AI Meets Database: AI4DB and DB4AI

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

Lei Cao
CSAIL, MIT
USA
lcao@csail.mit.edu

ABSTRACT

Database and Artificial Intelligence (AI) can benefit from each other. On one hand, AI can make database more intelligent (AI4DB). 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, learning-based techniques can alleviate this problem. On the other hand, database techniques can optimize AI models (DB4AI). For example, AI is hard to deploy, because it requires developers to write complex codes and train complicated models. Database techniques can be used to reduce the complexity of using AI models, accelerate AI algorithms and provide AI capability inside databases. DB4AI and AI4DB have been extensively studied recently. In this tutorial, we review existing studies on AI4DB and DB4AI. For AI4DB, we review the techniques on learning-based database configuration, optimization, design, monitoring, and security. For DB4AI, we review AI-oriented declarative language, data governance, training acceleration, and inference acceleration. Finally, we provide research challenges and future directions in AI4DB and DB4AI.

ACM Reference Format:
Guoliang Li, Xuanhe Zhou and Lei Cao. 2021. AI Meets Database: AI4DB and DB4AI. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3448016.3457542

1 INTRODUCTION

Artificial intelligence (AI) and database (DB) have been extensively studied over the last five decades. First, database systems have been widely used in many applications, because databases are easy to use by providing user-friendly declarative query paradigms and encapsulating complicated query optimization functions. Second, AI has recently made breakthroughs due to three driving forces: large-scale data, new algorithms and high computing power.

Moreover, AI and database can benefit from each other. On one hand, AI can make database more intelligent (AI4DB). For example, traditional empirical 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, reinforcement learning can be used to optimize join order selection, and deep reinforcement learning can be used to tune database knobs [3, 42, 87].

On the other hand, database techniques can optimize AI models (DB4AI). In many real applications, AI is hard to deploy, because it requires developers to write complex codes and train complicated models. Fortunately, database techniques can be used to reduce the complexity of using AI models, accelerate AI algorithms and provide AI capability inside databases. For example, database techniques can be used to improve data quality (e.g., data discovery, data cleaning, data integration, data labeling, and data lineage), automatically select appropriate models, recommend model parameters, and accelerate the model inference.

DB4AI and AI4DB have been extensively studied recently [17, 26, 41, 44, 67, 73, 76, 77, 89]. In this tutorial, we summarize existing techniques on DB4AI and AI4DB, and provide research challenges and open problems.

Tutorial Overview. We will provide a 3 hours tutorial. In the first section (1.5 hours), we introduce AI4DB techniques.
(1) *Learning-based database configuration* (20min). It aims to utilize machine learning techniques to automate database configurations, e.g., deep reinforcement learning for knob tuning [42, 87], classifier for index advisor [30, 50], reinforcement learning for view advisor [21, 30, 45, 84], SQL rewriter, and reinforcement learning for database partition [23].
(2) *Learning-based database optimization* (20min). It aims to utilize machine learning techniques to address the hard problems in database optimization, e.g., SQL rewrite, cost/cardinality estimation [70, 80, 82], join order selection [54, 55, 83], and end-to-end optimizer [53, 80].
(3) *Learning-based database design* (20min). It aims to utilize machine learning techniques to design database components, e.g., learned indexes [12, 24, 59], learned KV store design [24, 25], and transaction management [86].
(4) *Learning-based database monitoring* (20min). Traditional methods rely on DBAs to monitor database activities and report the anomalies, and these methods are incomplete and inefficient for autonomous monitoring. Thus, machine learning based techniques are proposed to predict query arrival rates [49], estimate query performance [56, 90], diagnose root causes of slow queries [51], and determine when and how to monitor what database metrics [28].
(5) *Learning-based database security* (10min). Traditional database security techniques (e.g., data masking and auditing, sensitive data

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457542

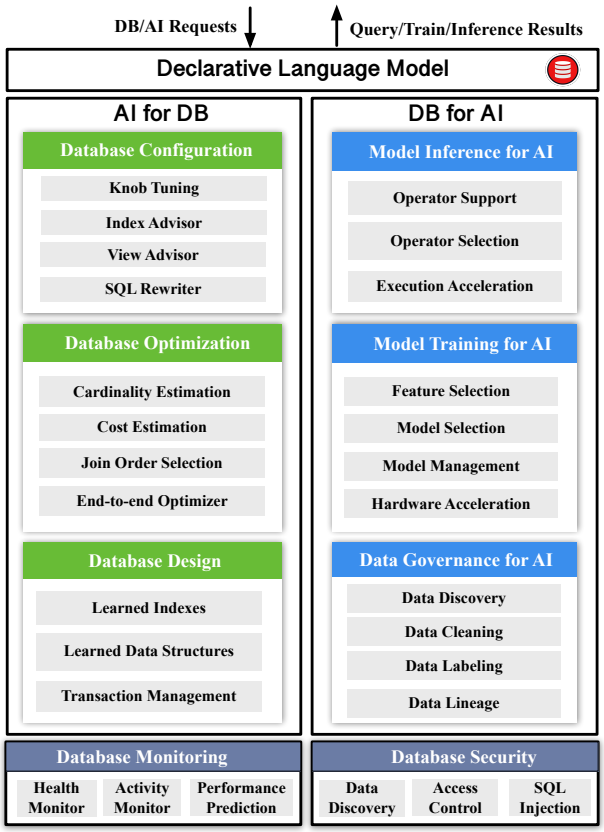

Figure 1: Overview of DB4AI and AI4DB.

