# Machine Learning for Databases

Guoliang Li, Xuanhe Zhou
Tsinghua University, Beijing, China
liguoliang@tsinghua.edu.cn,zhouxuan19@mails.tsinghua.edu.cn

Lei Cao
MIT, Cambridge, MA USA
lcao@csail.mit.edu

## ABSTRACT

Machine learning techniques have been proposed to optimize the databases. For example, traditional empirical database optimization techniques (e.g., cost estimation, join order selection, knob tuning, index and view advisor) cannot meet the high-performance requirement for large-scale database instances, various applications and diversified users, especially on the cloud. Fortunately, machine learning based techniques can alleviate this problem by judiciously selecting optimization strategy. In this tutorial, we categorize database tasks into three typical problems that can be optimized by different machine learning models, including NP-hard problems (e.g., knob space exploration, index/view selection, partition-key recommendation for offline optimization; query rewrite, join order selection for online optimization), regression problems (e.g., cost/cardinality estimation, index/view benefit estimation, query latency prediction), and prediction problems (e.g., query workload prediction). We review existing machine learning based techniques to address these problems and provide research challenges.

**PVLDB Reference Format:**
Guoliang Li, Xuanhe Zhou, Lei Cao. Machine Learning for Databases. PVLDB, 14(12): 3190 - 3193, 2021.
doi:10.14778/3476311.3476405

## 1 INTRODUCTION

Machine learning (ML) techniques have been extensively studied in recent years to optimize the databases. For example, traditional database optimization techniques (e.g., cost estimation, join order selection, knob tuning, index and view advisor) are based on empirical methodologies and specifications, and requires human involvement (e.g., DBAs) to tune and maintain the databases. Thus existing empirical techniques cannot meet the high-performance requirement for large-scale database instances, various applications and diversified users, especially on the cloud. Fortunately, learning-based techniques can alleviate this problem. For instance, deep learning can improve the quality of cost estimation [7, 35, 36, 40], reinforcement learning can be used to optimize join order selection [21, 26, 38, 42], and deep reinforcement learning can be used to tune database knobs [2, 23, 46, 49].

**Tutorial Overview.** We plan to provide a 1.5 hours tutorial to thoroughly review existing ML techniques for databases.

**(1) Background and Motivation (10min).** We first introduce the background and motivation of learning based techniques.

**(2) Machine learning for Optimizing NP Problems (30min).** Many database optimization problems can be modeled as exploring the optimal solutions for NP-hard problems. However traditional empirical or heuristic based methods will fall in local optimums. Fortunately, machine learning based techniques can efficiently learn exploration models and utilize the learned models to optimize the NP-hard problems [1, 17, 18, 23, 25, 26, 38, 42, 43, 46, 47]. There are two main challenges here. First, it is challenging to select an appropriate model based on the scenario requirements. Second, it is challenging to select and characterize effective features. To present existing solutions of addressing these challenges, we classify the NP-hard problems into two cases based on exploring the solutions online or offline. (1.1) *Offline exploration:* it first trains a machine learning model to learn the exploration policy and then fine-tunes the model during online exploration, e.g., deep reinforcement learning for knob tuning [1, 18, 23, 46, 47], index advisor [17, 17, 25], view advisor [9, 43], join order selection [26, 27, 42], and partition-key recommendation [11]; (1.2) *Online exploration:* it pre-learns a model and utilizes the model to achieve online exploration, e.g., Monte Carlo Tree Search for query rewrite and join order selection [38, 42].

**(3) Machine learning for Regression Problems (30min).** Many database problems can be modeled as a regression problem. Cardinality estimation aims to estimate the cardinality of a query and a regression model (e.g., deep learning model) can be used [7, 10, 30, 35, 36, 40, 41]. Index/view benefit estimation aims to estimate the benefit of creating an index (or a view), and a regression model can be used to estimate the benefit [5, 19]. Latency prediction aims to estimate the execution time of executing a query and a regression model can be used to estimate the performance based on query and concurrency features [28, 50].

