Learned Query Optimizer: What is New and What is Next

Rong Zhu
red.zr@alibaba-inc.com
Alibaba Group
Hangzhou, China

Lianggui Weng
Alibaba Group
Hangzhou, China
lianggui.wlg@alibaba-inc.com

Bolin Ding
bolin.ding@alibaba-inc.com
Alibaba Group
Hangzhou, China

Jingren Zhou*
jingren.zhou@alibaba-inc.com
Alibaba Group
Hangzhou, China

ABSTRACT

In recent times, learned query optimizer has becoming a hot research topic in learned databases. It serves as the most suitable experimental plots for utilizing numerous machine-learning techniques and exhibits its superiority with enough evidence. In this tutorial, we aim to provide a wide and deep review and analysis on this field, ranging from theory to practice. At first, we would categorize and introduce representative methods for each learned component in the query optimizer, as well as for the end-to-end learned query optimizer. Then, we describe some benchmark evaluations and prototype applications. Their results have exhibited the bright future of applying learned query optimizers in practice. Based on them, we describe a cutting edge system with step-by-step guidelines. It is a middleware proposed recently to reduce the difficulties of developing and deploying learned algorithms in databases. It would help researchers to iterate their work and make learned query optimizers truly applicable in production. Finally, we summarize and point out several future directions. We hope this tutorial could inspire and guide both researchers and engineers working on learned query optimizers, as well as other contexts in learned databases.

CCS CONCEPTS

• **Information systems** → **Data management systems**.

KEYWORDS

query optimizer, machine learning, AI4DB

ACM Reference Format:
Rong Zhu, Lianggui Weng, Bolin Ding, and Jingren Zhou. 2024. Learned Query Optimizer: What is New and What is Next. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3626246.3654692

1 INTRODUCTION

Query optimizer, the core part of DBMS and big data processing platform, directly determines the plan quality and system performance. Although it has been extensively studied and refined, the intrinsic nature of varied data and query workload pose great challenges. Until now, it is a consensus that the performance of query optimizer is not fully favorable, especially on complex and/or tail cases [37].

Recently, the development of machine learning (ML), especially deep learning, exhibits great superiority in the data processing area. This cross-filed, called "AI4DB", has become a hot spot in the database research field. In AI4DB, a learned query optimizer serve as a *pioneer*. It provides suitable experimental plots for various kinds of ML techniques, including supervised, unsupervised, reinforcement learning and etc. By some surveys [54, 77], more than 100+ papers have been published on this topic in the last decade. Meanwhile, it is still growing fast and exhibits bright futures in some prototype applications. Therefore, we propose this tutorial to summarize the advances, analyze the status and guide the development on the learned query optimizer. Our tutorial contains the following content in terms of both theoretical and practical perspectives:

1) A comprehensive review and deep analysis on the learned methods for query optimizer. In the last decade, numerous ML-based approaches have been proposed to optimize each component in query optimizer, as well as the end-to-end query performance. Their scope, technical routines and properties are very different. We try to categorize and introduce the representative methods in each class, together with some benchmark evaluation results and prototype applications, to exhibit the advantages and disadvantages of each method. This would give the audience a deep and holistic understanding to the field of learned query optimizers.

2) An in-depth introduction on the cutting edge system to deploy learned query optimizers in actual databases. Deploying learned query optimizers to benefit the real-world DBMS is the ultimate goal, but it is a very difficult task. We summarize the challenges for the actual deployment of learned query optimizers. Then, we introduce an cutting edge system called PilotScope [80], which is a middleware that largely reduces the difficulties of developing and deploying AI4DB algorithms in databases. We describe its system architecture, workflow, programming APIs and step-by-step guidelines on sample applications. This would help researchers to iterate their work and make learned query optimizers truly applicable in production.

3) A summary on the promising future works of learned query optimizers. Based on 1) and 2), we point out a series of important directions, including but not limited to the model and system design, evaluation, application and deployment. We hope this could inspire and guide the following researchers and engineers to do more work to make learned query optimizer more powerful and applicable.

*Corresponding author.

SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0422-2/24/06
https://doi.org/10.1145/3626246.3654692

2 ALGORITHMS, BENCHMARKS AND PROTOTYPE APPLICATIONS

In this tutorial, we focus on the seminar structure of query optimizer, *e.g.*, PostgreSQL and Calcite, following the volcano framework [11]. On a high level, the query optimizer module is composed of three components, namely *cardinality estimator*, *cost model* and *plan enumerator*. For any input SQL query Q, the plan enumerator first explores the plan space with some algorithms, *e.g.*, dynamic programming or greedy search, to generate a number of candidate plans with different join orders. Then, the cost model, together with the cardinality estimator, is applied for plan selection. Specifically, for each sub-query Q' of the input query Q, the cardinality estimator could estimate the cardinality of Q' without executing Q'. Based on the estimated cardinality, the cost model could predict the cost of each candidate plan. Finally, the plan with the minimum estimated cost is returned for execution.

Within the query optimizer, there exists ample room to apply ML techniques to improve its performance. This work covers multiple different aspects: from model design, benchmark evaluation to prototype applications; from individually learned components to end-to-end learned query optimizers; from unsupervised or supervised models to reinforcement learning policies; and from statistical models to deep models. In the following content, we try to carefully organize them and provide some in-depth insights.

2.1 Learned Methods for Each Component

In this subsection, we focus on the ML techniques designed for each component in the query optimizer, namely *learned cardinality estimators*, *learned cost models* and *learned join order search methods* in the plan enumerator.