discovery) rely on user-defined rules, which cannot automatically detect unknown security vulnerabilities. Learning based algorithms are proposed to discover sensitive data, detect anomaly [46], conduct access control [18], and avoid SQL injection [72].

In the second section, we focus on DB4AI techniques (70min). To lower the barrier for using AI, the database community extends the database techniques to encapsulate the complexity of AI algorithms and enable users to use declarative languages, e.g., SQL, to utilize the AI algorithms.

(1) *Declarative language model* (15min). SQL is relatively easy to be used and widely accepted in database systems. We can extend SQL to support AI models [66], and we can also design user-friendly tools to support AI models [15].

(2) *Data governance* (15min). Data quality is important for machine learning, and we can use data governance techniques to improve data quality and enhance the efficiency, e.g., data discovery [16], data cleaning [79], data labeling [40, 57], and data lineage.

(3) *Model training* (20min). Model training is a time-consuming and complicated process, and thus it requires optimization techniques, e.g., feature selection [85], model selection, model management [75], hardware acceleration [29].

(4) *Model inference* (20min). Model inference aims to effectively infer the results using a trained model with in-database optimization techniques, e.g., operator support, operator selection, execution acceleration.

In the third section, we provide research challenges and open problems (20min). For AI4DB, we discuss AI for transactions and database reliability when AI models cannot converge. For DB4AI, we discuss how to enhance AI training inside database, how to reduce errors with error-tolerant techniques, and build a database-like AI optimizer. For hybrid AI and DB, we present hybrid data models, hybrid computation models, and hybrid AI&DB systems.

Target Audience. The intended audience include SIGMOD attendees from both research and industry communities that are interested in database optimization and machine learning. We will not require any prior background knowledge in database or machine learning domain. The tutorial will be self-contained, and we will include a broad introduction and motivating examples for non-specialists to follow.

Difference with Existing Tutorials. There are some existing tutorials on machine learning and databases [62, 64, 76]. Different from them, we focus on the fundamental techniques for using AI techniques to optimize databases and using DB techniques to accelerate AI models.

2 TUTORIAL OUTLINE

We start with a brief overview of this tutorial, to give the audience a clear outline and talk goals. We then summarize existing techniques of AI4DB, DB4AI, and hybrid AI and DB. Finally, we provide research challenges and open problems.

2.1 AI for DB

Traditional database design is based on empirical methodologies and specifications, and requires human involvement (e.g., DBAs) to tune and maintain the databases. AI techniques can be used to alleviate these limitations – exploring larger design space than humans and replacing heuristics to address hard problems. We categorize existing AI4DB techniques as below.

Learning-based Database Configuration. It aims to utilize AI techniques to automate database configurations, including knob tuning, index advisor, materialized view advisor, SQL rewriter, and database partition.

(1) *Knob tuning*. Databases have hundreds of tunable system knobs (e.g., `Work_Mem`, `Max_Connections`) [48], which control many important aspects of databases (e.g., memory allocation, I/O control, logging) and affect database performance. Traditional manual methods leverage DBAs to manually tune these knobs based on their experiences but they always spend too much time to tune the knobs (several days to weeks) and cannot handle millions of database instances on cloud databases. To address this problem, the database community utilizes learning-based techniques [38, 42, 87] to automate knob tuning, which not only achieve higher tuning performance but less tuning time. For example, CDBTune [87] models database tuning as a sequential decision problem and relies on reinforcement learning to improve tuning performance. Moreover, since CDBTune only takes previous database state as input, QTune [42] further characterizes query features using deep learning and can achieve finer granularity tuning, e.g., query-level tuning, session-level tuning, and system-level tuning.

(2) *Index advisor*. Indexes are vital to speed up query execution, and indexes on appropriate columns can achieve high performance. However, it is expensive to recommend and build indexes with large

number of column combinations. Hence, there are some learning-based works that automatically recommend indexes [30, 50]. For example, Sadri et al [65] propose a reinforcement-learning-based index selection method. First, without expert rules, they denote workload features as the arrival rate of queries, column features as the access frequency and selectivity of each column. Second, they use the Markov Decision Process model (MDP) to learn from features of queries, columns, and outputs a set of actions, which denote creating/dropping an index.