**(4) Machine learning for Prediction Problems (10min).** It is also vital to proactively optimize the database by predicting incoming queries. These prediction problems identify the temporal workload patterns and rearrange query execution to maximize the query performance or resource usage. And there are some machine learning methods for such prediction problems, e.g., cluster-based algorithms for trend prediction [24], reinforcement learning for workload scheduling [44].

**(5) Challenges and Opportunities (10min).** Finally, we provide research challenges and opportunities.

**Target Audience.** The audience includes VLDB attendees from both research and industry communities that are interested in database optimization and machine learning. The tutorial will be self-contained, and we will not require any prior background knowledge in machine learning.

**Difference with other Tutorials.** There are tutorials on machine learning and databases [20, 31, 32, 39]. Different from them, we focus on the fundamental problems of ML4DB and how to select suitable machine learning models to solve the problems.

**Related Works from Authors.** The authors did some works on learning-based database configuration [9, 23, 43, 46], optimizer [35, 42], monitoring [50], and autonomous systems [21, 22, 48, 49].

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476405

Table 1: Machine Learning Techniques for Databases

| | Database Problem | Method | Performance | Overhead | Training Data | Adaptivity |
|---|---|---|---|---|---|---|
| **Offline NP Problem** | knob space exploration | gradient-based [1, 18, 47] | High | High | High | – |
| | | dense network [37] | Medium | High/Medium | High | – / instance |
| | | DDPG [23, 46] | High | High | Low/Medium | query |
| | index selection | q-learning [19] | – | High | Low | – |
| | view selection | q-learning [43] | Medium | High | Low | – |
| | | DDQN [9] | High | High | Low | query |
| | partition-key selection | q-learning [11] | – | High | Low | – |
| **Online NP Problem** | join order selection | q-learning [27] | High | High | Low | – |
| | | DQN [26, 42] | High | High | Low | query |
| | | MCTS [38] | Medium | Low | Low | instance |
| | query rewrite | MCTS [21, 49] | – | Low | Low | query |
| **Regression Problem** | cost estimation | tree-LSTM [35] | High | High | High | query |
| | cardinality estimation | tree-ensemble [7] | Medium | Medium | High | query |
| | | autoregressive [41] | High | High/Medium | Low | data |
| | | dense network [16] | High | High | High | query |
| | | sum-product [12] | Medium | High | Low | data |
| | index benefit estimation | dense network [5] | – | High | High | query |
| | view benefit estimation | dense network [9] | – | High | High | query |
| | latency prediction | dense network [28] | Medium | High | High | query |
| | | graph embedding [50] | High | High | High | instance |
| | learned index | dense network [3] | – | High | High | query |
| **Prediction Problem** | trend prediction | clustering-based [24] | – | Medium | Medium | instance |
| | transaction scheduling | q-learning [44] | – | High | Low | query |

## 2 TUTORIAL OUTLINE

### 2.1 Optimizing NP-hard Problems

Many database optimization problems can be modeled as optimizing NP-hard problems. We can further categorize these NP-hard problems into offline optimization and online optimization. The former does not care about the inference time (the inference time can be seconds or even minutes); while the latter has instant inference requirement (e.g., the inference time should be milliseconds). Thus, the former trains a machine learning model to learn the exploration policy and then fine-tunes the model during online exploration, e.g., deep reinforcement learning for knob tuning [1, 18, 23, 46, 47], index advisor [17, 17, 25], view advisor [9, 43], and partition-key recommendation [11]; while the latter pre-learns a model and utilizes the model to achieve online exploration, e.g., Monte Carlo Tree Search for query rewrite and join order selection [26, 27, 38, 42].

**Offline Optimizing NP-hard Problems [1, 11, 17, 18, 23, 25, 43, 46, 47].** We first train a model and then use the model to optimize the NP-hard problems. The model can be fine-tuned during optimizing the NP-hard problems, and thus may be heavy.