2.1.1 Learned Cardinality Estimators. Given the data D and query Q, the cardinality estimator aims at building a sketch-based synopses using their information to estimate the number of tuples in D satisfying Q. Traditional methods mainly utilize very simple statistical models, such as one or multi-dimensional histogram, or various sampling techniques. ML-based methods build more complex models. We list these methods in Table 1. They could be categorized into three main classes as follows:

Query-Driven Methods. They learn supervised models directly mapping featurized query to its cardinality. Some works directly apply the traditional statistical models. At the very early stage, [36] featurizes each query into a number of parameters and builds a linear regression model to map parameters to the cardinality. Later, [10] and [9] propose to use tree-based ensembles and XGBoost to model the mapping functions. After that, QuickSel [47] uses a mixture model with overlapping to approximate the probability density function. It could be refined significantly faster to yield increasingly more accurate selectivity estimates over time.

Some other works apply DNNs to model the mapping functions. [32] applies fully connected neural networks to learn the cardinality of range queries for the first time. [23] designs the more complex multi-set convolutional network (MSCN) to extract features from tables, range predicates and join conditions and combine them together to learn the cardinality.

Table 1: A list of learned cardinality estimators.

Category	Method	Applied ML Techniques
Query-Driven (Statistical Model)	[36]	Linear Model
	[10]	Tree-based Ensembles
	[9]	XGBoost
	QuickSel [47]	Mixture Model
Query-Driven (DNN-Based Model)	[32]	Fully Connected Neural Network
	MSCN [23]	Multi-Set Convolutional Network
	[22]	Adding Pooling Layers
	CRN [13]	Learning Containment Rate
	Robust-MSCN [45]	Query Masking
	GL+[52]	Segmentation Technique
	Fauce [33]	Ensemble of Deep Models
	NNGP [75]	Bayesian Deep Learning
	LPCE [59]	Query Re-Optimization
Data-Driven (Kernel-Based)	[14]	Kernel Density Function
	[21]	Kernel Density Function
Data-Driven (Auto-Regression Model)	Naru [71]	Single Table
	NeuroCard [70]	Multi-Tables
	IAM [40]	Adding Gaussian Mixture Model
Data-Driven (Probabilistic Graphical Model)	BayesNet [57]	Bayesian Networks
	BayesCard [65]	Revitalized Bayesian networks
	DeepDB [17]	Sum-Product Network
	FLAT [81]	FSPN
	FactorJoin [64]	Factor Graph and Join Histogram
Data-Driven	FACE [60]	Normalizing Flow
	Iris [35]	Summarization Models
Hybrid	UAE [63]	Deep Auto-Regression Model
	GLUE [82]	Merging Single Table Results
	ALECE [30]	Attention on Transformer Model

Based on them, [22] proposes to add pooling layers with the fully connected neural networks to only capture strong intra-table correlations. CRN [13] improves over MSCN by learning the containment rate between pair of queries (Q_1, Q_2), i.e., the percentage of result tuples in Q_1 that are also result tuples of Q_2. The trained network could also be generally applied to learn cardinality of any query. [45] proposes Robust-MSCN using the query masking technique to adapt to workload changes.

Recent works make further extensions. GL+ [52] integrates DNNs with segmentation techniques to resolve the data hungry problem. To resolve the uncertainty of the predicted results, Fauce [33] uses an ensemble of deep models to estimate the cardinality and the corresponding uncertainty. Later, NNGP [75] employs Bayesian deep learning (BDL) to bridge between Bayesian inference and deep learning. This algorithm inherits the advantages of the Bayesian approach while keeping a universal approximation of neural networks. LPCE [59] adopts a query re-optimization methodology. It consists of an initial model to estimate cardinality before query execution, and a refinement model to progressively refine the cardinality estimations using the actual cardinalities of the executed operators.

Data-Driven Methods. They learn unsupervised models of the joint data distribution so the probability (cardinality) of any query could be computed. The main modeling tools and the corresponding methods are listed as follows:

1) [14] and [21] use different kernel density functions centered around sampled points to estimate the cardinality.

2) Naru [71] and NeuroCard [70] use deep auto-regression models to decompose and represent the joint data distribution. Later, IAM [40] integrates Gaussian mixture model with auto-regression model to fit the distribution of continuous attributes and reduce their domain size.

3) BayesNet [57] and BayesCard [65] apply traditional or revitalized Bayesian networks to model the joint data distribution, respectively.

4) DeepDB [17] applies sum-product networks (SPNs) to represent the joint distribution. FLAT [81] further extends this method by replacing SPN to FSPN, a new kind of PGM proposed in [67].

5) FactorJoin [64] combines the factor graph with join histogram together to efficiently handle joins and accurately capture attribute correlation.

6) FACE [60] leverages the normalizing flow based models to learn a continuous joint distribution for relational data.

7) Iris [35] uses different summarization models for different set of columns and maintains them together to answer cardinality estimation queries.

Besides these works, Astrid [48] applies natural language processing techniques with deep models to learn cardinality of queries with string predicates. DREAM [26] builds deep models to learn cardinality of approximate substring queries. LMKG [8] considers cardinality estimation on knowledge graphs.

Hybrid Methods. They extract information from both queries and data to estimate the cardinality, including:

1) The UAE method [63] proposes a unified deep autoregressive model to learn the joint data distribution from both the data and query workload. It applies differentiable progressive sampling technique to inject the supervised query information into the deep autoregressive model of data.

2) The GLUE method [82] proposes a general framework merging single table estimation results produced by any cardinality estimation method to predict join query size.

3) The ALECE method [30] tries to discover the hidden relationships between queries and underlying data using attention mechanisms and transformer models. It applies a data-encoder module to learn data aggregations representing implicit correlations among attributes and a query-analyzer module to map featuized queries and data aggregations to predict the cardinality of queries. ALECE can be applied to both static and dynamic data settings in the same manner and is shown to attain nearly optimal performance.

Extensions. Besides these, some works are proposed to enhance their performance in other perspectives. [74] proposes a model advisor, called AutoCE, which uses deep metric learning to learn a recommendation model to adaptively select the best model for each dataset. [44] introduces a new loss function, Flow-Loss, to approximate the optimizer's cost model and enforces the learned cardinality estimators to pay more attention to queries that matter to the final plan quality. [28] generates additional queries when limited examples are available from the new workload and carefully picks which queries to use to update the cardinality estimation model. [42] proposes techniques to featurize queries with mixed combinations of conjunctive and disjunctive predicates.

