# Learned Query Optimizer: At the Forefront of AI-Driven Databases

Rong Zhu[1], Ziniu Wu[2], Chengliang Chai[3], Andreas Pfadler[1], Bolin Ding[1], Guoliang Li[3], Jingren Zhou[1]

[1]*Alibaba Group, Hangzhou, China* [2]*MIT CSAIL, MA, USA* [3]*Tsinghua University, Beijing, China*
[1]{red.zr, andreaswernerrober, bolin.ding, jingren.zhou}@alibab-inc.com, [2]ziniuw@mit.edu,
[3]{ccl, liguoliang}@tsinghua.edu.cn

## ABSTRACT

Applying ML-based techniques to optimize traditional databases, or AI4DB, has becoming a hot research spot in recent. Learned techniques for query optimizer(QO) is the forefront in AI4DB. QO provides the most suitable experimental plots for utilizing ML techniques and learned QO has exhibited superiority with enough evidence. In this tutorial, we aim at providing a wide and deep review and analysis on learned QO, ranging from algorithm design, real-world applications and system deployment. For algorithm, we would introduce the advances for learning each individual component in QO, as well as the whole QO module. For system, we would analyze the challenges, as well as some attempts, for deploying ML-based QO into actual DBMS. Based on them, we summarize some design principles and point out several future directions. We hope this tutorial could inspire and guide researchers and engineers working on learned QO, as well as other context in AI4DB.

**Duration:** 1.5h

## 1 BACKGROUND, GOALS, AND OBJECTIVES

*Query optimizer*(QO), the core part of DBMS and big data processing platform, directly determines the query plan quality as well as the system performance. Although it has been extensively studied and refined, the intrinsic nature of varied data and query workload pose great challenges for QO. Until now, it is a consensus that the performance of QO is not fully favorable, especially on complex and/or tail cases [10].

Recently, the development of machine learning (ML), especially deep learning, exhibits great superiority in the data processing area. ML techniques enable automatic, fine-grained and more accurate characterization of the problem space, thus are potential to benefit a variety of tasks in DBMS. This cross-filed, called "AI4DB", has become a hot spot in the database research field. In AI4DB, learned QO serve as a *pioneer*. By some survey [28] and paper collection repository [1] on AI4DB, more than 100+ of the paper are published to optimize QO in the last decade.

The prosperity of learned QO arises as it provides suitable experimental plots for various kinds of ML techniques. The techniques applied in QO including supervised, unsupervised, reinforcement learning and etc. Meanwhile, QO is much easier to be hooked for evaluation than other hardcore parts of DBMS. Enough evidence has shown the benefits of applying learned QO techniques on improving the query performance [4, 17].

In a nutshell, learned QO is fast growing and occupies the pivotal position in AI4DB. It attracts considerable research efforts today and exhibits bright future. Therefore, it is necessary to summarize its advances, analyze its status and guide its development. Motivated by this, we organize this tutorial to fulfill the following goals and objectives:

**1) A comprehensive review on the advances of learned QO.** Recent work in the last decade has proposed numerous ML-based approaches for each individual component in QO, as well as the whole QO module. 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, to exhibit the advantages and disadvantages of each method. This would give the audience a deep and holistic understanding of learned QO techniques.

**2) A deep analysis on the challenges, as well as some attempts, on applying and deploying learned QO.** Using learned QO to benefit the real-world DBMS is the *ultimate goal*, but it is very difficult due to the inherently stochastic nature of ML. We try to summarize the challenges and requirements for the actual deployment of learned QO, from both algorithm and system perspectives. Meanwhile, we would introduce some valuable tentative work, which tries to apply learned QO in real-world scenarios or solve the practical deployment problem. This would give the audience a clear picture on the core difficulties of learned QO (as well as AI4DB).

**3) A summary on promising future work of learned QO.** Based on 1) and 2), we try to point out a series of important directions for learned QO, 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 *practical* work on learned QO, as well as AI4DB, and make this technology applicable as early as possible.

## 2 CONTENT AND ORGANIZATION

We organize the tutorial into five units in a logic order corresponding to the three goals. We outline the main structure as follows and elaborate the details in each subsection.