(3) *View advisor.* It is important in DBMS that utilizes views to improve the query performance based on the space-for-time trade-off principle. Judiciously selecting materialized views can significantly improve the query performance within an acceptable overhead. However, existing methods rely on DBAs to generate and maintain materialized views. Unfortunately, even DBAs cannot handle large-scale databases, especially cloud databases that have millions of database instances and support millions of users. Thus, it calls for the view advisor, which automatically identifies the appropriate views for a given query workload [30, 45, 84]. For example, Han et al [21] propose a deep reinforcement learning method to estimate the benefit of different MV candidates and queries, and select MVs for dynamic workloads.

(4) *SQL rewriter.* SQL rewriter can remove the redundant or inefficient operators in logic query and enhance query performance significantly. However, there are numerous rewrite orders for a slow query (e.g., different operators and applicable rules), and traditional empirical query rewriting methods only rewrite in a fixed order (e.g., top down) and may derive suboptimal queries. Instead, deep reinforcing learning can be used to judiciously select the appropriate rules and apply the rules in a good order.

(5) *Database Partition.* Traditional methods heuristically select columns as partition keys (single column mostly) and cannot balance between load balance and access efficiency. Some work [23] also utilizes reinforcement learning model to explore different partition keys and implements a fully-connected neural network to estimate partition benefits.

Learning-based Database Optimization. It aims to utilize machine learning techniques to address the hard problems in database optimization, including cost estimation, join order selection, and end-to-end optimizers.

(1) *Cardinality/Cost estimation.* 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 [13], RNN [70], Mixture Model [60]) are proposed to estimate the cost and cardinality by using deep neural networks to capture data correlations. For example, a LSTM based work [70] learns a representation for each sub-plan with physical operator and predicates, and outputs the estimated cardinality and cost simultaneously by using an estimation layer.

(2) *Join order selection.* 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 [54, 55, 83] that automatically select good plans. For example, SkinnerDB [74] uses a Monte-Carlo tree search based methods to try out different join orders in each time slice and can optimize the join order on the fly.

(3) *End-to-end optimizer.* A full-fledged optimizer not only replies on cost estimation and join order, but also requires to consider indexes and views, and it is important to design an end-to-end optimizer. Learning-based optimizers [53, 55, 80] use deep neural networks to optimize SQL queries. For example, Marcus et al [55] propose an end-to-end optimizer *NEO* to generate the final physical plan. Without information from the cost model, *NEO* uses PostgreSQL's plan to pre-train the neural network and uses latency as feedback to train the neural network. This end-to-end method learns from the latency to generate the whole physical plan, which can be applied to many scenarios and robust to estimation errors.

Learning-based Database Design. Traditional databases are designed by database architects based on their experiences, but database architects can only explore a limited number of possible design spaces. Recently some learning-based self-design techniques have been proposed.

(1) *Learned indexes* [12, 24, 59] are proposed for not only reducing the index size but also improving the query performance using the indexes. For example, Kraska et al. [32] propose that *indexes are models*, where the B^+tree index can be seen as a model that maps each query key to its page. Learned indexes are also studied for data updates and high-dimensional data.

(2) *Learned data structure design.* Different data structures may be suit for different environments (e.g., different hardware, different read/write transactions) and it is hard to design appropriate structures for every scenario. Techniques like data structure alchemy [24] are proposed to automatically recommend and design data structures. They define the *design space* by the fundamental design components (e.g., fence pointers, links, and temporal partitioning). To design a data structure, they first identify the bottleneck of the total cost and then tweak different knobs in one direction until reaching the cost boundary or the total cost is minimal, which is similar to the gradient descent procedure.

(3) *Learning-based transaction management.* Effective workload scheduling can greatly improve the performance by avoiding the data conflicts. We introduce learned transaction management techniques from two aspects: transaction prediction and transaction scheduling. First, for transaction prediction, traditional workload prediction methods are rule-based. For example, a rule-based method [11] uses domain knowledge of database engines (e.g., internal latency, resource utilization) to identify signals relevant to workload characteristics, which takes much time to rebuild a statistics model when workload changes, so Ma et al. [49] propose an ML-based system that predicts the future trend of different workloads. Second, for transaction scheduling, traditional database systems either schedule workload sequentially, which cannot consider potential conflicts, or schedule workloads based on the execution costs predicted by the database optimizer. Sheng et al. [68] propose a learning based transaction scheduling method, which can balance concurrency and conflict rates using supervised algorithms.

Learning-based Database Monitoring. Traditional methods rely on database administrators to monitor most database activities and report the anomalies and these methods are incomplete and inefficient. Thus, machine learning based techniques [28, 56, 90] are proposed for three main cases – health monitor, activity monitor and performance prediction.

(1) *Database Health Monitor (DHM).* It records database health related metrics, e.g., the number of queries per second, the query latency, to optimize database or diagnose failures. In [51], they assume that intermittent slow queries with similar key performance indicators (e.g., CPU usage, transactions per second) have the same root causes. Thus, they first extract slow SQLs from the failure records, cluster them with KPI states, and ask DBAs to assign root causes for each cluster. Next, for an incoming slow SQL, they match it to a cluster C based on similarity score of KPI states. If matched, they use the root cause of C to notify DBAs; otherwise, they generate a new cluster and ask DBAs to assign the root causes. However, these methods [51] cannot prevent potential database failure and it highly relies on DBA's experience. So Taft et al. [71] propose to proactively monitor database to adapt to workload changes.