There also exist some theoretical works on learned cardinality estimators. [19] proves that the selectivity function of a range space with bounded VC-dimension is learnable, using classic learning theory for real-valued functions based on shattering dimension. [55] investigates how to quantify the uncertainty associated with the cardinality estimate of a learned model through prediction intervals.

2.1.2 Learned Cost Models. Let P be a physical plan for the query Q. Based on Q's cardinality and P's operators, the cost model returns a cost value to predict its execution time. Traditional cost models are rule based and driven by experience. Learned cost models can be categorized into two classes as follows:

Cost Models for Single Query. In fact, we can directly leverage learned cardinality for cost estimation, but this leads to accumulative errors. To resolve this problem, existing approaches often apply deep models to capture the plan structure for cost estimation. [39] proposes to use the tree convolutional network to predict the plan cost. [51] proposes the Tree-LSTM model to learn an end-to-end cost model. The work Saturn [34] encodes each query plan tree into a compressed vector using a traversal-based query plan auto-encoder to cope with diverse plan structures. The compressed vectors can be leveraged to distinguish different query types, which is highly useful for downstream tasks. Later, [76] uses the transformer to learn the embeddings of plans. The learned embeddings could be applied to cost estimation, as well as other tasks in query optimization.

Besides these works purely on cost models, other works make some extensions. [16] introduces zero-shot cost models, which enable learned cost estimation that generalizes to unseen databases. [68] proposes a cost model for graph databases. [49] investigates two key questions: 1) can we learn accurate cost models for big data systems, and 2) can we integrate the learned models within the query optimizer.

Cost Models for Concurrent Queries. Cost models over concurrent queries are non-trivial as it is rather difficult to characterize the correlations between different queries. GPredictor [78] utilizes the graph neural network to capture the query relationships and estimate the query performance accurately. Prestroid [20] leverages the tree convolution based approach to estimate the concurrent query performance in a cloud environment. [31] proposes a resource-aware deep learning model. It embeds the query plans with features extracted from the allocated resources. Then, a deep learning model with an adaptive attention mechanism is trained to predict the execution time of query plans.

2.1.3 Learned Join Order Search Methods. The join order search method enumerates all candidate plans in the plan search space to find a near-optimal plan with the minimum estimated cost. Traditional approaches typically explore the search space using some pruning rules, which are efficient but may miss good plans. ML-based methods could learn from history and overcome the bias in the estimated cost. They could be mainly categorized into two classes as follows:

Offline Learning Methods. This class of methods learns from the previous queries to improve the performance of future ones. DQ [15] and ReJoin [24] are proposed to use neural network and reinforcement learning to optimize the join orders, but the simple neural architecture limits their learning ability. Hence, RTOS [73] proposes a model that utilizes the TreeLSTM to represent the join state. Later, JOGGER [2] proposes a novel framework with graph-based representation to better capture the join tree structure. Further, the framework MLMTF [66] proposes a pre-trained model to represent shared knowledge across data and tasks, which would

be fine-tuned for a specific data. Upon it, several small models are learned together using multi-task learning for each task, i.e., cardinality estimation, cost model and join order search, respectively.

Online Learning Methods. This class of methods learns a join order through adaptive query processing, which can change the order even during the execution of queries. Eddy-RL [58] models the query execution as a reinforcement learning problem and automatically learns how to adjust the join order during execution using Q-learning. SkinnerDB [56] optimizes the join order on the fly with the help of a Monte-Carlo tree search based approach, where different join orders are tried in each time slice.

2.2 End-to-End Learned Query Optimizers

Besides learned techniques for each individual component, there also proposed a number of methods to learn end-to-end query optimizers. Their procedures can be generalized into a unified framework with two main steps. For the input query Q, a learned query optimizer first generates a set of candidate plans $\mathcal{P}_Q = \{P_0, P_1, \ldots, P_k\}$ using some plan exploration strategies. The exploration strategies are different from the plan enumeration methods in traditional query optimizer. Then, a learned risk model M_r, i.e., a complex ML-based model, is applied for plan selection. M_r can predict the goodness of each plan in $\mathcal{P}_Q$ in terms of its cost. The best plan $P_r \in \mathcal{P}_Q$ minimizing the predicted cost is selected for execution. Different learned query optimizers apply different plan exploration strategies and risk models, but they can all be subsumed under this framework. We describe the details as follows.

2.2.1 Different Learned Query Optimizes. One class of learned query optimizers explores all plans from scratch by themselves. Neo [38] and Balsa [69] generate the candidate plans $\mathcal{P}_Q$ by best-first and beam search strategy, respectively. Then, their risk models apply the tree convolution network [41] to predict the execution time of each plan $P \in \mathcal{P}_Q$. LOGER [3] applies the ϵ-beam search strategy for candidate plan generation. It models each plan by the graph transformer to capture relationships between tables and predicates to predict its latency. BASE [5] explores all plans using the same method as Neo. However, it learns a calibrated cost model to approach the latency.

Another class of learned query optimizers tries to steer or aid the native query optimizer for plan generation. Bao [37] steers the native traditional query optimizer with different hints to enable or prohibit certain physical operators to generate different candidate plans. Its risk model is also based on a tree convolution network with simpler features. HyperQO [72] uses different leading hints to control the join order of tables to collect candidate plans. Its risk model relies on the multi-head LSTM structure for predicting execution time. Lero [79] applies the estimated cardinality as the tuning knob for generating candidate plans. It scales the estimated cardinality by some factors to produce different (possibly better) plans. Unlike other works, Lero trains a pairwise classification model. For each pair of plans $P, P' \in \mathcal{P}_Q$, the model learns to predict which plan is better in terms of cost. The plan surpassing the most number of other plans is then selected to execute. LEON [4] applies the same method, *i.e.*, dynamic programming, in traditional query optimizer for plan generation. However, it also applies a pairwise comparison model for plan selection.