- **Part 1: Preliminary and Background (5 mins)**
  - 1.1 Basic concepts and knowledge of QO (3 mins)
  - 1.2 Status and outline of learned QO (2 mins)
- **Part 2: Learned Individual Components (35 minutes)**
  - 2.1 Learned cardinality estimation (15 mins)
    - 2.1.1 Problem definition and traditional methods
    - 2.1.2 Query-driven methods
    - 2.1.3 Data-driven methods
    - 2.1.4 Hybird methods
    - 2.1.5 Benchmark evaluation and analysis
  - 2.2 Learned cost model (10 mins)
    - 2.2.1 Experience-driven literature methods
    - 2.2.2 Single query CostEst

    – 2.2.3 Concurrent query CostEst
  ◦ 2.3 Learned join order search (7 mins)
    – 2.3.1 Problem formulation and existing methods
    – 2.3.2 Offline learning methods
    – 2.3.3 Online learning methods
  ◦ 2.4 Summary (3 mins)

- **Part 3: Learned Whole QO Module (20 minutes)**:
  ◦ 3.1 End-to-end learned QO: NEO [11] (5 mins)
  ◦ 3.2 Learn to steer QO: BAO [10] (5 mins)
  ◦ 3.3 One model for all: MTMLF [24] (5 mins)
  ◦ 3.4 Comparison and analysis (5 mins)
- **Part 4: Applications and deployment (25 minutes)**:
  ◦ 4.1 Application case studies of learned QO (10 mins)
    – 4.1.1 Learned cost model for SCOPE [15]
    – 4.1.2 Learn to steer SCOPE [12]
  ◦ 4.2 Challenges of actual deployment (5 mins)
  ◦ 4.3 Start-up system for deployment: Baihe [14] (10 mins)
- **Part 5: Summary and Future Works (5 minutes)**:
  ◦ 5.1 Summary (2 mins)
  ◦ 5.2 Future work (3 mins)

## 2.1 Preliminary and Background

QO plays a significant role in DBMS. In this tutorial, we focus on the seminar structure of QO, such as PostgreSQL and Calcite, follows the volcano framework [3] shown in Figure 1. On a high level, the QO module is composed of three components, namely *cardinality estimation* (CardEst), *cost estimation* (CostEst) and *join order search* (JoinSel). It provides ample room for using ML techniques. These work covers different aspects: from ML models to benchmark evaluation; from system design to applications; from individually learned component to the whole QO module; from unsupervised/supervised models to reinforcement learning policies; and from statistical models to deep models. We try to carefully organize them and provide some in-depth insights.

## 2.2 Learned Individual Components

In this part, we first introduce the ML-based advances in each component(CardEst, CostEst and JoinSel), and then analyze their relations to give some future work suggestions.

*2.2.1* ***Learned Cardinality Estimation***. Given the data $D$ and query $Q$, CardEst methods build a sketch-based synopses using their information to estimate the number of tuples in $D$ satisfying $Q$. Traditional CardEst methods mainly utilize one or multi-dimensional histogram or various sampling techniques. ML-based CardEst explore more technical routines, which could be categorized into the three main classes as follows:

1) *Query-driven methods* learn supervised models directly mapping featurized query to its cardinality. The representative work includes: a) MSCN [7] built upon the multi-set deep convolutional network; b) LW-XGB and LW-NN [2] using gradient boosted trees and neural networks with the lightweight regression model; and c) Fauce [9] using ensembles of deep neural networks.

2) *Data-driven methods* learn unsupervised models of the data distribution, then the probability (cardinality) of any query could be computed. The main modeling tools and the corresponding CardEst methods include: a) NeuroCard [25] using deep auto-regression; b) BayesNet [19] and BayesCard [23] using traditional or revitalized Bayesian networks; c) DeepDB [6] using sum-product networks; and d) FLAT [30] using FSPNs.

