Join Order Selection with Deep Reinforcement Learning: Fundamentals, Techniques, and Challenges

Zhengtong Yan
University of Helsinki
Helsinki, Finland
zhengtong.yan@helsinki.fi

Valter Uotila
University of Helsinki
Helsinki, Finland
valter.uotila@helsinki.fi

Jiaheng Lu
University of Helsinki
Helsinki, Finland
jiaheng.lu@helsinki.fi

ABSTRACT

Join Order Selection (JOS) is a fundamental challenge in query optimization, as it significantly affects query performance. However, finding an optimal join order is an NP-hard problem due to the exponentially large search space. Despite the decades-long effort, traditional methods still suffer from limitations. Deep Reinforcement Learning (DRL) approaches have recently gained growing interest and shown superior performance over traditional methods. These DRL-based methods could leverage prior experience through the trial-and-error strategy to automatically explore the optimal join order. This tutorial will focus on recent DRL-based approaches for join order selection by providing a comprehensive overview of the various approaches. We will start by briefly introducing the core concepts of join ordering and the traditional methods for JOS. Next, we will provide some preliminary knowledge about DRL and then delve into DRL-based join order selection approaches by offering detailed information on those methods, analyzing their relationships, and summarizing their weaknesses and strengths. To help the audience gain a deeper understanding of DRL approaches for JOS, we will present two open-source demonstrations and compare their differences. Finally, we will identify research challenges and open problems to provide insights into future research directions. This tutorial will provide valuable guidance for developing more practical DRL approaches for JOS.

PVLDB Reference Format:
Zhengtong Yan, Valter Uotila, and Jiaheng Lu. Join Order Selection with Deep Reinforcement Learning: Fundamentals, Techniques, and Challenges. PVLDB, 16(12): 3882 - 3885, 2023.
doi:10.14778/3611540.3611576

1 BACKGROUND AND MOTIVATION

Joining multiple tables is a common but expensive operation in relational databases and modern data warehouses. The query performance of the same query with different join orders could vary by orders of magnitude [9, 10]. However, finding the optimal join order with the lowest cost is an NP-hard problem because the valid join orders grow exponentially with the number of tables to be joined. In the past four decades, a large number of approaches have been proposed to solve this problem, such as dynamic programming, heuristic search, adaptive optimization, etc. These traditional methods typically search the solution space of all possible join orders with some pruning techniques based on cardinality and cost estimations. Despite the decades-long effort, traditional methods still suffer from limitations especially when handling complicated and large joins. Traditional approaches usually employ static join order enumeration without feedback about the quality of the query plans. Hence, they often repeatedly choose the same bad plan, as they have no mechanism for "learning from the experience".

To address the shortcomings of conventional solutions, a number of methods based on DRL have been developed. Those methods, which include `ReJOIN` [16], `DQ` [8], `SkinnerDB` [22, 23], `RTOS` [32], `AlphaJoin` [33], and `JOGGER` [2], have shown superior performance over traditional methods. Join order selection is modeled as a Markov Decision Process and solved by deep reinforcement learning methods like Deep Q-Network (DQN) [2, 8, 32] and Proximal Policy Optimization (PPO) [16]. DRL-based approaches could be categorized in various dimensions, such as *offline* and *online* learning (depending on whether queries and runtime metrics need to be collected before learning), *value-based* and *policy-based* (depending on learning a value function or a policy for solving the JOS problem). Compared to conventional methods, these learning-based methods could leverage prior experience through trial-and-error strategies to provide better query plans within less optimization time. In addition to DRL-based join order selection, several end-to-end learned optimizers (e.g., `Neo` [15], `Bao` [14], `Balsa` [30], `HybridQO` [31], `Lero` [34], `LOGGER` [3], and `COOOL` [27]) have been proposed to provide more functionalities beyond join order selection, such as join operator selection, index selection, access method selection, and query plan generation.