(2) *Database Activity Monitor (DAM).* It externally monitors and controls database activities (e.g., creating new accounts, viewing sensitive information), which are vital to protecting sensitive data. We broadly classify DAM into two classes, activity selection and activity trace. For activity selection, traditional DAM methods are required to record all the activities on extra systems according to trigger rules [10]. However, it is still a heavy burden to record all the activities, which brings frequent data exchanges between databases and monitoring systems. Hence, it requires to automatically select and record risky activities, and Hagit et al. [19] take database monitoring as a multi-armed bandits problem (MAB), which selects risky database activities by exploiting current policy and exploring new policies. The goal is to train an optimal policy with the maximal risk score.

(3) *Performance Prediction.* Query performance prediction is vital to meet the service level agreements (SLAs), especially for concurrent queries. Marcus et al [56] use deep learning to predict query latency under concurrency scenarios, including interactions between child/parent operators, and parallel plans. However, it adopts a pipeline structure (causing information loss) and fails to capture operator-to-operator relations like data sharing/conflict features. Hence, Zhou et al [90] propose a performance prediction method with graph embedding. They use a graph model to characterize concurrent queries and utilize a graph convolution network to embed the workload graph into performance metrics.

Learning-based Database Security. Traditional database security techniques (e.g., data masking and auditing) rely on user-defined rules, which cannot automatically detect the unknown security vulnerabilities. Thus, learning based algorithms [18, 46, 72] are proposed to discover sensitive data, conduct access control, and avoid SQL injection.

(1) *Learning-based Sensitive Data Discovery.* Since sensitive data leakage will cause great financial and personal information loss, it is important to protect the sensitive data in a database. Sensitive data discovery aims to automatically detect and protect confidential data. For example, Fernandez et al. propose *Aurum* [16], which is a data discovery system that provides flexible queries to search dataset based on users' demand.

(2) *Access Control.* It aims to prevent unauthorized users to access the data, including table-level and record-level access control. Traditional methods cannot effectively prevent these attacks. Recently, machine learning based algorithms are proposed to estimate the legality of access requests. Colombo et al [9] propose a purpose-based access control model, which customizes control polices to regulate data requests. As different actions and data content may lead to different private problems, this method aims to learn legal access purposes.

(3) *SQL Injection.* SQL injection is a common and harmful vulnerability to database. Attackers may modify or view data that exceeds their priorities by bypassing additional information or interfering with the SQL statement, such as retrieving hidden data, subverting application logic, union attacks and etc. There are mainly two types of SQL injection detection methods that utilize machine learning techniques, including classification tree [47, 69] and fuzzy neural network [5].

Learning-based Database Systems. There are some learning-based database systems which are studied by both the academia and industry [31, 41, 61]. For example, `SageDB` [31] provided a vision to specialize the database implementation by learning the data distribution (CDF models) and designing database components based on the knowledge, e.g., learned index, learned query scheduling.

2.2 DB for AI

Declarative Language Model. Traditional machine learning algorithms are mostly implemented with programming languages (e.g., Python, R) and have several limitations. First, they require engineering skills to define the complete execution logic, e.g., the iterative patterns of model training, and tensor operations like matrix multiplication and flattening. Second, machine learning algorithms have to load data from database systems, and the data import/export costs may be very high. Instead, SQL is relatively easy to be used and widely accepted in database systems. However, SQL lacks some complex processing patterns (e.g., iterative training) compared with other high-level machine learning languages. Fortunately, SQL can be extended to support AI models [66], and we can also design user-friendly tools to support AI models in SQL statements [15].

Data Governance. AI models rely on high-quality data, and data governance aims to discover, clean, integrate, and label the data to improve the data quality.

(1) *Data discovery.* Data discovery aims to automatically find relevant datasets from data warehouse considering the applications and user needs. Learning based data discovery [8, 16, 20, 81, 88] enhances the ability of finding relevant data, which effectively finds out relevant data among a large number of data sources. For example, Fernandez et al. propose *Aurum* [16], which is a data discovery system that provides flexible queries to search dataset based on users' demand. It leverages enterprise knowledge graph (EKG) to capture a variety of relationships to support a wide range of queries. The EKG is a hyper-graph where each node denotes a table column, each edge represents the relationship between two nodes and

hyper-edges connect nodes that are hierarchically related such as columns in the same table.

(2) *Data cleaning.* Dirty or inconsistent data can affect the training performance terribly. Data cleaning and integration techniques [14, 27, 34, 36, 39] can detect and repair the dirty data, and integrate the data from multiple sources. Wang et al. propose a cleaning framework *ActiveClean* for machine learning tasks [34]. Given a dataset and machine learning model with a convex loss, it selects records that can improve the performance of the model most and cleans those records iteratively.

