FITing-Tree: A Data-aware Index Structure. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 1189–1206. https://doi.org/10.1145/3299869.3319860

[38] Vahid Ghadakchi, Mian Xie, and Arash Termehchy. 2020. Bandit join: preliminary results. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2020, Portland, Oregon, USA, June 19, 2020*. ACM, 1:1–1:4. https://doi.org/10.1145/3401071.3401655

[39] Justin Gottschlich, Armando Solar-Lezama, Nesime Tatbul, Michael Carbin, Martin Rinard, Regina Barzilay, Saman P. Amarasinghe, Joshua B. Tenenbaum, and Tim Mattson. 2018. The three pillars of machine programming. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. ACM, 69–80. https://doi.org/10.1145/3211346.3211355

[40] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. 1997. Index Selection for OLAP. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997, Birmingham, UK*. IEEE Computer Society, 208–219. https://doi.org/10.1109/ICDE.1997.581755

[41] Yaniv Gur, Dongsheng Yang, Frederik Stalschus, and Berthold Reinwald. 2021. Adaptive Multi-Model Reinforcement Learning for Online Database Tuning. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. OpenProceedings.org, 439–444. https://doi.org/10.5441/002/edbt.2021.48

[42] Ali Hadian and Thomas Heinis. 2019. Considerations for handling updates in learned index structures. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 3:1–3:4. https://doi.org/10.1145/3329859.3329874

[43] Rojeh Hayek and Oded Shmueli. 2019. Improved Cardinality Estimation by Learning Queries Containment Rates. *arXiv:1908.07723 [cs]* (Aug. 2019). arXiv:1908.07723 [cs]

[44] Shunsuke Higuchi, Junji Takemasa, Yuki Koizumi, Atsushi Tagami, and Toru Hasegawa. 2021. Feasibility of Longest Prefix Matching Using Learned Index Structures. *SIGMETRICS Perform. Eval. Rev.* 48, 4 (May 2021), 45–48. https://doi.org/10.1145/3466826.3466842

[45] Benjamin Hilprecht and Carsten Binnig. 2021. One Model to Rule them All: Towards Zero-Shot Learning for Databases. *CoRR* abs/2105.00642 (2021). arXiv:2105.00642 https://arxiv.org/abs/2105.00642

[46] Darryl Ho, Jialin Ding, Sanchit Misra, Nesime Tatbul, Vikram Nathan, Vasimuddin Md, and Tim Kraska. 2019. LISA: Towards Learned DNA Sequence Search. *CoRR* abs/1910.04728 (2019). arXiv:1910.04728 http://arxiv.org/abs/1910.04728

[47] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-Based Frequency Estimation Algorithms. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=r1lohoCqY7

[48] Stratos Idreos, Niv Dayan, Wilson Qin, Mali Akmanalp, Sophie Hilgard, Andrew Ross, James Lennon, Varun Jain, Harshita Gupta, David Li, and Zichen Zhu. 2019. Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p143-idreos-cidr19.pdf

[49] Stratos Idreos and Tim Kraska. 2019. From Auto-tuning One Size Fits All to Self-designed and Learned Data-intensive Systems. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2054–2059. https://doi.org/10.1145/3299869.3314034

[50] Stratos Idreos, Kostas Zoumpatianos, Subarna Chatterjee, Wilson Qin, Abdul Wasay, Brian Hentschel, Mike S. Kester, Niv Dayan, Demi Guo, Minseo Kang, and Yiyou Sun. 2019. Learning Data Structure Alchemy. *IEEE Data Eng. Bull.* 42, 2 (2019), 47–58. http://sites.computer.org/debull/A19june/p47.pdf

[51] Amir Ilkhechi, Andrew Crotty, Alex Galakatos, Yicong Mao, Grace Fan, Xiran Shi, and Ugur Çetintemel. 2020. DeepSqueeze: Deep Semantic Compression for Tabular Data. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1733–1746. https://doi.org/10.1145/3318464.3389734

[52] Nicholas Jacek and J. Eliot B. Moss. 2019. Learning When to Garbage Collect with Random Forests. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on Memory Management (ISMM 2019)*. Association for Computing Machinery, Phoenix, AZ, USA, 53–63. https://doi.org/10.1145/3315573.3329983

[53] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *9th Biennial Conference on Innovative Data Systems Research (CIDR '19)*.