In this tutorial, we will focus on recent DRL-based approaches for join order selection. We will provide a comprehensive overview and detailed introduction to this topic, from the basic concepts, state-of-the-art methods, and open-source implementations, to open challenges and future directions. Table 1 summarizes the main research works we will present in this tutorial. Since we will concentrate on the join order selection problem rather than the whole query optimizer, we will only introduce the JOS component of those learned optimizers. Figure 1 shows the relationships among those works. Specifically, we will first introduce the fundamentals and critical concepts of join order selection, such as query graphs, join trees, join forests, cost models, and the NP-hardness of JOS. We will then provide a comprehensive overview and detailed introduction of the DRL-based methods with different categories. Next, we will present two demonstrations with value-based and policy-based implementations. Finally, we will discuss the open problems and future directions of DRL-based methods for join order selection, especially focusing on the difficulties in practical applications [6].

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611576

Table 1: A summary of the DRL-based methods for join order selection.

Reference	Year	Representation Learning		DRL Framework			
		Features [1]	Encodings	State [2]	Action	Reward	Algorithms
ReJOIN [16]	2018	• Join trees • Join predicates • Selection predicates	Row vector with relation height in the join tree	Join tree	Join two subtrees	Cost	PPO
DQ [8]	2018	• Query graph • Selections • Physical operators	One-Hot	Query graph	Merge two vertices	Cost & Latency	Deep Q-Learning
SkinnerDB [22, 23]	2019	n/a	n/a	Permutations of tables	Choose tables	• Number of join result tuples • Number of Cartesian product tuples	UCT algorithm
AlphaJoin [33]	2020	• SQL queries • Query plans	One-Hot	Join order	Join	Execution time	MCTS
RTOS [32]	2020	• Join forest • Queries • Table and columns	• One-Hot • Tree-LSTM	Join forest	Join two subtrees	Cost & Latency	• DQN • Multi-task learning
JOGGER [2]	2022	• Query graph • Join forest • Schema graph • Columns	• GCN • DeepWalk • Attention model	Join forest	Join two subtrees	Cost	• DQN • Curriculum learning
Neo [15]	2019	• Queries • Query plans	• One-Hot • Tree-CNN • Row vector embedding	Partial plan	Steps for building a query plan	Latency	• Value network • Learning from demonstration
Bao [14]	2021	Query plans	• One-Hot • Tree-CNN	Constant state	Select a hint	Execution time	• CMABs • Thompson sampling
Balsa [30]	2022	• Queries • Query plans	• One-Hot • Tree-CNN	Partial plan	Add operators to the partial plan	Latency	• On-policy learning • Safe learning
HybridQO [31]	2022	• Queries • Join order	• One-Hot • Tree-LSTM	n/a	n/a	n/a	MCTS
Lero [34]	2023	• Queries • Query plans	• One-Hot • Tree-CNN	n/a	n/a	n/a	n/a
LOGGER [3]	2023	• Join forest • Queries • Table and columns	• Tree-LSTM • Graph Transformer	Query plan	Select the next table and join algorithm	Latency	State & Action Network
COOOL [27]	2023	Query plans	• One-Hot • Tree-CNN	n/a	n/a	n/a	n/a

[1] *Features* are representations of the information about queries and databases, such as join trees (i.e., join orders), query graphs (i.e., the undirected graph based on join tables and predicates), join predicates (i.e., the attributes for join), and database schema (e.g., tables, columns, and the primary-foreign key relationships among tables).
[2] *States* are used to indicate the stages of the learning process. Join order selection is typically an *episodic task*, thus reaching the terminal state (i.e., all tables are joined in the query plan) means the end of a learning *episode*.