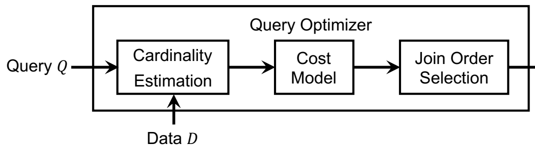

**Figure 1: Architecture of Query Optimizer.**

3) *Hybird methods* utilize statistical models as well as data and/or query distributions: a) [22] learns a statistical estimator of cardinality from samples drawn from synthetic data and workload; b) Flowloss [13] learns cardinality considering the importance of each sub-query; and c) GLUE [31] proposes a general framework merging single table estimation results produced by any method to predict join query size.

Besides these CardEst methods, some benchmarks [4, 17, 21] are proposed for comprehensively evaluation. By summarizing their results, we could clearly understand the pros and cons of each ML-based CardEst method and when (and how much) it could improve the QO performance.

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

1) *Single query CostEst.* In fact, we can directly leverage learned cardinality for cost estimation, but this leads to accumutative errors. [16] proposed an end-to-end method that uses TreeLSTM to encode the queries and learns from previous examples, so as to directly output the cost.

2) *Concurrent queries CostEst.* CostEst over concurrent queries is non-trivial because it is rather hard to capture the correlations between different queries. GPredictor [29] utilizes the graph neural network to capture the query relationships and estimate the query perofrmance accurately. Prestroid [27] leverages the tree convolution based approach to estimate the concurrent query performance in a cloud environment.

*2.2.3* ***Learned Join Order Search***. The JoinSel enumerates all candidate plans in the 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:

1) *Offline learning methods* learn from the previous queries to improve the performance of future ones. DQ [5] and ReJoin [8] 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 [26] proposes a model that utilizes the TreeLSTM to represent the join state, so as to better capture join tree structure.

2) *Online learning methods* learn a join order through adaptive query processing, which can change the order even during the execution of queries. Eddy-RL [20] models the query execution as a reinforcement learning problem and automatically learn how to adjust the join order during execution using Q-learning. SkinnerDB [18] 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.

*2.2.4* ***Summary***. Finally, we analyze the roles and importance. Based on this, we give some future research suggestions on: 1) how to design practical ML-based methods for each component; and 2) how to coordinate them together to optimize the end-to-end performance.

## 2.3 Learning the Whole QO Module

In this part, we first introduce three representative work following different technique routines to optimize the whole QO module, and then make a comparison and analysis.

*2.3.1* ***End-to-end learned QO: NEO [11].*** It replace each component in the traditional QO to be the first end-to-end learned system. Specifically, a value network is learned, upon featurized queries and data statistical, to predict the execution time of any query plan, which replaces CardEst and CostEst. A DNN-guided learned best-first search strategy is utilized to replace the dynamic programming based plan enumeration in JoinSel. Meanwhile, NEO use RL to evolve its experience during execution.

*2.3.2* ***Learn to steer QO: BAO [10].*** It regards the original QO as a black-box but learn to provide per-query optimization hints. Specifically, for each query and each possible hint set (i.e., disable or enable some join/scan operations), the original QO would generate a candidate plan. BAO learns a predictive model on the execution time of each plan, and returns the plan with minimum predicted latency. By this way, BAO make minimal changes of the existing QO and is more flexible to data, schema and query workload updates.

*2.3.3* ***One model for all: MTMLF [24].*** This is more of a vision work considering how ML could help for the multiple tasks in QO. It provides a new perspective on classifying knowledge that ML models trying to comprehend. A pre-trained model is learned 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., CardEst, CostEst and JoinSel, respectively.

*2.3.4* ***Comparison and analysis.*** First, we conceptually compare the architecture difference of the three work. Based on this, we analyze their success and shortcomings in terms of different aspects: training cost, integration hardness, transferability, generality, update speed and etc. Later, we verify our claims using benchmark experimental results. Finally, we summarize some key insights on the design choices of learned QO systems and provide some valuable suggestions for the future work.

## 2.4 Applications and Deployment

In this part, we move from designing to applying and deploying learned QO in real-world scenarios. We first introduce some specific case studying of applying learned QO in the industry production environments. Then, we stand on the more general view to analyze the challenges of deploying learned QO in actual DBMS. After that, we introduce a start-up work trying to mitigate this gap with a demonstration.