(3) *Data labeling.* We can properly utilize domain experts, crowdsourcing and existing knowledge to label a large number of training data for ML algorithms [1, 57]. For example, with commercial public crowdsourcing platforms like Amazon Mechanical Turk (https://www.mturk.com), crowdsourcing is an effective way to address such tasks by utilizing hundreds or thousands of workers to label the data.

Model Training. Model training aims to train a high-quality model for online inference. Model training is a time-consuming and complicated process, and thus it requires optimization techniques, including feature selection, model selection, model management and hardware acceleration.

(1) *Feature selection* aims to search appropriate features from a large number of possible features, which is laborious and time-consuming. Database techniques like batching [85], materialization [35, 85], active learning [4] are proposed to address this issue. For example, batching and materialization techniques [85] are utilized to reduce the feature enumeration cost. Active learning based techniques [4] are utilized to accelerate the evaluation process.

(2) *Model selection* aims to select an appropriate model (and parameter values) from a large number of possible models. Parallelism techniques are proposed to accelerate this step, including task parallel [58], bulk synchronous parallel [33], parameter server [43] and model parallelism. A key bottleneck of this problem is model selection throughput, i.e., the number of training configurations is tested per unit time. High throughput allows the user to test more configurations during a fixed period, which makes the entire training process efficient. A solution is to enhance the throughput is parallelism, and the popular parallelism strategies include task parallel [58] , bulk synchronous parallel [33] , and parameter server [43].

(3) *Model management.* Since model training is a trial-and-error process that needs to maintain many models and parameters that have been tried, it is necessary to design a model management system to track, store and search the ML models. We review GUI-based [6] and command-based model [75] management system.

(4) *Hardware acceleration.* Morden hardwares, like GPU and FPGA, are also utilized to accelerate the model training. We introduce hardware acceleration techniques in row-store and column-store [29] databases respectively. For example, *DAnA* [52] parses the query and utilizes a hardware mechanism that connects the FPGA and database. It retrieves the training data from the buffer pool to the accelerator directly without accessing the CPU. Besides, they design an execution model to combine thread-level and data-level parallelism for accelerating the ML algorithms.

Model Inference. It aims to infer the results using a trained model with in-database optimization techniques.

(1) *Operator support.* An ML model may contain different types of operators (e.g., scalar, tensor), which have different optimization requirements. Thus in-database techniques are proposed to support AI operators [22, 78]. For example, Boehm et al. [7] propose an in-database machine learning system *SystemML*. *SystemML* supports matrix operations with user-defined aggregation functions, which provide parallel data processing in the column level. For example, Boehm et al. [7] propose an in-database machine learning system *SystemML*. *SystemML* supports matrix operations with user-defined aggregation functions, which provide parallel data processing in the column level.

(2) *Operator selection.* The same ML model can be converted to different physical operators, which may bring significant performance difference. In-database operator selection can estimate resource consumption and judiciously schedule the operators [7, 36].

(3) *Execution acceleration.* Different from model training, model inference needs to choose ML models and execute forward propagation to predict for different problems. Existing execution acceleration techniques include in-memory methods [37] and distributed methods[2, 63]. In-memory methods aim to compress data into memory and conduct in-memory computation as much as possible. And the distributed methods route tasks to different nodes and reduce the burden of data processing and model computation using parallel computing.

2.3 Challenges and Open Problems

Although AI4DB and DB4AI have been extensively studied, there are still many opportunities and challenges to apply AI4DB and DB4AI techniques in practice, and it also calls for hybrid AI and DB techniques.

AI4DB. There are several challenges that utilize AI techniques to optimize databases.

(1) *Model Selection.* There are two challenges. First, there are different kinds of ML models (e.g., forward-feeding, sequential, graph embedding) and it is inefficient to manually select appropriate models and 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) *Model Validation.* It is hard to evaluate whether a learned model is effective and outperforms non-learning methods. For example, whether a knob tuning strategy really works for a workload? It requires to design a validation model to evaluate a learned model.

(3) *Model Management.* Different database components may use different ML models and it is important to provide a unified ML platform to achieve a unified resource scheduling and a unified model management.

(4) *Training data.* Most AI models require large-scale, high-quality, diversified training data to achieve high performance. However, it is rather hard to get training data in AI4DB, because the data either is security critical or relies on DBAs. For example, in database knob tuning, the training samples are collected based on DBAs' experiences. And it is laborious to get a large number of training samples. Moreover, to build an effective model, the training data

should cover different scenarios, different hardware environments, and different workloads. It calls for new methods that use a small training dataset to get a high-quality model.

(5) *Adaptability.* The adaptability is a big challenge, e.g., adapting to dynamic data updates, other datasets, new hardware environments, and other database systems. We need to address the following challenges. First, how to adapt a trained model (e.g., optimizer, cost estimation) on a dataset to other datasets? Second, how to adapt a trained model on a hardware environment to other hardware environments? Third, how to adapt a trained model on a database to other databases? Fourth, how to make a trained model support dynamic data updates?