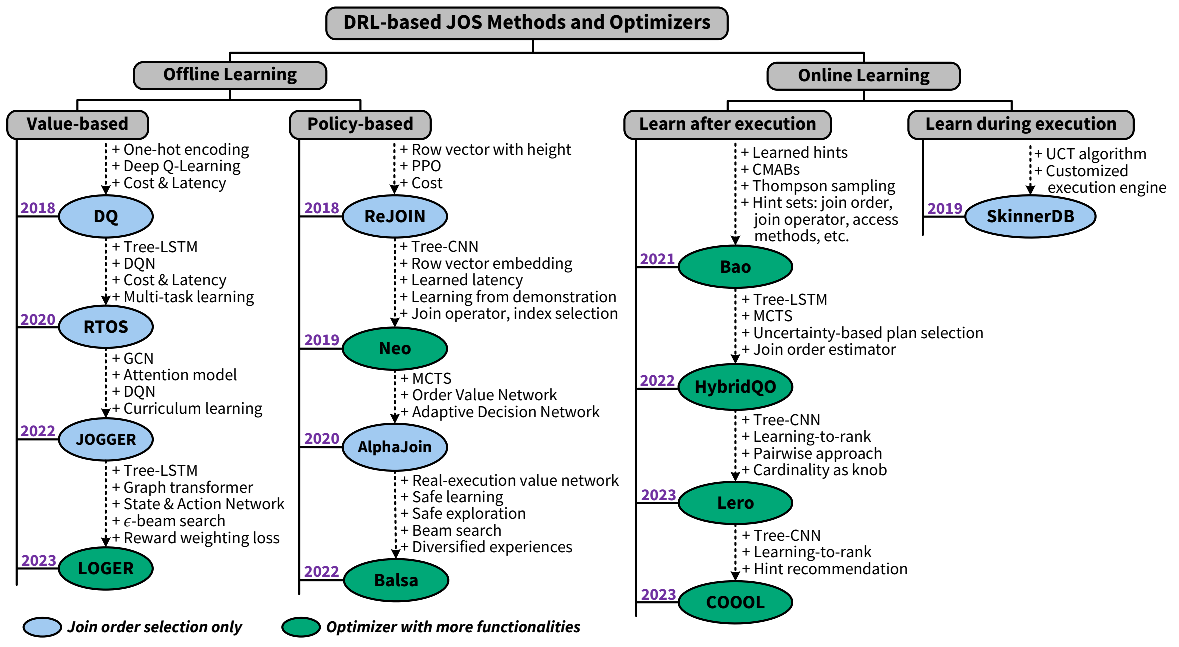

Figure 1: Taxonomy and relationships among the DRL-based methods for JOS and query optimizer.

2 TUTORIAL OUTLINE

We plan to deliver a lecture-style tutorial within 1.5 hours:

Part I: Fundamentals of JOS (15 min)

1.1 Basic concepts and knowledge of join ordering (10 min)

1.2 Traditional approaches and their limitations (5 min)

Part II: DRL-Based Methods for JOS (60 min)

2.1 Basics of deep reinforcement learning (15 min)

2.2 DRL-based methods for JOS (45 min)

Part III: Two Open-Source Demos (10 min)

3.1 ReJOIN[1]: a policy-based method (4 min)

3.2 RTOS[2]: a value-based method (4 min)

3.3 Comparison of ReJOIN and RTOS (2 min)

Part IV: Open Challenges and Future Directions (5 min)

[1]https://github.com/antonismand/ReJOIN
[2]https://github.com/TsinghuaDatabaseGroup/AI4DBCode/tree/master/RTOS

2.1 Fundamentals of Join Ordering

In this part, we will first introduce some basic and essential concepts for join ordering [17]. Those concepts include query graphs (e.g., *chain, star, cyclic*, and *clique graph*), join trees (including *left-deep, right-deep, zig-zag*, and *bushy trees*), join forest [32], cost functions (e.g., C_{out} [4] and C_{mm} [9]), and the search space of join ordering. We will use some join query examples from the JOB benchmark [9, 10] to help the audiences understand those concepts.

Then, we will give a brief overview of traditional approaches for JOS by comparing and analyzing their shortcomings with learning-based methods, especially when dealing with large and complex join queries. Traditional methods can be categorized into heuristic methods (e.g., IKKBZ [7], GOO [5], GEQO [1], QuickPick [26]), dynamic programming methods (e.g., DPsize [21], DPsub [25], DPccp [18], and DPhyp [19]), adaptive methods (e.g., LinDP [20]), and massively parallel methods (e.g., MPDP [13]). Some of those methods have been widely used in open-source and commercial databases.