(1) *Knob space exploration* [1, 2, 18, 23, 45–47]. Databases have hundreds of knobs and traditional databases rely highly on DBAs to tune the knobs in order to support knob tuning in different scenarios. Recently, learning-based techniques are proposed to improve the tuning performance or resource utilization. There are three types of models. (*i*) Gradient-based models [1, 18, 47] like Gaussian Process are widely used to explore local-optimal knob settings based on gradient descent. Besides, Zhang et al [47] share the learned workload encoder across instances to improve the generalization ability; (*ii*) Similarly, deep-learning models [37] estimate the performance of selected knob settings. They take selected knobs and internal metrics as input and output the predicted response time; (*iii*) Reinforcement-learning-based methods [23, 46] take knob tuning as a trail-and-error procedure, where the agent inputs tuning factors (e.g., internal metrics, queries), outputs proper knobs, and relies on the execution results to learn the tuning policy. We also summarize the advantages and disadvantages of these three types of models as shown in Table 1.

(2) *Index/View selection* [5, 9, 17, 19, 43]. Database indexes and views are fairly crucial to achieve high performance. But it is expensive to recommend and build appropriate indexes/views with a large number of columns/tables and queries/subqueries. Hence, there are some reinforcement-learning models that recommend indexes [17, 25] and materialized views [9, 43]. For example, they formalize view selection as an integer programming problem and use RL models like DDQN to solve this problem, where the state denotes the query and view features, and the action is adding/removing a view.

(3) *Database Partition* [11]. Traditional methods heuristically select columns as partition keys (single column mostly) and cannot balance between load balancing and access efficiency. Some work [11] also utilizes reinforcement learning model to explore different partition keys and implements a fully-connected neural network to estimate partition benefits. The RL model encodes table, query, and existing partition features, and iteratively selects new partition keys or replicate tables to optimize the overall performance.

**Online Optimizing NP-hard problems [26, 27, 38, 42].** Compared with offline NP problems, some database problems (e.g., query rewrite, online plan optimization) have instant feedback requirements (e.g., optimizing within milliseconds). Hence, it cannot tolerate a long time to update a model. Taking query rewriting as an example, traditional methods rely on heuristic rules or greedy algorithms to quickly select query plans, but the optimization quality cannot be guaranteed, especially when the query is complicated.

Hence, we want to select machine learning models that can (*i*) adaptively learn the policy during optimization and (*ii*) balance between performance and efficiency.
(1) *SQL rewriter* [21, 49]. Rule-based query rewriting methods may not find high-quality rules and appropriate rule order. Instead, learned tree search algorithms (e.g., MCTS) can be used to judiciously select the appropriate rules and apply rules in a good order.
(2) *Join order selection* [26, 27, 38, 42]. A SQL query may have millions, even billions of possible plans and it is very important to efficiently find a good plan. Traditional heuristics methods cannot find optimal plans for dozens of tables and dynamic programming is costly to explore the huge plan space. Thus there are some deep reinforcement learning based methods [26, 27, 42] that automatically select good plans. However, reinforcement learning models take too long time (e.g., hours) to learn the join policy for specific scenarios (e.g., workload, schema). Hence, there are works that utilize tree search algorithms like MCTS to adaptively learn the optimal join strategy while query execution [38].

## 2.2 Optimizing Regression Problems

**Regression Problems [5–8, 13, 13–15, 19, 28–30, 35, 50].** Some database problems, like cardinality estimation and index/view benefit estimation, can be modeled as regression problems, which fit the input variables (e.g., data tuples, query features) into estimation features. Traditional methods are based on empirical functions to estimate the execution costs or index benefit, which cannot handle complex scenarios (e.g., multiple index) and have low accuracy. Hence, we want to develop deep probabilistic models to achieve high estimation accuracy.
(1) *Cardinality/Cost estimation* [7, 10, 30, 35, 36, 40, 41]. Database optimizer relies on cardinality and cost estimation to select an optimized plan, but traditional techniques cannot effectively capture the correlations between different columns/tables and thus cannot provide high-quality estimation. Recently, deep learning based techniques (e.g., CNN [7], RNN [35], Mixture Model [30]) are proposed to estimate the cost and cardinality by using deep neural networks to capture data and query correlations. Based on different input features, we can categorize existing learning-based works into query-driven models [7, 35, 36] that learn a mapping from a query to its cardinality, data-driven models [10, 41] that learn the data distributions and use the distributions to estimate the cardinality of a query, and mixture models [30, 40] that combine the query model and data model.
(2) *Index/view benefit estimation* [5, 19]. To verify the effectiveness of selected indexes/views, it will be time consuming to actually build the indexes (or materializing views) and run the workload. Hence, there are learning based methods that replace empirical equations with neural networks to estimate the index/view benefit [5, 19], i.e., the neural network inputs the plans with/without index/view, and output the reduced costs.
(3) *Query latency prediction* [14, 28, 50]. Traditional methods rely on database administrators to monitor most database activities and report the problems, which is incomplete and inefficient. Thus, machine learning based techniques [14, 28, 50] are proposed to automatically predict query performance. For example, graph-embedding-based work [50] first characterizes the execution state as a graph model, where the vertices are operators being executed and edges are operator correlations (e.g., table share/conflict). And then they apply a graph convolution network to input the graph and learn the execution time of each vertex in the graph.