(6) *Model convergence.* It is very important that whether a learned model can be converged. If the model cannot be converged, we need to provide alternative ways to avoid making delayed and inaccurate decisions. For example, in knob tuning, if the model is not converged, we cannot utilize the model for online knob suggestion.

(7) *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. Moreover, besides traditional aggregation queries, many applications require to use machine learning algorithms to enhance data analytics, e.g., image analysis. Thus it is rather challenging to integrate AI and DB techniques to provide new data analytics functionality.

(8) *Learning for OLTP.* Transaction modeling and scheduling are rather important to OLTP systems, because different transactions may have conflicts. However, it is not free to model and schedule the transactions, and it calls for more efficient models that can instantly model and schedule the transactions in multiple cores and multiple machines.

DB4AI. There are still several challenges to utilize in-database techniques to optimize AI algorithms.

(1) *In-database training.* It is challenging to support AI training inside databases, including model storage, model update and parallel training. First, it is challenging to store a model in databases, such that the model can be trained and used by multi-tenants, and we need to consider the security and privacy issues. Second, it is challenging to update a model, especially when the data is dynamically updated.

(2) *Training acceleration using database techniques.* Most of studies focus on the effectiveness of AI algorithms but do not pay much attention to efficiency. It calls for utilizing database techniques to improve the performance of AI algorithms, e.g., indexes and views. For example, self-driving vehicles require a large number of examples for training, which is rather time consuming. Actually, it only requires some *important examples*, e.g., the training cases in the night or rainy day, but not many redundant examples. Thus we can index the samples and features for effective training.

(3) *AI optimizer.* Existing studies use user-defined functions (UDF) to support AI models, which are not effectively optimized. It requires to implement the AI models as operators insider databases, and design physical operators for each operator. Most importantly, it requires to push down the AI operators and estimate the cost/-cardinality of AI operators. It calls for an AI optimizer to optimize the AI training and inference. Furthermore, it is more important to efficiently support AI operators in a distributed environment.

(4) *Fault-tolerant learning.* Existing learning model training does not consider error tolerance. If a process crashes and the whole task will fail. We can use the error tolerance techniques to improve the robustness of in-database learning. However, to ensure business continuity under predictable/unpredictable disasters, database systems must provide capabilities like fault tolerance and disaster recovery.

AI&DB co-optimization. There are several challenges in model support, inference, and system integration.

(1) *Hybrid relational and tensor model.* Traditional CPU cannot efficiently process tensor models and AI chips cannot efficiently process relational models. It calls for effective methods that accelerate relational operations on AI chips, schedule operators across traditional CPU and AI chips, and supporting both relational and tensor model.

(2) *Hybrid DB&AI inference.* Many applications require both DB and AI operations, e.g., finding all the patients of a hospital whose stay time will be longer than 3 days. A native way is to predict the hospital stay of each patient and then prune the patients whose stay time is less than 3. Obviously this method is rather expensive, and it calls for a new optimization model to optimize both DB and AI, e.g, new optimization model, AI operator push-down, AI cost estimation, and AI index/views.

(3) *Hybrid DB&AI system.* It calls for an end-to-end hybrid AI&DB system that supports a declarative language, e.g., AISQL, which extends SQL to support AI operators, an AI&DB optimizer that co-optimizes the two operations, an effective (distributed) execution engine that schedules the two types of tasks, and an appropriate storage engine.

3 BIOGRAPHY

Guoliang Li is a full professor in the Department of Computer Science, Tsinghua University. His research interests mainly include data cleaning and integration, crowdsourcing, and AI4DB, DB4AI, hybrid DB&AI. He got VLDB 2017 early research contribution award, TCDE 2014 early career award, CIKM 2017 best paper award, and best of VLDB/ICDE. He will present AI4DB and open problems.

Xuanhe Zhou is currently a PhD student in the Department of Computer Science, Tsinghua University. His research interests lie in the interdisciplinary technologies of database and machine learning. He will present AI4DB.

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. He focused on developing end-to-end tools for data scientists to make sense of data. He will present DB4AI.

Acknowledgement. This paper was supported by NSF of China (61925205, 61632016), Huawei, Beijing National Research Center for Information Science and Technology(BNRist), and TAL education.

REFERENCES

[1] C. C. Aggarwal, X. Kong, Q. Gu, J. Han, and P. S. Yu. Active learning: A survey. In *Data Classification: Algorithms and Applications*, pages 571–606. 2014.

[2] D. Agrawal and et al. Rheem: Enabling multi-platform task execution. In *SIGMOD 2016*, pages 2069–2072, 2016.

[3] 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.

[4] M. R. Anderson and M. J. Cafarella. Input selection for fast feature engineering. In *ICDE 2016,*, pages 577–588, 2016.

[5] L. O. Batista, G. A. de Silva, V. S. Araújo, V. J. S. Araújo, T. S. Rezende, A. J. Guimarães, and P. V. de Campos Souza. Fuzzy neural networks to create an expert system for detecting attacks by SQL injection. *CoRR*, abs/1901.02868, 2019.