2.2 DRL-Based Methods for JOS

In this part, we will first briefly review DRL, including the general framework of RL, the trade-off of exploration and exploitation, state and action value functions, value-based and policy-based methods, etc. Specifically, we will review some widely used algorithms in previous works, such as Deep Q-Network (DQN), Proximal Policy Optimization (PPO), Contextual Multi-Armed Bandits (CMABs), Monte Carlo Tree Search (MCTS), and UCB applied to Trees (UCT).

Next, we will discuss how to apply DRL algorithms to solve the JOS problem. We will introduce DRL-based methods and their relationships (see Table 1 and Figure 1) as well as their weaknesses and strengths. Based on whether we need to collect queries and runtime metrics before the learning process, these methods could be classified into *offline* and *online* learning.

Offline learning. ReJOIN [16] and DQ [8] are the first policy-based and value-based methods for JOS, respectively. Despite their differences in the training data format and learning process, they both adopt simple encodings to represent the join trees. RTOS [32] further enhance the performance of ReJOIN and DQ in two aspects: 1) using Tree-LSTM to capture the structural information of join trees, and 2) employing multi-task learning for joint cost and latency as the reward in different training stages. AlphaJoin [33] utlizes MCTS to replace the random search strategies in ReJOIN, DQ, and RTOS. MCTS can simulate many possible join orders in one tree structure and select join orders with the highest estimated performance. This addresses the limitation of random search, which is prone to miss high-quality join orders. JOGGER [2] is the latest work that aims to address the inefficiency issues of DRL methods. The idea is to reduce the number of parameters in Deep Neural Networks (DNNs) through graph-based models, including a Graph Convolutional Network (GCN) and a tailored-tree-based attention module. This strategy enables JOGGER to achieve state-of-the-art performance with less optimization time and computational resources.

Online learning. SkinnerDB [22, 23] is an online approach that stands out from previous works in that it does not require collecting training data beforehand. Instead, it can collect training data directly online and produce optimal join orders quickly during the execution of the current query. This online learning approach can provide a faster and more adaptive way to optimize query performance. However, SkinnerDB requires customized execution engines, which limits its practicality in general database systems.

Learned optimizer. In addition to individual learned components for JOS, several *end-to-end* learned optimizers have been introduced with more ingredients for join operator selection, index selection, plan generation, and plan search. Neo [15] was the first end-to-end learned optimizer that could directly predict query execution time using a value network (a Tree-CNN model), without relying on cost models or explicit cardinality estimates. Bao [14], inherited from Neo, also uses the Tree-CNN to estimate the runtime of queries. However, Bao only focuses on recommending query hints rather than rebuilding the entire optimizer with learning models. Specifically, Bao utilizes CMABs to model the problem in which each hint set (e.g., `SET enable_nestloop TO off`) is treated as an arm. Due to the simple and efficient architecture, Bao is easy to train and integrate into existing databases without extra modifications. Similar to Neo, Balsa [30] uses the Tree-CNN model and a simple (best-first) beam search strategy to find the best plan. The difference is that Balsa does not rely on expert demonstrations, which are required by Neo. HybridQO [31] is the first hybrid optimizer that combines traditional cost-based and modern learning-based techniques. It utilizes the same representation model as RTOS (i.e. Tree-LSTM) to encode the query structure. Similar to Bao, it uses hints to generate high-quality candidate plans. The candidate plan space is then explored by MCTS, which has better performance than DQN. Moreover, HybridQO uses an uncertainty-based method to avoid selecting bad plans. Lero [34] is a *learning-to-rank* optimizer that selects query plans based on their relative order rather than the cost or latency. Lero adopts a pairwise approach to compare two plans and choose the better one. LOGGER [3] is a learned query optimizer that aims at producing both high-quality join orders and operators. It leverages a Graph Transformer to encode the relationships between tables and predicates and exploits ϵ-beam search to explore the plan space. COOOL [27] shares similar principles with Bao to recommend learned query hints. However, COOOL incorporates the *learning-to-rank* mechanism to compare query plans, while Bao relies on regression models to estimate query latency.