## 2.3 Optimizing Prediction Problems

**Machine learning for Prediction Problems** [24, 33, 34, 44]. Query workload prediction (or transaction prediction) is also important to database optimization (e.g., resource control, transaction scheduling). Traditional workload prediction methods are rule-based. For example, a rule-based method [4] uses domain knowledge of database engines (e.g., internal latency, resource utilization) to identify signals that are relevant to workload characteristics, e.g., memory utilization. However, rule-based methods waste much time to rebuild a statistics model when workload changes. Hence, learning based methods [24] predict the future trend of different workloads. First, they model the workload features based on query structures and arrival rate. Second, they cluster queries with similar arrival rates based on clustering algorithm like DBSCAN. Third, they predict the arrival rate patterns of queries in each cluster. Besides, for transaction management, traditional techniques are based on static transaction protocols, e.g., OCC, PCC, MVCC, which may cause sub-optimum. Hence, there are studies that utilize learned models to predict and schedule the transactions [33, 34, 44]. For example, in Q-learning [44], they character the state of memory blocks as input features, and output queries that execute together.

## 2.4 Open Problems

There are several research challenges that utilize ML techniques to optimize databases.
(1) *Model Selection.* It aims to select an appropriate model. There are two challenges. First, there are different kinds of ML models (e.g., forward-feeding, sequential, graph embedding) and it is inefficient to try out those models and manually adjust the parameters. Second, it is hard to evaluate whether a learned model is effective in most scenarios, for which a validation model is required.
(2) *Training data.* Most learning-based models require large-scale, high-quality, diversified training data to achieve high performance. However, it is hard to get training data, because the data either is security critical or relies on DBAs.
(3) *Adaptability.* Learning-based models are generally trained on specific scenarios and the adaptability is a big challenge, e.g., adapting to dynamic data updates, new workloads, datasets, hardware environments, and other systems.
(4) *Model convergence.* It is important to know whether a learned model can converge. If the model cannot converge, we need to provide alternative ways to avoid making inaccurate decisions.
(5) *Learning for OLAP.* Traditional OLAP focuses on relational data analytics. However, in the big data era, many new data types have emerged, e.g., graph data, time-series data, spatial data, it calls for new data analytics techniques to analyze these multi-model data.
(6) *Learning for OLTP.* Transaction modeling and scheduling are important to OLTP systems, because different transactions may have conflicts. However, it is not free to model and schedule transactions, and it calls for more efficient models that can instantly model and schedule the transactions in multiple cores and multiple machines.

## 3 BIOGRAPHY

**Guoliang Li** is a professor in the Department of Computer Science, Tsinghua University. His research interests mainly include data cleaning and integration and machine learning for databases. He got VLDB 2017 early research contribution award, TCDE 2014 early career award, CIKM 2017 best paper award.

**Xuanhe Zhou** is currently a PhD student in the Department of Computer Science, Tsinghua University. His research interests lie in AI/ML for data management.

**Lei Cao** is a Postdoc Associate at MIT CSAIL, working with Prof. Samuel Madden and Prof. Michael Stonebraker. Before that he worked for IBM T.J. Watson Research Center as a Research Staff Member. He received his Ph.D. in Computer Science from Worcester Polytechnic Institute, supervised by Prof. Elke Rundensteiner. He focused on developing end-to-end tools for data scientists.