Besides them, the framework PerfGuard [18] could support any method to generate candidate plans. It also applies the pairwise model, which incorporates graph convolution networks, for plan selection. AutoSteer [1] extends Bao with the new capabilities, *e.g.*, automated hint-set discovery, to minimize integration effort and facilitate usability in more database systems.

2.2.2 Performance Regression Elimination Techniques. Despite such learned query optimizers have shown superiority in some benchmarks, its performance regression seems inevitable for some queries due to model under-fitting and difficulty in generalization to unseen data. In recent times, there has been a significant shift towards enhancing the robustness of such learned models to eliminate performance regression.

For instance, Warper [29] enhances the cardinality estimation model by generating additional queries to update it when data or workload drift is detected. DDUp [25] uses a two-stage sampling procedure to test whether the model should be updated w.r.t. dynamic data. These methods are post-processing techniques used for model updating. Another routine is to detect and eliminate regression before query execution.

The HyperQO work [72] tries to apply the ensemble method. It deploys a multi-head LSTM model to learn multiple prediction results for each plan $P \in \mathcal{P}_Q$. All candidate plans P with a large variance are filtered and the remaining plan with the best average estimated cost is selected to execute. A very recent work Eraser [62] aims at eliminating performance regressions while still attaining considerable overall performance improvement. It applies a two-stage strategy, including: 1) a coarse-grained filter that selects to remove all highly risky plans with unseen feature values; and 2) a more fine-grained plan cluster method to group plans according to the prediction quality for selecting the final execution plan. Eraser can be deployed as a plugin on top of any learned query optimizer to select more reliable plans.

2.3 Benchmark Evaluations

Except for these algorithmic works, the research community also proposes some benchmarks to comprehensively evaluate learned methods on query optimizers.

[61] explores several learned cardinality estimators on a single table to investigate whether they are ready for system deployment in both static and dynamic environments. It further analyzes different properties of learned cardinality estimators that affect deployment. Then, [53] performs a design space exploration of learned cardinality estimators to have a comprehensive comparison of these approaches. The results could provide a guidance for practitioners to decide what method to use under various practical scenarios.

For traditional query optimizers, the synthetic benchmarks such as TPC-H [7], TPC-DS [6] and Star Schema benchmarks (SSB) [46] and the real-world IMDB dataset with its JOB workload [27] are often applied for benchmark evaluation. However, these benchmarks either make oversimplified assumptions on the joint distribution of attributes (TPC-H, TPC-DS and SSB) or contain very simple forms of joins (IMDB with JOB), which cannot reflect the actual performance of learned methods on complex real-world data and varied join settings. To resolve this problem, [12] proposes a new benchmark, called STATS, which is more complex and close to the

real-world settings. Based on this, it integrates multiple learned query optimizers into the real-world DBMS, *i.e.*, PostgreSQL, to conduct an end-to-end performance evaluation. The results are very convincing to exhibit the pros and cons of each method.

2.4 Prototype Applications

Due to the superiority of learned query optimizers, we gradually move from designing to applying and deploying learned query optimizers in real-world scenarios. In this part, we introduce some prototype applications on deploying learned query optimizers in the industry production environments. We find two representative works on applying the learned cost model and Bao in SCOPE, the distributed data processing platform in Microsoft:

1) [50] tries to learn accurate cost models for big data systems and integrate them within the original query optimizer. It analyzes the workload patterns to learn a large number of individual cost models and combine them together to achieve high accuracy and coverage over a long period. Meanwhile, the models are integrated into the Cascade-style query optimizer of SCOPE and exhibit the superiority.

2) [43] applies BAO's idea proposed in [37] to steer SCOPE's query optimizer. It tries to bridge the gaps between the research assumptions and industry scenarios. Specifically, a rule signature method is proposed to collect a small number of configuration hints that could be tuned to optimize per-query performance.

3 PILOTSCOPE: A SYSTEM FOR DEPLOYMENT

Although learned query optimizers, as well as the AI4DB field, have rapidly developed in the last decade. Deploying ML algorithms into actual databases is still prohibitively difficult due to the complexity of database systems, the difference between ML and DB programming paradigms, and the diversity of ML models. Even for each specific database engine, deploying specific ML algorithms and models still requires close cooperation between ML and DB developers and heavy engineering costs, such as the previous prototype applications in [43, 50]. To this end, a very recent work [80] designs and develops the PilotScope system. It is an AI4DB middleware with a programming model that largely reduces the difficulties of developing and deploying AI4DB algorithms in databases. The PilotScope is already open-source at https://github.com/alibaba/pilotscope. By applying PilotScope, we could attain the following benefits:

- PilotScope is easy-to-use for database users. The database user could access PilotScope to start any AI4DB task as needed and operate the database as usual. The execution of any AI4DB algorithm is totally transparent to the database user.
- PilotScope enables ML and DB developers to work independently to play their own strengths in developing ML and DB programs. They do not need to know the details on the other side.
- PilotScope attains high generality. It could support deploying a variety of AI4DB tasks on different database systems. Moreover, it could be easily extended to new tasks and new systems.

In this section, we present the PilotScope system architecture and workflow. After that, we present a step-by-step demonstration to guide users to apply PilotScope on sample applications.

3.1 Architecture and Workflow

System Architecture. PilotScope provides a *console* to operate the whole system and manages multiple AI4DB *drivers* and the *DB interactor*. The DB interactor contains an *interface* connecting AI4DB drivers with databases. Each database steered by the AI4DB drivers is attached with its specific *implementations of DB interactor*.

Specifically, in PilotScope, each task, which targets to replace a database component, is packaged as a *driver*. Each driver contains the *algorithm* describing the algorithmic workflow, *e.g.*, how to learn to generate the execution plan in a learned query optimizer, and one or more *ML models* to be consulted in the algorithm. The algorithms and models in the driver are all written in an AI-friendly language, e.g., Python. The PilotScope integrates a *runtime environment* with third-party dependencies and libraries to support their execution.