[6] L. Bavoil, S. P. Callahan, C. E. Scheidegger, H. T. Vo, P. Crossno, C. T. Silva, and J. Freire. Vistrails: Enabling interactive multiple-view visualizations. In *VIS 2005*, pages 135–142, 2005.

[7] M. Boehm, M. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. Reiss, P. Sen, A. Surve, and S. Tatikonda. Systemml: Declarative machine learning on spark. *PVLDB*, 9(13):1425–1436, 2016.

[8] N. Chepurko, R. Marcus, E. Zgraggen, and et al. ARDA: automatic relational data augmentation for machine learning. *VLDB*, 13(9):1373–1387, 2020.

[9] P. Colombo and E. Ferrari. Efficient enforcement of action-aware purpose-based access control within relational database management systems. In *ICDE 2016*, pages 1516–1517, 2016.

[10] C. Curino, E. P. C. Jones, S. Madden, and H. Balakrishnan. Workload-aware database monitoring and consolidation. In *SIGMOD 2011*, pages 313–324, 2011.

[11] 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.

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

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

[14] J. Fan and G. Li. Human-in-the-loop rule learning for data integration. *IEEE Data Eng. Bull.*, 41(2):104–115, 2018.

[15] S. Fernandes and J. Bernardino. What is bigquery? In *Proceedings of the 19th International Database Engineering & Applications Symposium, Yokohama, Japan, July 13-15, 2015*, pages 202–203, 2015.

[16] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE 2018*, pages 1001–1012, 2018.

[17] Z. Gharibshah, X. Zhu, A. Hainline, and M. Conway. Deep learning for user interest and response prediction in online display advertising. *Data Science and Engineering*, 5(1):12–26, 2020.

[18] M. L. Goyal and G. V. Singh. Access control in distributed heterogeneous database management systems. *Computers & Security*, 10(7):661–669, 1991.

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

[20] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *SIGMOD 2016*, pages 795–806, 2016.

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

[22] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The madlib analytics library or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012.

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

[24] 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.

[25] S. Idreos, N. Dayan, W. Qin, and et al. Learning key-value store design. *CoRR*, abs/1907.05443, 2019.

[26] S. Idreos and T. Kraska. From auto-tuning one size fits all to self-designed and learned data-intensive systems. In *SIGMOD*, pages 2054–2059. ACM, 2019.

[27] I. F. Ilyas and X. Chu. *Data Cleaning*. ACM, 2019.

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

[29] K. Kara, K. Eguro, C. Zhang, and G. Alonso. Columnml: Column-store machine learning with on-the-fly data transformation. *PVLDB*, 12(4):348–361, 2018.

[30] 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.

[31] T. Kraska, M. Alizadeh, A. Beutel, E. H. Chi, and et al. Sagedb: A learned database system. In *CIDR*, 2019.

[32] T. Kraska, A. Beutel, and E. H. C. et al. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.

[33] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR 2013*, 2013.

[34] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[35] A. Kumar, M. Boehm, and J. Yang. Data management in machine learning: Challenges, techniques, and systems. In *SIGMOD 2017*, pages 1717–1722, 2017.

[36] A. Kumar, J. F. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD 2016*, pages 19–34, 2016.

[37] M. Kunjir and S. Babu. Thoth in action: Memory management in modern data analytics. *PVLDB*, 10(12):1917–1920, 2017.

[38] M. Kunjir and S. Babu. Black or white? how to develop an autotuner for memory-based analytics [extended version]. *CoRR*, abs/2002.11780, 2020.

[39] G. Li. Human-in-the-loop data integration. *Proc. VLDB Endow.*, 10(12):2006–2017, 2017.

[40] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.

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

[42] 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.

[43] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su. Scaling distributed machine learning with the parameter server. In *OSDI 2014*, pages 583–598, 2014.

[44] M. Li, H. Wang, and J. Li. Mining conditional functional dependency rules on big data. *Big Data Mining and Analytics*, 03(01):68, 2020.

[45] X. Liang, A. J. Elmore, and S. Krishnan. Opportunistic view materialization with deep reinforcement learning. *CoRR*, abs/1903.01363, 2019.

[46] Z. Lin, X. Li, and X. Kuang. Machine learning in vulnerability databases. In *ISCID 2017*, pages 108–113, 2017.

[47] M. Lodeiro-Santiago, C. Caballero-Gil, and P. Caballero-Gil. Collaborative sql-injections detection system with machine learning. In *IML 2017*, pages 45:1–45:5, 2017.

[48] J. Lu, Y. Chen, H. Herodotou, and S. Babu. Speedup your analytics: Automatic parameter tuning for databases and big data systems. *PVLDB*, 12(12):1970–1973, 2019.

[49] 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 2018*, pages 631–645, 2018.

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

[51] M. Ma, Z. Yin, S. Zhang, S. Wang, and et al. Diagnosing root causes of intermittent slow queries in large-scale cloud databases. *VLDB*, 13(8):1176–1189, 2020.