## ACKNOWLEDGMENTS

This work was supported by NSF of China (61925205, 61632016), Beijing National Research Center for Information Science and Technology, Huawei, TAL education. Guoliang Li and Xuanhe Zhou are corresponding authors.

## REFERENCES

[1] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *SIGMOD 2017*, pages 1009–1024, 2017.

[2] D. V. Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Billian, and A. Pavlo. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *VLDB*, 14(7):1241–1253, 2021.

[3] A. Al-Mamun, H. Wu, and W. G. Aref. A tutorial on learned multi-dimensional indexes. In C. Lu, F. Wang, G. Trajcevski, Y. Huang, S. D. Newsam, and L. Xiong, editors, *SIGSPATIAL*, pages 1–4, 2020.

[4] S. Das, F. Li, V. R. Narasayya, and A. C. König. Automated demand-driven resource scaling in relational database-as-a-service. In *SIGMOD 2016*, pages 1923–1934, 2016.

[5] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya. AI meets AI: leveraging query executions to improve index recommendations. In *SIGMOD*, pages 1241–1258, 2019.

[6] J. Ding, U. F. Minhas, J. Yu, and et al. ALEX: an updatable adaptive learned index. In *SIGMOD*, pages 969–984, 2020.

[7] A. Dutt, C. Wang, A. Nazi, and et al. Selectivity estimation for range predicates using lightweight models. *VLDB*, 12(9):1044–1057, 2019.

[8] H. Grushka-Cohen, O. Biller, O. Sofer, L. Rokach, and B. Shapira. Diversifying database activity monitoring with bandits. *CoRR*, abs/1910.10777, 2019.

[9] Y. Han, G. Li, H. Yuan, and J. Sun. An autonomous materialized view management system with deep reinforcement learning. In *ICDE*, 2021.

[10] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep learning models for selectivity estimation of multi-attribute queries. In *SIGMOD*, pages 1035–1050, 2020.

[11] B. Hilprecht, C. Binnig, and U. Röhm. Learning a partitioning advisor for cloud databases. In *SIGMOD*, pages 143–157, 2020.

[12] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! *VLDB*, 13(7):992–1005, 2020.

[13] S. Idreos, N. Dayan, W. Qin, M. Akmanalp, S. Hilgard, A. Ross, J. Lennon, V. Jain, H. Gupta, D. Li, and Z. Zhu. Design continuums and the path toward self-designing key-value stores that know and learn. In *CIDR*, 2019.

[14] H. Kaneko and K. Funatsu. Automatic database monitoring for process control systems. In *IEA/AIE 2014*, pages 410–419, 2014.

[15] J. K. Z. Kang, Gaurav, S. Y. Tan, F. Cheng, S. Sun, and B. He. Efficient deep learning pipelines for accurate cost estimations over large scale query workload. In *SIGMOD*, pages 1014–1022, 2021.

[16] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*, 2019.

[17] J. Kossmann, S. Halfpap, M. Jankrift, and R. Schlosser. Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms. *Proc. VLDB Endow.*, 13(11):2382–2395, 2020.

[18] M. Kunjir and S. Babu. Black or white? how to develop an autotuner for memory-based analytics. In *SIGMOD*, pages 1667–1683, 2020.

[19] H. Lan, Z. Bao, and Y. Peng. An index advisor using deep reinforcement learning. In *CIKM*, pages 2105–2108, 2020.

[20] G. Li, X. Zhou, and L. Cao. AI meets database: AI4DB and DB4AI. In *SIGMOD*, pages 2859–2866, 2021.

[21] G. Li, X. Zhou, S. Ji, X. Yu, Y. Han, L. Jin, W. Li, T. Wang, and S. Li. opengauss: An autonomous database system. *VLDB*, 2021.

[22] G. Li, X. Zhou, and S. Li. Xuanyuan: An ai-native database. *IEEE Data Eng. Bull.*, 42(2):70–81, 2019.

[23] G. Li, X. Zhou, S. Li, and B. Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. *VLDB*, 12(12):2118–2130, 2019.

[24] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *SIGMOD*, pages 631–645, 2018.

[25] L. Ma, B. Ding, S. Das, and et al. Active learning for ML enhanced database systems. In *SIGMOD*, pages 175–191, 2020.