The interface of DB interactor shields the underlying details of different databases and serves as a unified bridge for drivers to interact with the databases. It abstracts two operators, namely *push* and *pull*, to allow drivers to enforce actions and exchange data with the databases. The DB interactor, as well as the operators, is implemented in different ways on different databases. However, all of them satisfy the same interface of DB interactor and fulfill the required functions. In such a way, each driver could apply the interface to steer different databases with minimum modifications. In practice, the implementations are often developed as lightweight patches to the database codebase so that the changes incurred to the database kennel are minimal.

Workflow. After the database user establishes the connection and starts a driver through PilotScope console, the driver starts to collect the pre-defined training data from databases. For example, the query execution time would be collected for training the time prediction model in the learned query optimizer. After collecting enough data, the driver would start to train each model using the provided training function. After that, the algorithm in the driver would wait to be executed. When needed, it would be invoked by PilotScope to replace the original database component, *e.g.*, the learned query optimizer is called whenever the database user executes a SQL query. It obtains the inputs from the injection interface (*e.g.*, the query), executes its algorithm in PilotScope, consults the ML models using the provided model inference functions and interacts with the database through push/pull operators to fulfill its job. After it finishes executing, the injection interface sends its result (*e.g.*, the selected execution plan) to the database. For some drivers, the ML models are updated in the background to keep track of database changes.

3.2 Demonstration on Sample Applications

In this section, we provide a thorough description to guide developers and researchers to apply PilotScope to develop and deploy their ML methods onto actual databases. The audience could try to apply PilotScope together with our interpretation.

At first, we show users how to install and configure PilotScope system. The PilotScope could support any database. Here we demonstrate its functions on the well-known database PostgreSQL. The PilotScope repository has already provided a docker image with a PostgreSQL database pre-configured with PilotScope patches. The users could download this docker image to get started quickly.

Besides, we would also explain how to configure PilotScope to integrate with PostgreSQL from scratch. Users could experience more customized installation through this process.

Second, we introduce the programming APIs applied in PilotScope to develop a driver. The APIs in PilotScope are very simple. For each new driver, we only need to override: 1) an init() function to make some preparations and specify its injection type, *e.g.*, a learned cardinality estimator or a learned end-to-end query optimizer, and 2) an algo() function to describe the AI4DB algorithm that applies ML models and DB interactor operators to accomplish the task. Then, the driver would be called by PilotScope when needed.

For the interaction in PilotScope, we use the concept of *session* to define each interaction process between ML algorithm and database. For each session, the PilotScope creates a new connection *e.g.*, a database session, to the database. In this session, we could enforce the actions to databases, *e.g.*, updating its configurations, sending or acquiring data, using multiple push() and pull() operators.

Third, we present the details on applying PilotScope in two representative applications, namely learned cardinality estimator and learned end-to-end query optimizer.

For the learned cardinality estimator, we could apply the same driver to support any cardinality estimation method. The injection interface could support replacing the cardinality of all sub-queries generated by the learned cardinality estimator methods in a batch manner.

For the learned end-to-end query optimizer, we present the drivers of Bao [37] and Lero [79]. They apply the same injection interface to obtain the input SQL query and return the generated plan for execution. In the driver, Bao and Lero apply the push/pull interaction operators to tune the hints and cardinality and obtain the candidate plans, respectively.

Fourth, we execute PostgreSQL with different learned cardinality estimators and end-to-end query optimizers on benchmarks in PilotScope. We report the evaluation results to exhibit the pros and cons of all learned methods. This would provide valuable insights to researchers to iterate the following research works.

4 TUTORIAL INFORMATION

Length of Time. Our tutorial prefers a duration of 3 hours, but could also be compressed to 1.5 hours. In the option of 3 hours, we would spend 50 minutes on introducing the learned techniques for individual components, 40 minutes on end-to-end learned query optimizer, 20 minutes on benchmarks and prototype applications, 20 minutes on the PilotScope system architecture and workflow, 40 minutes on the demonstration and the final 10 minutes on the conclusion and future work. In the option of 1.5 hours, we would omit some details to spend 70 minutes on introducing the learned techniques, 10 minutes for the benchmarks and prototype applications and only reserve 10 minutes on briefly introducing PilotScope without the demonstration.

Intended Audience. Learned query optimizer, as well as AI4DB, is a crossing field between ML, DB and system. Meanwhile, it is a hot topic in both academic and industry areas. We target to attract the audience from three fields:

1) ML and DB researchers who are interested in designing AI-driven techniques for databases, especially for query optimizer. This tutorial would give them a comprehensive picture and provide guidelines for their future work.

2) System researchers who are interested in developing AI4DB systems. This tutorial would provide the scenarios and requirements for deploying ML techniques in databases and guide them to design and optimize such systems.

3) System providers and engineers who are now using or eager to apply ML techniques in real-world industry data processing systems. This tutorial would help them to know about the current advances of learned query optimizers and guide them to apply PilotScope to resolve the practical deployment issues.

Difference with Previous Tutorials. Our tutorial has overlaps with two of our previous tutorials:

1) *AutoML: From Methodology to Application* in CIKM 2021. The CIKM tutorial focuses on AutoML techniques. ML-based cardinality estimation is discussed as an application. The main topics are totally different from our SIGMOD tutorial.

2) *Learned Query Optimizer: At the Forefront of AI-Driven Databases* in EDBT 2022. This SIGMOD tutorial has made substantial extensions in comparison to the one in EDBT. First, we add a sufficient number of new works proposed in recent years, including [1, 3–5, 8, 13, 16, 18, 19, 22, 25, 26, 28–31, 33–35, 40, 44, 45, 47–49, 52, 55, 59, 60, 62, 64, 68, 69, 72, 74–76, 79], which occupies more than 45% of the works reviewed in this tutorial. Second, we pay more attention on practically deploying learned techniques in databases. We add the new content to introduce and demonstrate the PilotScope system in this tutorial.