2.3 Open Challenges and Future Directions

While some progress has been made in DRL-based methods for join order selection, there are still open problems and challenges, especially the difficulties in practical and real-world applications [6]. Some possible challenges and directions for future research include:

1) How to achieve a better trade-off of *exploration* (trying unexplored join orders) and *exploitation* (taking the most advantage of the explored join orders) in the learning process?

2) How to speed up the slow learning and time-consuming training process, especially for joins with a large number of tables?

3) How to improve the generalization performance to handle various changes, including query changes (e.g., join graphs, join predicates, and join table numbers) and schema changes?

4) How to provide an error bound to guarantee that the join orders produced do not have extremely poor quality?

5) How to improve the performance of join order selection for complex queries (e.g., nested and correlated subqueries)?

3 DIFFERENCES WITH PREVIOUS ONES

This tutorial has not been presented at other conferences. While there are some related tutorials, they cover broader research topics such as machine learning meets databases [11, 12, 28, 29] (including AI4DB and autonomous databases) and learned optimizers [24, 35] with topics like learned cardinality and cost estimations. In contrast, this tutorial has a strong focus on reinforcement learning for join order selection. In addition, we include recent advancements that utilize graph neural networks and attention models for encoding query structures. We also provide two demos to help the audience to understand the differences between value-based and policy-based methods. To the best of our knowledge, this is the first tutorial that discusses deep reinforcement learning-based join order selection.

4 TARGET AUDIENCES AND GOALS

Intended Audience. This tutorial is intended for a broad scope of researchers and practitioners from database and machine learning fields, with a focus on deep reinforcement learning for join order optimization. Those interested in query optimization and learned optimizers will also gain useful knowledge from this tutorial to guide their future research. Basic knowledge in query optimization and machine learning is sufficient to follow this tutorial. Some background in deep reinforcement learning would be quite helpful.

Learning Goals. The main goal of this tutorial is to provide a deep introduction to recent works on DRL-based methods for JOS. The learning goals include: (1) understanding the fundamental concepts of join ordering, (2) knowing the traditional methods and their limitations, (3) knowing the comprehensive picture of DRL-based methods for JOS and recognizing their relationships and differences, (4) identifying some open challenges and future directions.

5 SHORT BIOGRAPHIES

Zhengtong Yan is a Ph.D. student at the University of Helsinki, Finland. His research topics lie in reinforcement learning and quantum computing for databases.

Valter Uotila is a Ph.D. student at the University of Helsinki, Finland. His research is at the intersection of databases, quantum computing, and category theory.

Jiaheng Lu is a Professor at the University of Helsinki, Finland. His current research interests focus on multi-model databases and quantum computing for databases. He has written four books on Hadoop and NoSQL databases, and more than 130 papers published in SIGMOD, ICDE, VLDB, TODS, etc.

REFERENCES

[1] Swati V Chande and Madhavi Sinha. 2011. Genetic Optimization for the Join Ordering Problem of Database Queries. In *INDICON*. IEEE, 1–5.

[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 *SIGKDD*. ACM, 97–107.

[3] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. 2023. LOGER: A Learned Optimizer towards Generating Efficient and Robust Query Execution Plans. *PVLDB* 16, 7 (2023), 1777–1789.

[4] Sophie Cluet and Guido Moerkotte. 1995. On the Complexity of Generating Optimal Left-Deep Processing Trees with Cross Products. In *ICDT*. Springer, 54–67.

[5] Leonidas Fegaras. 1998. A New Heuristic for Optimizing Large Queries. In *DEXA*. Springer, 726–735.

[6] Runsheng Benson Guo and Khuzaima Daudjee. 2020. Research Challenges in Deep Reinforcement Learning-based Join Query Optimization. In *aiDM*. 1–6.

[7] Toshihide Ibaraki and Tiko Kameda. 1984. On the Optimal Nesting Order for Computing N-Relational Joins. *TODS* 9, 3 (1984), 482–502.