*2.4.1* ***Application case studies of learned QO.*** We introduce two works on applying learned QO in the Microsoft SCOPE:

1) [15] tries to learn accurate cost models for big data systems and integrate them within the original QO. 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 QO of SCOPE and exhibit the superiority.

2) [12] applies BAO's idea to steer SCOPE's QO. 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. Based on this, an in-depth evaluation is done on petabyte-scale big data and 150k daily jobs.

*2.4.2* ***Challenges of actual deployment.*** Most prior research has been justified on numerical experiments *outside* of real systems, there has been a striking absence of work focusing on the practical deployment of learned QO *inside* real world systems. In this part we would discuss challenges, as well as concrete requirements and design patterns, for learned QO deployment.

First, the inherently stochastic nature of ML appears to be at odds with the rock-solid, stable and deterministic requirements of DB systems. ML models may produce results with unpredictable error bounds and their generalization ability may fail catastrophically on different data.

Second, failures of ML models may not be readily detectable, since they might only be visible through a change in some numerical value rather than a concrete and meaningful error message.

Third, ML model training is very specific. It requires close supervision by experts to tune optimizers, hyper-parameters and termination criteria. Hence, it is very difficult to integrate automated training procedures into a DB system as maintaining simple rules and statistical information.

*2.4.3* ***A start-up system for deployment: Baihe [14].*** We develop the Baihe, a general framework developed for integrating ML models into a relational DB systems. Its fundamental architecture provides a blueprint for deploying learned QO, as well as other modules in AI4DB. Baihe will be made open source before the start of the tutorial.

Baihe is developed upon PostgreSQL, a widely used and extensible DBMS. We would present a quick overview of Baihe's design principle, a detailed description on its architecture and them perform a live demo showing how Baihe can be used to quickly develop and deploy a query latency prediction module for PostgreSQL. Afterwards, we will briefly discuss how Baihe can be used in other contexts related to learned QO.

## 2.5 Summary and Future Work

We summarize the status, including both advances and problems for learned QO. We target at putting forward the learned QO technology applicable as early as possible. Thus, we outline several future directions for this. First is on the design principles and evaluation metrics for each learned component and better models considering the role of each in the QO module. Second is on how to consider QO as a whole upon which the possible new routines for learned QO can be designed. Third is about the deployment of such AI-aided components in real database systems as well as algorithm-system co-design. Forth is on how to transfer the learned QO techniques to other fields in AI4DB.

## 3 TUTORIAL INFORMATION

**Intended Audience.** We target to attract audience from three fields:

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

2) ML researchers who are interested in finding ML applications in system-related fields. This tutorial would educate them the requirements of the DB scenarios and guide them to design and optimize ML models.

3) System engineers who are now using ML techniques in real-world industry data processing systems. This tutorial would

help them to know about the current advances of learned QO and inspire them to resolve the practical deployment issues.

**Difference with Previous Tutorials.** Our tutorial has sightly overlaps with two of our tutorials before:

1) *AI Meets Database: AI4DB and DB4AI* in SIGMOD 2021. This tutorial provides a very board review on AI4DB and DB4AI topics. Learned QO is only a very small part (around 5mins) in this tutorial, covering the main ideas in some Card/CostEst and JoinSel methods. However, our EDBT tutorial aims at providing a comprehensive and deep review and analysis on learned QO.

2) *AutoML: From Methodology to Application* in CIKM 2021. This tutorial focuses on AutoML techniques, including hyper-parameter tuning, NAS, and meta-learning. ML-based CardEst is discussed as an important application. The main topics are totally different with our EDBT tutorial.

## 4 BIOGRAPHY

**Rong Zhu** is a research scientist in the Data Analytics and Intelligence Lab, Alibaba Damo Academy. He obtained his PhD degree in Department of Computer Science and Technology, Harbin Insitute of Technology in 2019. His research interests lie in AI4DB, graph data mining and graph processing system.

**Ziniu Wu** is a PhD candidate in CSAIL, MIT. He holds a Master degree in Computer Science from Oxford University in 2020. His research focuses on databases and ML for systems.