5 BIOGRAPHY

Rong Zhu is a research scientist in the Institute for Intelligent Computing, Alibaba Group. He obtained his PhD degree in Department of Computer Science and Technology, Harbin Insitute of Technology in 2019. His research interests lie in AI4DB and data mining.

Lianggui Weng is an engineer in the Institute for Intelligent Computing, Alibaba Group. He obtained his Master degree in Department of Computer Science and Technology, Huazhong University of Science and Technology in 2021. His research interests lie in AI4DB and database system.

Bolin Ding is a senior research scientist in the Institute for Intelligent Computing, Alibaba Group. He completed his PhD in Computer Science at University of Illinois at Urbana-Champaign in 2012. His research centers on large-scale data management and analytics, with focuses on data privacy for databases, AI for systems, and query optimization.

Jingren Zhou is Senior Vice President at Alibaba and CTO of Alibaba Cloud. He has managed several core technical divisions at Alibaba to drive data intelligent infrastructure and applications in e-commerce and cloud business, including big data and AI infrastructure, search and recommendation, and advertising platform. His research interests include cloud-computing, databases, and large-scale machine learning. He received his PhD in Computer Science from Columbia University. He is a Fellow of IEEE.

REFERENCES

[1] Christoph Anneser, Nesime Tatbul, David E. Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. 2023. AutoSteer: Learned Query Optimization for Any SQL Database. *Proc. VLDB Endow.* 16, 12 (2023), 3515–3527. https://doi.org/10.14778/3611540.3611544

[2] Jin Chen, Guanyu Ye, Yan Zhao, Shuncheng Liu, Liwei Deng, Xu Chen, Rui Zhou, and Kai Zheng. 2022. Efficient Join Order Selection Learning with Graph-based Representation. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, Aidong Zhang and Huzefa Rangwala (Eds.). ACM, 97–107. https://doi.org/10.1145/3534678.3539303

[3] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. 2023. LOGER: A Learned Optimizer towards Generating Efficient and Robust Query Execution Plans. *Proc. VLDB Endow.* 16, 7 (2023), 1777–1789. https://www.vldb.org/pvldb/vol16/p1777-gao.pdf

[4] Xu Chen, Haitian Chen, Zibo Liang, Shuncheng Liu, Jianhong Wang, Kai Zeng, Han Su, and Kai Zheng. 2023. LEON:: A New Framework for ML-Aided Query Optimization. *Proc. VLDB Endow.* 16, 9 (2023), 2261–2273. https://www.vldb.org/pvldb/vol16/p2261-chen.pdf

[5] Xu Chen, Zhen Wang, Shuncheng Liu, Yaliang Li, Kai Zeng, Bolin Ding, Jingren Zhou, Han Su, and Kai Zheng. 2023. BASE: Bridging the Gap between Cost and Latency for Query Optimization. *Proc. VLDB Endow.* 16, 8 (2023), 1958–1966. https://www.vldb.org/pvldb/vol16/p1958-chen.pdf

[6] Transaction Processing Performance Council(TPC). 2023. TPC-DS Vesion 2 and Version 3. *http://www.tpc.org/tpcds/* (2023).

[7] Transaction Processing Performance Council(TPC). 2023. TPC-H Vesion 2 and Version 3. *http://www.tpc.org/tpch/* (2023).

[8] Angjela Davitkova, Damjan Gjurovski, and Sebastian Michel. 2022. LMKG: Learned Models for Cardinality Estimation in Knowledge Graphs. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*. OpenProceedings.org, 2:169–2:182. https://doi.org/10.48786/edbt.2022.07

[9] Anshuman Dutt, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2020. Efficiently Approximating Selectivity Functions using Low Overhead Regression Models. *Proc. VLDB Endow.* 13, 11 (2020), 2215–2228. http://www.vldb.org/pvldb/vol13/p2215-dutt.pdf

[10] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057. https://doi.org/10.14778/3329772.3329780

[11] Goetz Graefe. 1995. The cascades framework for query optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29. http://sites.computer.org/debull/95SEP-CD.pdf

[12] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (2021), 752–765. https://doi.org/10.14778/3503585.3503586

[13] Rojeh Hayek and Oded Shmueli. 2020. Improved Cardinality Estimation by Learning Queries Containment Rates. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*. OpenProceedings.org, 157–168. https://doi.org/10.5441/002/edbt.2020.15

[14] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM, 1477–1492. https://doi.org/10.1145/2723372.2749438

[15] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John P. Agapiou, Joel Z. Leibo, and Audrunas Gruslys. 2018. Deep Q-learning From Demonstrations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 3223–3230. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16976

[16] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *Proc. VLDB Endow.* 15, 11 (2022), 2361–2374. https://www.vldb.org/pvldb/vol15/p2361-hilprecht.pdf

[17] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proc. VLDB Endow.* 13, 7 (2020), 992–1005. https://doi.org/10.14778/3384345.3384349

[18] H. M. Sajjad Hossain, Marc T. Friedman, Hiren Patel, Shi Qiao, Soundar Srinivasan, Markus Weimer, Remmelt Ammerlaan, Lucas Rosenblatt, Gilbert Antonius, Peter Orenberg, Vijay Ramani, Abhishek Roy, Irene Shaffer, and Alekh Jindal. 2021. PerfGuard: Deploying ML-for-Systems without Performance Regressions, Almost! *Proc. VLDB Endow.* 14, 13 (2021), 3362–3375. http://www.vldb.org/pvldb/vol14/p3362-hossain.pdf

[19] Xiao Hu, Yuxi Liu, Haibo Xiu, Pankaj K. Agarwal, Debmalya Panigrahi, Sudeepa Roy, and Jun Yang. 2022. Selectivity Functions of Range Queries are Learnable. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 959–972. https://doi.org/10.1145/3514221.3517896