[54] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H. Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2019. SageDB: A Learned Database System. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p117-kraska-cidr19.pdf

[55] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2017. The Case for Learned Index Structures. *CoRR* abs/1712.01208 (2017). arXiv:1712.01208 http://arxiv.org/abs/1712.01208

[56] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. 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*. ACM, 489–504. https://doi.org/10.1145/3183713.3196909

[57] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *arXiv:1808.03196 [cs]* (Aug. 2018). arXiv:1808.03196 [cs]

[58] Ani Kristo, Kapil Vaidya, Ugur Çetintemel, Sanchit Misra, and Tim Kraska. 2020. The Case for a Learned Sorting Algorithm. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1001–1016. https://doi.org/10.1145/3318464.3389752

[59] Ani Kristo, Kapil Vaidya, and Tim Kraska. 2021. Defeating duplicates: A redesign of the LearnedSort algorithm. In *International Workshop on Applied AI for Database Systems and Applications (AIDB@VLDB) (AIDB '21)*.

[60] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to Develop an AutoTuner for Memory-based Analytics. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1667–1683. https://doi.org/10.1145/3318464.3380591

[61] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 2105–2108. https://doi.org/10.1145/3340531.3412106

[62] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. *Proc. VLDB Endow.* 12, 12 (2019), 2118–2130. https://doi.org/10.14778/3352063.3352129

[63] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality Estimation Using Neural Networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering (CASCON '15)*. IBM Corp., Riverton, NJ, USA, 53–59.

[64] Qiyu Liu, Libin Zheng, Yanyan Shen, and Lei Chen. 2020. Stable Learned Bloom Filters for Data Streams. *Proc. VLDB Endow.* 13, 11 (2020), 2355–2367. http://www.vldb.org/pvldb/vol13/p2355-liu.pdf

[65] Konstantinos Lolos, Ioannis Konstantinou, Verena Kantere, and Nectarios Koziris. 2017. Elastic Management of Cloud Applications Using Adaptive Reinforcement Learning. In *IEEE International Conference on Big Data (Big Data '17)*. IEEE, 203–212. https://doi.org/10.1109/BigData.2017.8257928

[66] Baotong Lu, Jialin Ding, Eric Lo, Umar Farooq Minhas, and Tianzheng Wang. 2021. APEX: A High-Performance Learned Index on Persistent Memory. *CoRR*

abs/2105.00683 (2021). arXiv:2105.00683 https://arxiv.org/abs/2105.00683

[67] Jiaheng Lu, Yuxing Chen, Herodotos Herodotou, and Shivnath Babu. 2019. Speedup Your Analytics: Automatic Parameter Tuning for Databases and Big Data Systems. *Proc. VLDB Endow.* 12, 12 (2019), 1970–1973. https://doi.org/10.14778/3352063.3352112

[68] Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive Caching with Machine Learned Advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80).* PMLR, 3302–3311. http://proceedings.mlr.press/v80/lykouris18a.html

[69] Marcel Maltry and Jens Dittrich. 2021. A Critical Analysis of Recursive Model Indexes. *CoRR* abs/2106.16166 (2021). arXiv:2106.16166 https://arxiv.org/abs/2106.16166

[70] Abdullah Al Mamun, Hao Wu, and Walid G. Aref. 2020. A Tutorial on Learned Multidimensional Indexes. https://www.cs.purdue.edu/homes/aref/learned-indexes-tutorial.html.

[71] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, Frank Cangialosi, Shaileshh Bojja Venkatakrishnan, Wei-Hung Weng, Song Han, Tim Kraska, and Mohammad Alizadeh. 2019. Park: An Open Platform for Learning-Augmented Computer Systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.* 2490–2502. https://proceedings.neurips.cc/paper/2019/hash/f69e505b08403ad2298b9f262659929a-Abstract.html

[72] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2018. Learning Scheduling Algorithms for Data Processing Clusters. *arXiv:1810.01963 [cs, stat]* (2018). arXiv:1810.01963 [cs, stat]

[73] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2018. Learning Scheduling Algorithms for Data Processing Clusters. *arXiv:1810.01963 [cs, stat]* (Oct. 2018). arXiv:1810.01963 [cs, stat]

[74] Ryan Marcus. 2021. More Bao Results: Learned Distributed Query Optimization on Vertica, Redshift, and Azure Synapse. https://learnedsystems.mit.edu/bao-distributed/.

[75] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. 2020. Benchmarking Learned Indexes. *Proc. VLDB Endow.* 14, 1 (2020), 1–13. https://doi.org/10.14778/3421424.3421425

[76] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021.* ACM, 1275–1288. https://doi.org/10.1145/3448016.3452838

[77] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM @ SIGMOD '18).* Houston, TX.