**Chengliang Chai** is a postdoc in the Department of Computer Science, Tsinghua University. He received his PhD degree in Computer Science from Tsinghua University in 2020. His research interests lie in crowdsourcing data management, data preparation and AI & DB co-optimization.

**Andreas Pfadler** is a research scientist in the Data Analytics and Intelligence Lab, Alibaba Damo Academy. He obtained his PhD degree in Mathematics from TU Berlin in 2011. His research focuses on ML for systems and data management. He has more than ten years of experience as a quantitative developer, ML engineer and project manager.

**Bolin Ding** is a research scientist in in the Data Analytics and Intelligence Lab, Alibaba Damo Academy. 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.

**Guoliang Li** is a professor in the Department of Computer Science, Tsinghua University. He received his PhD degree in Computer Science from Tsinghua University in 2009. His research interests mainly include database system, data cleaning and integration, crowdsourcing, and AI & DB co-optimization.

**Jingren Zhou** is Senior Vice President at Alibaba and Deputy Director of Alibaba DAMO Academy. 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 & 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] MIT Data Systems Group. 2021. *http://dsg.csail.mit.edu/mlforsystems/papers/* (2021).

[2] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity estimation for range predicates using lightweight models. *PVLDB* 12, 9 (2019), 1044–1057.

[3] Goetz Graefe. 1995. The cascades framework for query optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.

[4] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Tan Wei Liang, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2022. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *VLDB* (2022).

[5] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep q-learning from demonstrations. In *AAAI*.

[6] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2019. DeepDB: learn from data, not from queries!. In *PVLDB*.

[7] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*.

[8] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to optimize join queries with deep reinforcement learning. *arXiv:1808.03196* (2018).

[9] Jie Liu, Wenqian Dong, Qingqing Zhou, and Dong Li. 2021. Fauce: fast and accurate deep ensembles with uncertainty for cardinality estimation. *PVLDB* 14, 11 (2021), 1950–1963.

[10] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making learned query optimization practical. In *SIGMOD*. 1275–1288.

[11] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *PVLDB* 12, 11 (2019).

[12] 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*. 2557–2569.

[13] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2022. Flow-Loss: Learning Cardinality Estimates That Matter. *PVLDB* (2022).

[14] Andreas Pfadler, Rong Zhu, Wei Chen, Botong Huang, Tianjing Zeng, Bolin Ding, and Jingren Zhou. 2021. Baihe: SysML Framework for AI-driven Databases. *arXiv:2112.14460* (2021).

[15] 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 *SIGMOD*. 99–113.

[16] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *PVLDB* 13, 3 (2019).

[17] Ji Sun, Jintao Zhang, Zhaoyan Sun, Guoliang Li, and Nan Tang. 2022. Learned cardinality estimation: A design space exploration and a comparative evaluation. VLDB.

[18] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In *SIGMOD*. 1153–1170.

[19] Kostas Tzoumas, Amol Deshpande, and Christian S Jensen. 2011. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB* 4, 11 (2011), 852–863.

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

[21] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *VLDB* 14, 9 (2021), 1640–1654.

[22] Renzhi Wu Wu, Bolin Ding, Xu Chu, Zhewei Wei, Xiening Dai, Tao Guan, and Jingren Zhou. 2022. Learning to be a Statistician: Learned Estimator for Number of Distinct Values. In *PVLDB*.

[23] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *arXiv preprint arXiv:2012.14743* (2020).

[24] Ziniu Wu, Peilun Yang, Pei Yu, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2021. A Unified Transferable Model for ML-Enhanced DBMS. *CIDR* (2021).

[25] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2021. NeuroCard: One Cardinality Estimator for All Tables. *PVLDB* 14, 1 (2021), 61–73.

[26] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement learning with tree-lstm for join order selection. In *ICDE*. 1297–1308.

[27] Johan Kok Zhi Kang, 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*. 1014–1022.

[28] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database Meets Artificial Intelligence: A Survey. *TKDE* (2020).

[29] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query performance prediction for concurrent queries using graph embedding. *PVLDB* 13, 9 (2020), 1416–1428.

[30] 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. *VLDB* 14, 9 (2021), 1489–1502.

[31] 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. *arXiv:2112.03458* (2021).