[20] Johan Kok Zhi Kang, Gaurav, Sien Yi Tan, Feng Cheng, Shixuan Sun, and Bingsheng He. 2021. Efficient Deep Learning Pipelines for Accurate Cost Estimations Over Large Scale Query Workload. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1014–1022. https://doi.org/10.1145/3448016.3457546

[21] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *Proc. VLDB Endow.* 10, 13 (2017), 2085–2096. https://doi.org/10.14778/3151106.3151112

[22] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned Cardinality Estimation: An In-depth Study. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 1214–1227. https://doi.org/10.1145/3514221.3526154

[23] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. 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/p101-kipf-cidr19.pdf

[24] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR* abs/1808.03196 (2018). arXiv:1808.03196 http://arxiv.org/abs/1808.03196

[25] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. *Proc. ACM Manag. Data* 1, 1 (2023), 33:1–33:27. https://doi.org/10.1145/3588713

[26] Suyong Kwon, Woohwan Jung, and Kyuseok Shim. 2022. Cardinality Estimation of Approximate Substring Queries using Deep Learning. *Proc. VLDB Endow.* 15, 11 (2022), 3145–3157. https://www.vldb.org/pvldb/vol15/p3145-jung.pdf

[27] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215. http://www.vldb.org/pvldb/vol9/p204-leis.pdf

[28] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 1920–1933. https://doi.org/10.1145/3514221.3526179

[29] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1920–1933. https://doi.org/10.1145/3514221.3526179

[30] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. *Proc. VLDB Endow.* 17, 2 (2023), 197–210. https://www.vldb.org/pvldb/vol17/p197-li.pdf

[31] Yan Li, Liwei Wang, Sheng Wang, Yuan Sun, and Zhiyong Peng. 2022. A Resource-Aware Deep Cost Model for Big Data Query Processing. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 885–897. https://doi.org/10.1109/ICDE53745.2022.00071

[32] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *Proceedings of 25th Annual International Conference on Computer Science and Software Engineering, CASCON 2015, Markham, Ontario, Canada, 2-4 November, 2015*. IBM / ACM, 53–59. http://dl.acm.org/citation.cfm?id=2886453

[33] Jie Liu, Wenqian Dong, Dong Li, and Qingqing Zhou. 2021. Fauce: Fast and Accurate Deep Ensembles with Uncertainty for Cardinality Estimation. *Proc. VLDB Endow.* 14, 11 (2021), 1950–1963. https://doi.org/10.14778/3476249.3476254

[34] Shuncheng Liu, Xu Chen, Yan Zhao, Jin Chen, Rui Zhou, and Kai Zheng. 2022. Efficient Learning with Pseudo Labels for Query Cost Estimation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, Mohammad Al Hasan and Li Xiong (Eds.). ACM, 1309–1318. https://doi.org/10.1145/3511808.3557305

[35] Yao Lu, Srikanth Kandula, Arnd Christian König, and Surajit Chaudhuri. 2021. Pre-training Summarization Models of Structured Datasets for Cardinality Estimation. *Proc. VLDB Endow.* 15, 3 (2021), 414–426. https://doi.org/10.14778/3494124.3494127

[36] Tanu Malik, Randal C. Burns, and Nitesh V. Chawla. 2007. A Black-Box Approach to Query Cardinality Estimation. In *Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*. www.cidrdb.org, 56–67. http://cidrdb.org/cidr2007/papers/cidr07p06.pdf

[37] 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

[38] 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

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

[40] Zizhong Meng, Peizhi Wu, Gao Cong, Rong Zhu, and Shuai Ma. 2022. Unsupervised Selectivity Estimation by Integrating Gaussian Mixture Models and an Autoregressive Model. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*. OpenProceedings.org, 2:247–2:259. https://doi.org/10.48786/edbt.2022.13

[41] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[42] Magnus Müller, Lucas Woltmann, and Wolfgang Lehner. 2023. Enhanced Featurization of Queries with Mixed Combinations of Predicates for ML-based Cardinality Estimation. In *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*. OpenProceedings.org, 273–284. https://doi.org/10.48786/edbt.2023.22

[43] Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc T. 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*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2557–2569. https://doi.org/10.1145/3448016.3457568

[44] Parimarjan Negi, Ryan C. Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *Proc. VLDB Endow.* 14, 11 (2021), 2019–2032. https://doi.org/10.14778/3476249.3476259

[45] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. 2023. Robust Query Driven Cardinality Estimation under Changing Workloads. *Proc. VLDB Endow.* 16, 6 (2023), 1520–1533. https://www.vldb.org/pvldb/vol16/p1520-negi.pdf

[46] Patrick E. O'Neil, Elizabeth J. O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 5895)*, Raghunath Othayoth Nambiar and Meikel Poess (Eds.). Springer, 237–252. https://doi.org/10.1007/978-3-642-10424-4_17

[47] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2020. QuickSel: Quick Selectivity Learning with Mixture Models. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1017–1033. https://doi.org/10.1145/3318464.3389727

[48] Suraj Shetiya, Saravanan Thirumuruganathan, Nick Koudas, and Gautam Das. 2020. Astrid: Accurate Selectivity Estimation for String Predicates using Deep Learning. *Proc. VLDB Endow.* 14, 4 (2020), 471–484. https://doi.org/10.14778/3436905.3436907

[49] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 99–113. https://doi.org/10.1145/3318464.3380584

[50] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 99–113. https://doi.org/10.1145/3318464.3380584

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

[52] Ji Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation for Similarity Queries. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 1745–1757. https://doi.org/10.1145/3448016.3452790

[53] Ji Sun, Jintao Zhang, Zhaoyan Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation: A Design Space Exploration and A Comparative Evaluation. *Proc. VLDB Endow.* 15, 1 (2021), 85–97. https://doi.org/10.14778/3485450.3485459

[54] Luming Sun. 2023. Papers for database systems powered by artificial intelligence (machine learning for database). *https://github.com/LumingSun/ML4DB-paper-list* (2023).