[78] Ryan Marcus, Emily Zhang, and Tim Kraska. 2020. CDFShop: Exploring and Optimizing Learned Index Structures. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020.* ACM, 2789–2792. https://doi.org/10.1145/3318464.3384706

[79] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019), 1705–1718. https://doi.org/10.14778/3342263.3342644

[80] Michael Mitzenmacher. 2018. A Model for Learned Bloom Filters and Optimizing by Sandwiching. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada.* 462–471. https://proceedings.neurips.cc/paper/2018/hash/0f49c89d1e7298bb9930789c8ed59d48-Abstract.html

[81] Michael Mitzenmacher. 2020. Scheduling with Predictions and the Price of Misprediction. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA (LIPIcs, Vol. 151).* Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:18. https://doi.org/10.4230/LIPIcs.ITCS.2020.14

[82] Michael Mitzenmacher and Sergei Vassilvitskii. 2020. Algorithms with Predictions. In *Beyond the Worst-Case Analysis of Algorithms*, . Cambridge University Press, 646–662. https://doi.org/10.1017/9781108637435.037

[83] C. Mohan and Frank E. Levine. 1992. ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 2-5, 1992.* ACM Press, 371–380. https://doi.org/10.1145/130283.130338

[84] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020.* ACM, 985–1000. https://doi.org/10.1145/3318464.3380579

[85] Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc Friedman, and Alekh Jindal. 2021. Steering Query Optimizers: A Practical Take on Big Data Workloads. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021.* ACM, 2557–2569. https://doi.org/10.1145/3448016.3457568

[86] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. to appear. Flow-Loss: Learning Cardinality Estimates That Matter. *Proc. VLDB Endow.* (to appear).

[87] Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2020. Cost-Guided Cardinality Estimation: Focus Where it Matters. In *36th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2020, Dallas, TX, USA, April 20-24, 2020.* IEEE, 154–157. https://doi.org/10.1109/ICDEW49219.2020.00034

[88] Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2020. Cost-Guided Cardinality Estimation: Focus Where It Matters. In *Workshop on Self-Managing Databases (SMDB @ ICDE '20).*

[89] Thomas Neumann and Sebastian Michel. 2008. Smooth Interpolating Histograms with Error Guarantees. In *Sharing Data, Information and Knowledge, 25th British National Conference on Databases (BNCOD '08).* 126–138. https://doi.org/10.1007/978-3-540-70504-8_12

[90] Harrie Oosterhuis, J. Shane Culpepper, and Maarten de Rijke. 2018. The Potential of Learned Index Structures for Index Compression. In *Proceedings of the 23rd Australasian Document Computing Symposium, ADCS 2018, Dunedin, New Zealand, December 11-12, 2018.* ACM, 7:1–7:4. https://doi.org/10.1145/3291992.3291993

[91] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2019. An Empirical Analysis of Deep Learning for Cardinality Estimation. *arXiv:1905.06425 [cs]* (Sept. 2019). arXiv:1905.06425 [cs]

[92] Jennifer Ortiz, Brendan Lee, Magdalena Balazinska, and Joseph L. Hellerstein. 2016. PerfEnforce: A Dynamic Scaling Engine for Analytics with Performance Guarantees. *arXiv:1605.09753 [cs]* (May 2016). arXiv:1605.09753 [cs]

[93] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2018. QuickSel: Quick Selectivity Learning with Mixture Models. *arXiv:1812.10568 [cs]* (Dec. 2018). arXiv:1812.10568 [cs]

[94] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings.* www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p42-pavlo-cidr17.pdf

[95] Andrew Pavlo, Evan P. C. Jones, and Stan Zdonik. 2011. On Predictive Modeling for Optimizing Transaction Execution in Parallel OLTP Systems. *PVLDB* 5, 2 (2011), 86–96. https://doi.org/10.14778/2078324.2078325

[96] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.* ACM, 1981–1984. https://doi.org/10.1145/3299869.3320212

[97] Jack W. Rae, Sergey Bartunov, and Timothy P. Lillicrap. 2019. Meta-Learning Neural Bloom Filters. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97).* PMLR, 5271–5280. http://proceedings.mlr.press/v97/rae19a.html

[98] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy M. Lohman. 2002. Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3-6, 2002.* ACM, 558–569. https://doi.org/10.1145/564691.564757

[99] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2020. A Computational Approach to Packet Classification. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20).* Association for Computing Machinery, New York, NY, USA, 542–556. https://doi.org/10.1145/3387514.3405886