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

[9] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *PVLDB* 9, 3 (2015), 204–215.

[10] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query Optimization Through the Looking Glass, and What We Found Running the Join Order Benchmark. *The VLDB Journal* 27, 5 (2018), 643–668.

[11] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. AI Meets Database: AI4DB and DB4AI. In *SIGMOD*. ACM, 2859–2866.

[12] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. Machine Learning for Databases. *PVLDB* 14, 12 (2021), 3190–3193.

[13] Riccardo Mancini, Srinivas Karthik, Bikash Chandra, Vasilis Mageirakos, and Anastasia Ailamaki. 2022. Efficient Massively Parallel Join Optimization for Large Queries. In *SIGMOD*. ACM, 122–135.

[14] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD*. ACM, 1275–1288.

[15] 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), 1705–1718.

[16] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *aiDM*. 1–4.

[17] Guido Moerkotte. 2023. *Building Query Compilers.* https://pi3.informatik.uni-mannheim.de/~moer/querycompiler.pdf

[18] Guido Moerkotte and Thomas Neumann. 2006. Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. In *VLDB*. 930–941.

[19] Guido Moerkotte and Thomas Neumann. 2008. Dynamic Programming Strikes Back. In *SIGMOD*. ACM, 539–552.

[20] Thomas Neumann and Bernhard Radke. 2018. Adaptive Optimization of Very Large Join Queries. In *SIGMOD*. ACM, 677–692.

[21] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD*. 23–34.

[22] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. SkinnerDB: Regret-Bounded Query Evaluation via Reinforcement Learning. In *SIGMOD*. ACM, 1153–1170.

[23] Immanuel Trummer, Junxiong Wang, Ziyun Wei, Deepak Maram, Samuel Moseley, Saehan Jo, Joseph Antonakakis, and Ankush Rayabhari. 2021. SkinnerDB: Regret-bounded Query Evaluation via Reinforcement Learning. *TODS* 46, 3 (2021), 1–45.

[24] Dimitris Tsesmelis and Alkis Simitsis. 2022. Database Optimizers in the Era of Learning. In *ICDE*. IEEE, 3213–3216.

[25] Bennet Vance. 1998. *Join-order optimization with Cartesian products.* Ph.D. Dissertation. Oregon Graduate Institute of Science and Technology.

[26] Florian Waas and Arjan Pellenkoft. 2000. Join Order Selection—Good Enough is Easy. In *BNCOD*. Springer, 51–67.

[27] Xianghong Xu, Zhibing Zhao, Tieying Zhang, Rong Kang, Luming Sun, and Jianjun Chen. 2023. COOOL: A Learning-To-Rank Approach for SQL Hint Recommendations. *arXiv preprint arXiv:2304.04407* (2023).

[28] Zhengtong Yan, Jiaheng Lu, Naresh Chainani, and Chunbin Lin. 2021. Workload-Aware Performance Tuning for Autonomous DBMSs. In *ICDE*. IEEE, 2365–2368.

[29] Zhengtong Yan, Jiaheng Lu, Qingsong Guo, Gongsheng Yuan, Calvin Sun, and Steven Yang. 2022. Make Wise Decisions for Your DBMSs: Workload Forecasting and Performance Prediction Before Execution. In *DASFAA*. Springer.

[30] 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*. ACM, 931–944.

[31] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. 2022. Cost-based or Learning-based? A Hybrid Query Optimizer for Query Plan Selection. *PVLDB* 15, 13 (2022), 3924–3936.

[32] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *ICDE*. IEEE, 1297–1308.

[33] Ji Zhang. 2020. AlphaJoin: Join Order Selection à la AlphaGo. In *PhD@ VLDB*.

[34] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A Learning-to-Rank Query Optimizer. *PVLDB* 16, 6 (2023), 1466–1479.

[35] Rong Zhu, Ziniu Wu, Chengliang Chai, Andreas Pfadler, Bolin Ding, Guoliang Li, and Jingren Zhou. 2022. Learned Query Optimizer: At the Forefront of AI-Driven Databases. In *EDBT*. 1–4.