[55] Saravanan Thirumuruganathan, Suraj Shetiya, Nick Koudas, and Gautam Das. 2022. Prediction Intervals for Learned Cardinality Estimation: An Experimental Evaluation. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 3051–3064. https://doi.org/10.1109/ICDE53745.2022.00274

[56] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. SkinnerDB: Regret-Bounded Query Evaluation via Reinforcement Learning. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1153–1170. https://doi.org/10.1145/3299869.3300088

[57] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2011. Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions. *Proc. VLDB Endow.* 4, 11 (2011), 852–863. http://www.vldb.org/pvldb/vol4/p852-tzoumas.pdf

[58] Kostas Tzoumas, Timos Sellis, and Christian S Jensen. 2008. A reinforcement learning approach for adaptive query processing. *History* (2008).

[59] Fang Wang, Xiao Yan, Man Lung Yiu, Shuai LI, Zunyao Mao, and Bo Tang. 2023. Speeding Up End-to-end Query Execution via Learning-based Progressive Cardinality Estimation. *Proc. ACM Manag. Data* 1, 1 (2023), 28:1–28:25. https://doi.org/10.1145/3588708

[60] Jiayi Wang, Chengliang Chai, Jiabin Liu, and Guoliang Li. 2021. FACE: A Normalizing Flow based Cardinality Estimator. *Proc. VLDB Endow.* 15, 1 (2021), 72–84. https://doi.org/10.14778/3485450.3485458

[61] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proc. VLDB Endow.* 14, 9 (2021), 1640–1654. https://doi.org/10.14778/3461535.3461552

[62] Lianggui Weng, Rong Zhu, Di Wu, Bolin Ding, Bolong Zheng, and Jingren Zhou. 2024. Eraser: Eliminating Performance Regression on Learned Query Optimizer. *Proc. VLDB Endow.* (2024).

[63] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2009–2022. https://doi.org/10.1145/3448016.3452830

[64] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. FactorJoin: A New Cardinality Estimation Framework for Join Queries. *Proc. ACM Manag. Data* 1, 1 (2023), 41:1–41:27. https://doi.org/10.1145/3588721

[65] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *CoRR* abs/2012.14743 (2020). arXiv:2012.14743 https://arxiv.org/abs/2012.14743

[66] Ziniu Wu, Pei Yu, Peilun Yang, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2022. A Unified Transferable Model for ML-Enhanced DBMS. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org. https://www.cidrdb.org/cidr2022/papers/p6-wu.pdf

[67] Ziniu Wu, Rong Zhu, Andreas Pfadler, Yuxing Han, Jiangneng Li, Zhengping Qian, Kai Zeng, and Jingren Zhou. 2020. FSPN: A New Class of Probabilistic Graphical Model. *CoRR* abs/2011.09020 (2020). https://arxiv.org/abs/2011.09020

[68] Linglin Yang, Lei Yang, Yue Pang, and Lei Zou. 2022. gCBO: A Cost-based Optimizer for Graph Databases. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, Mohammad Al Hasan and Li Xiong (Eds.). ACM, 5054–5058. https://doi.org/10.1145/3511808.3557197

[69] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 931–944. https://doi.org/10.1145/3514221.3517885

[70] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proc. VLDB Endow.* 14, 1 (2020), 61–73. https://doi.org/10.14778/3421424.3421432

[71] 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. *Proc. VLDB Endow.* 13, 3 (2019), 279–292. https://doi.org/10.14778/3368289.3368294

[72] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. 2022. Cost-based or Learning-based? A Hybrid Query Optimizer for Query Plan Selection. *Proc. VLDB Endow.* 15, 13 (2022), 3924–3936. https://www.vldb.org/pvldb/vol15/p3924-li.pdf

[73] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 1297–1308. https://doi.org/10.1109/ICDE48307.2020.00116

[74] Jintao Zhang, Chao Zhang, Guoliang Li, and Chengliang Chai. 2023. AutoCE: An Accurate and Efficient Model Advisor for Learned Cardinality Estimation. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, California, USA, April 3-7, 2023*. IEEE.

[75] Kangfei Zhao, Jeffrey Xu Yu, Zongyan He, Rui Li, and Hao Zhang. 2022. Lightweight and Accurate Cardinality Estimation by Neural Network Gaussian Process. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 973–987. https://doi.org/10.1145/3514221.3526156

[76] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proc. VLDB Endow.* 15, 8 (2022), 1658–1670. https://www.vldb.org/pvldb/vol15/p1658-zhao.pdf

[77] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2022. Database Meets Artificial Intelligence: A Survey. *IEEE Trans. Knowl. Data Eng.* 34, 3 (2022), 1096–1116. https://doi.org/10.1109/TKDE.2020.2994641

[78] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query Performance Prediction for Concurrent Queries using Graph Embedding. *Proc. VLDB Endow.* 13, 9 (2020), 1416–1428. https://doi.org/10.14778/3397230.3397238

[79] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A Learning-to-Rank Query Optimizer. *Proc. VLDB Endow.* 16, 6 (2023), 1466–1479. https://www.vldb.org/pvldb/vol16/p1466-zhu.pdf

[80] Rong Zhu, Lianggui Weng, Wenqing Wei, Di Wu, Jiazhen Peng, Yifan Wang, Bolin Ding, Defu Lian, Bolong Zheng, and Jingren Zhou. 2024. PilotScope: Steering Databases with Machine Learning Drivers. *Proc. VLDB Endow.* (2024).

[81] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proc. VLDB Endow.* 14, 9 (2021), 1489–1502. https://doi.org/10.14778/3461535.3461539

[82] Rong Zhu, Tianjing Zeng, Andreas Pfadler, Wei Chen, Bolin Ding, and Jingren Zhou. 2021. Glue: Adaptively Merging Single Table Cardinality to Estimate Join Query Size. *CoRR* abs/2112.03458 (2021). https://arxiv.org/abs/2112.03458