[52] D. Mahajan, J. K. Kim, J. Sacks, A. Ardalan, A. Kumar, and H. Esmaeilzadeh. In-rdbms hardware acceleration of advanced analytics. *PVLDB*, 11(11):1317–1331, 2018.

[53] R. Marcus, P. Negi, H. Mao, and et al. Bao: Learning to steer query optimizers. *CoRR*, abs/2004.03814, 2020.

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

[55] 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.

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

[57] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL 2009*, pages 1003–1011, 2009.

[58] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. In *OSDI 2018*, pages 561–577, 2018.

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

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

[61] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In *CIDR 2017*, 2017.

[62] C. Ré, D. Agrawal, M. Balazinska, M. J. Cafarella, M. I. Jordan, T. Kraska, and R. Ramakrishnan. Machine learning and databases: The sound of things to come or a cacophony of hype? In *SIGMOD*, pages 283–284, 2015.

[63] J. M. Rzeszotarski and A. Kittur. Kinetica: naturalistic multi-touch data visualization. In *CHI 2014*, pages 897–906, 2014.

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

[65] Z. Sadri, L. Gruenwald, and E. Leal. Online index selection using deep reinforcement learning for a cluster database. In *ICDE*, pages 158–161, 2020.

[66] T. Schindler and C. Skornia. Secure parallel processing of big data using order-preserving encryption on google bigquery. *CoRR*, abs/1608.07981, 2016.

[67] C. Shan, N. Mamoulis, R. Cheng, G. Li, X. Li, and Y. Qian. An end-to-end deep RL framework for task arrangement in crowdsourcing platforms. In *ICDE*, pages 49–60, 2020.

[68] Y. Sheng, A. Tomasic, T. Sheng, and A. Pavlo. Scheduling OLTP transactions via machine learning. *CoRR*, abs/1903.02990, 2019.
[69] N. M. Sheykhkanloo. A learning-based neural network model for the detection and classification of SQL injection attacks. *IJCWT*, 7(2):16–41, 2017.
[70] J. Sun and G. Li. An end-to-end learning-based cost estimator. *PVLDB*, 13(3):307–319, 2019.
[71] R. Taft, N. El-Sayed, M. Serafini, Y. Lu, A. Aboulnaga, M. Stonebraker, R. Mayerhofer, and F. J. Andrade. P-store: An elastic database system with predictive provisioning. In *SIGMOD 2018*, pages 205–219, 2018.
[72] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu. SQL injection behavior mining based deep learning. In *ADMA 2018*, pages 445–454, 2018.
[73] S. Tian, S. Mo, L. Wang, and Z. Peng. Deep reinforcement learning-based approach to tackle topic-aware influence maximization. *Data Science and Engineering*, 5(1):1–11, 2020.
[74] I. Trummer, J. Wang, D. Maram, S. Moseley, S. Jo, and J. Antonakakis. Skinnerdb: Regret-bounded query evaluation via reinforcement learning. In *SIGMOD*, pages 1153–1170, 2019.
[75] M. Vartak, H. Subramanyam, W. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: a system for machine learning model management. In *SIGMOD 2016*, page 14, 2016.
[76] 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.
[77] Y. Wang, Y. Yao, H. Tong, F. Xu, and J. Lu. A brief review of network embedding. *Big Data Mining and Analytics*, 2(1):35, 2019.
[78] Y. R. Wang, S. Hutchison, D. Suciu, and et al. SPORES: sum-product optimization via relational equality saturation for large scale linear algebra. *VLDB*, 13(11):1919–1932, 2020.
[79] G. M. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In *KDD*, pages 359–363, 1998.
[80] C. Wu, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao, and S. Rao. Towards a learning optimizer for shared clouds. *PVLDB*, 12(3):210–222, 2018.
[81] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD 2012*, pages 97–108, 2012.
[82] Z. Yang, E. Liang, A. Kamsetty, C. Wu, and et al. Deep unsupervised cardinality estimation. *VLDB*, 13(3):279–292, 2019.
[83] 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.
[84] H. Yuan, G. Li, L. Feng, and et al. Automatic view generation with deep learning and reinforcement learning. In *ICDE*, pages 1501–1512, 2020.
[85] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. In *SIGMOD 2014*, pages 265–276, 2014.
[86] C. Zhang, R. Marcus, A. Kleiman, and O. Papaemmanouil. Buffer pool aware query scheduling via deep reinforcement learning. *CoRR*, abs/2007.10568, 2020.
[87] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *SIGMOD 2019*, pages 415–432, 2019.
[88] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *Proc. VLDB Endow.*, 10(5):541–552, 2017.
[89] X. Zhou, C. Chai, G. Li, and J. Sun. Database meets artificial intelligence: A survey. *TKDE*, 2020.
[90] X. Zhou, J. Sun, G. Li, and J. Feng. Query performance prediction for concurrent queries using graph embedding. *VLDB*, 13(9):1416–1428, 2020.