[26] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *SIGMOD 2018*, pages 3:1–3:4, 2018.

[27] R. C. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *PVLDB*, 12(11):1705–1718, 2019.

[28] R. C. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. *VLDB*, 12(11):1733–1746, 2019.

[29] V. Nathan, J. Ding, M. Alizadeh, and T. Kraska. Learning multi-dimensional indexes. In *SIGMOD*, pages 985–1000, 2020.

[30] Y. Park, S. Zhong, and B. Mozafari. Quicksel: Quick selectivity learning with mixture models. In *SIGMOD*, pages 1017–1033, 2020.

[31] C. Ré, D. Agrawal, M. Balazinska, and et al. Machine learning and databases: The sound of things to come or a cacophony of hype? In *SIGMOD*, pages 283–284, 2015.

[32] I. Sabek and M. F. Mokbel. Machine learning meets big spatial data. *PVLDB*, 12(12):1982–1985, 2019.

[33] Y. Sheng, A. Tomasic, T. Sheng, and A. Pavlo. Scheduling OLTP transactions via machine learning. *CoRR*, abs/1903.02990, 2019.

[34] H. J. Singh and A. S. Hafid. Prediction of transaction confirmation time in ethereum blockchain using machine learning. In *BLOCKCHAIN*, pages 126–133, 2019.

[35] J. Sun and G. Li. An end-to-end learning-based cost estimator. *PVLDB*, 13(3):307–319, 2019.

[36] J. Sun, G. Li, and N. Tang. Learned cardinality estimation for similarity queries. In *SIGMOD*, pages 1745–1757, 2021.

[37] J. Tan, T. Zhang, F. Li, J. Chen, Q. Zheng, P. Zhang, H. Qiao, Y. Shi, W. Cao, and R. Zhang. ibtune: Individualized buffer tuning for large-scale cloud databases. *PVLDB*, 12(10):1221–1234, 2019.

[38] I. Trummer, J. Wang, D. Maram, S. Moseley, S. Jo, and J. Antonakakis. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In *SIGMOD 2019*, pages 1153–1170, 2019.

[39] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K. Tan. Database meets deep learning: Challenges and opportunities. *SIGMOD Rec.*, 45(2):17–22, 2016.

[40] P. Wu and G. Cong. A unified deep model of learning from both data and queries for cardinality estimation. In *SIGMOD*, pages 2009–2022, 2021.

[41] Z. Yang, E. Liang, A. Kamsetty, C. Wu, and et al. Deep unsupervised cardinality estimation. *VLDB*, 13(3):279–292, 2019.

[42] X. Yu, G. Li, C. chai, and N. Tang. Reinforcement learning with tree-lstm for join order selection. In *ICDE 2020*, pages 196–207, 2019.

[43] H. Yuan, G. Li, L. Feng, and et al. Automatic view generation with deep learning and reinforcement learning. In *ICDE*, pages 1501–1512, 2020.

[44] C. Zhang, R. Marcus, A. Kleiman, and O. Papaemmanouil. Buffer pool aware query scheduling via deep reinforcement learning. *CoRR*, abs/2007.10568, 2020.

[45] J. Zhang, G. Li, Y. Liu, and et al. Cdbtune+: An efficient deep reinforcement learning-based automatic cloud database tuning system. *VLDBJ*, 2021.

[46] J. Zhang, Y. Liu, K. Zhou, G. Li, and et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *SIGMOD 2019*, pages 415–432, 2019.

[47] X. Zhang, H. Wu, Z. Chang, S. Jin, J. Tan, F. Li, T. Zhang, and B. Cui. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *SIGMOD*, pages 2102–2114, 2021.

[48] X. Zhou, C. Chai, G. Li, and J. Sun. Database meets artificial intelligence: A survey. *TKDE*, 2020.

[49] X. Zhou, L. Jin, S. Ji, X. Zhao, X. Yu, S. Li, T. Wang, K. Li, and L. Liu. Dbmind: A self-driving platform in opengauss. *VLDB*, 2021.

[50] X. Zhou, J. Sun, G. Li, and J. Feng. Query performance prediction for concurrent queries using graph embedding. *VLDB*, 13(9):1416–1428, 2020.