[100] Ibrahim Sabek, Kapil Vaidya, Dominik Horn, Andreas Kipf, and Tim Kraska. 2021. When Are Learned Models Better Than Hash Functions?. In *International Workshop on Applied AI for Database Systems and Applications (AIDB@VLDB) (AIDB '21).*

[101] Zahra Sadri, Le Gruenwald, and Eleazar Leal. 2020. DRLindex: deep reinforcement learning index advisor for a cluster database. In *IDEAS 2020: 24th International Database Engineering & Applications Symposium, Seoul, Republic of Korea, August 12-14, 2020.* ACM, 11:1–11:8. https://dl.acm.org/doi/10.1145/3410566.3410603

[102] Michael Schaarschmidt, Alexander Kuhnle, Ben Ellis, Kai Fricke, Felix Gessert, and Eiko Yoneki. 2018. LIFT: Reinforcement Learning in Computer Systems by Learning From Demonstrations. *arXiv:1808.07903 [cs, stat]* (Aug. 2018). arXiv:1808.07903 [cs, stat]

[103] Yangjun Sheng, Anthony Tomasic, Tieying Zhang, and Andrew Pavlo. 2019. Scheduling OLTP transactions via learned abort prediction. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 1:1–1:8. https://doi.org/10.1145/3329859.3329871

[104] Benjamin Spector, Andreas Kipf, Kapil Vaidya, Chi Wang, Umar Farooq Minhas, and Tim Kraska. 2021. Bounding the Last Mile: Efficient Learned String Indexing. In *International Workshop on Applied AI for Database Systems and Applications (AIDB@VLDB) (AIDB '21)*.

[105] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO - DB2's LEarning Optimizer. In *VLDB (VLDB '01)*. 19–28.

[106] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-Based Cost Estimator. *Proceedings of the VLDB Endowment* 13, 3 (Nov. 2019), 307–319. https://doi.org/10.14778/3368289.3368296

[107] Chuzhe Tang, Youyun Wang, Zhiyuan Dong, Gansen Hu, Zhaoguo Wang, Minjie Wang, and Haibo Chen. 2020. XIndex: A Scalable Learned Index for Multicore Data Storage. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (San Diego, California) *(PPoPP '20)*. Association for Computing Machinery, New York, NY, USA, 308–320. https://doi.org/10.1145/3332466.3374547

[108] Kapil Vaidya, Eric Knorr, Michael Mitzenmacher, and Tim Kraska. 2021. Partitioned Learned Bloom Filters. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=6BRLOfrMhW

[109] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*. IEEE Computer Society, 101–110. https://doi.org/10.1109/ICDE.2000.839397

[110] Youyun Wang, Chuzhe Tang, Zhaoguo Wang, and Haibo Chen. 2020. SIndex: a scalable learned index for string keys. In *APSys '20: 11th ACM SIGOPS Asia-Pacific Workshop on Systems, Tsukuba, Japan, August 24-25, 2020*. ACM, 17–24. https://dl.acm.org/doi/10.1145/3409963.3410496

[111] Xingda Wei, Rong Chen, and Haibo Chen. 2020. Fast RDMA-based Ordered Key-Value Store using Remote Learned Cache. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 117–135. https://www.usenix.org/conference/osdi20/presentation/wei

[112] Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner. 2019. Cardinality Estimation with Local Deep Learning Models. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM '19)*. Association for Computing Machinery, Amsterdam, Netherlands, 1–8. https://doi.org/10.1145/3329859.3329875

[113] Jiacheng Wu, Yong Zhang, Shimin Chen, Yu Chen, Jin Wang, and Chunxiao Xing. 2021. Updatable Learned Index with Precise Positions. *Proc. VLDB Endow.* 14, 8 (2021), 1276–1288. http://www.vldb.org/pvldb/vol14/p1276-wu.pdf

[114] Wenkun Xiang, Hao Zhang, Rui Cui, Xing Chu, Keqin Li, and Wei Zhou. 2019. Pavo: A RNN-Based Learned Inverted Index, Supervised or Unsupervised? *IEEE Access* 7 (2019), 293–303. https://doi.org/10.1109/ACCESS.2018.2885350

[115] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. 2020. Qd-tree: Learning Data Layouts for Big Data Analytics. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 193–208. https://doi.org/10.1145/3318464.3389770

[116] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *arXiv:2006.08109 [cs]* (June 2020). arXiv:2006.08109 [cs]

[117] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *Proceedings of the VLDB Endowment* 13, 3 (Nov. 2019), 279–292. https://doi.org/10.14778/3368289.3368294