XuanYuan: An AI-Native Database

Guoliang Li, Xuanhe Zhou, Sihao Li
Department of Computer Science,Tsinghua University, Beijing, China
Gauss Database Group, Huawei Company
liguoliang@tsinghua.edu.cn

Abstract

In big data era, database systems face three challenges. Firstly, the traditional empirical optimization techniques (e.g., cost estimation, join order selection, knob tuning) cannot meet the high-performance requirement for large-scale data, various applications and diversified users. We need to design learning-based techniques to make database more intelligent. Secondly, many database applications require to use AI algorithms, e.g., image search in database. We can embed AI algorithms into database, utilize database techniques to accelerate AI algorithms, and provide AI capability inside databases. Thirdly, traditional databases focus on using general hardware (e.g., CPU), but cannot fully utilize new hardware (e.g., ARM, GPU, AI chips). Moreover, besides relational model, we can utilize tensor model to accelerate AI operations. Thus, we need to design new techniques to make full use of new hardware.

To address these challenges, we design an AI-native database. On one hand, we integrate AI techniques into databases to provide self-configuring, self-optimizing, self-monitoring, self-diagnosis, self-healing, self-assembling, and self-security capabilities. On the other hand, we enable databases to provide AI capabilities using declarative languages in order to lower the barrier of using AI.

In this paper, we introduce five levels of AI-native databases and provide several open challenges of designing an AI-native database. We also take autonomous database knob tuning, deep reinforcement learning based optimizer, machine-learning based cardinality estimation, and autonomous index/view advisor as examples to showcase the superiority of AI-native databases.

1 INTRODUCTION

Databases have played a very important role in many applications and been widely deployed in many fields. Over the past fifty years, databases have undergone three main revolutions.

The first generation is stand-alone databases, which address the problems of data storage, data management and query processing [2]. The representative systems include PostgreSQL and MySQL.

The second generation is cluster databases, which aim to provide high availability and reliability for critical business applications. The representative systems include Oracle RAC, DB2 and SQL server.

The third generation is distributed databases (and cloud-native databases), which aim to address the problems of elastic computing and dynamic data migration in the era of big data [3]. The representative systems include

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Aurora [1] and GaussDB [2].

However, the traditional databases still have several limitations in the big data era, due to the large-scale data, various applications/users and diversified computing power.

(1) Traditional database design is still based on empirical methodologies and specifications, and require heavy human involvement (e.g., DBAs) to tune and maintain the databases. We use several examples to show that databases can be improved using AI techniques. First, databases have hundreds of knobs and it requires DBAs to tune the knobs to adapt to different scenarios. Recently the database committee attempts to utilize machine learning techniques [1, 9, 20] to automatically tune the knobs, which can achieve better results than DBAs. Second, database optimizer relies on cost and cardinality estimation but traditional techniques cannot provide accurate estimation. Recently deep learning based techniques [6, 14] are proposed to estimate the cost and cardinality which also achieve better results. Moreover, learning-based optimizers [8, 12], learning-based index recommendation [15], learning-based automatic view generation [10] provide alternative optimization opportunities for database design. Third, traditional databases are designed by database architects based on their experiences. Recently some learning-based self-designed techniques are proposed, e.g., learned indexes [7] and learned NoSQL database design [5]. Thus we can utilize AI techniques to enhance databases and make databases more intelligent [18, 17].

(2) Traditional databases focus on *relational model* and provide relational data management and analysis ability. However, in the big data era, there are more and more diverse data (e.g., graph data, time-series data, spatial data, array data) and applications (e.g., machine learning and graph computing). It calls for a new database system that can integrate multiple models (e.g., relational model, graph model, tensor model) to support diversified applications (e.g., relational data analysis, graph computing and machine learning). Moreover, we can embed AI algorithms into databases, design in-database machine learning frameworks, utilize database techniques to accelerate AI algorithms, and provide AI capability inside databases.

(3) Transitional databases only consider general-purpose hardware, e.g., CPU, RAM and disk, but cannot make full use of new hardware, e.g., ARM, AI chips, GPU, FPGA, NVM, RDMA. It calls for a heterogeneous computing framework that can efficiently utilize diversified computing powers to support data management, data analysis, and in-database machine learning.

To address these problems, we propose an AI-native database (XuanYuan), which not only integrates AI techniques into database to make database more intelligent but also provides in-database AI capabilities. In particular, on one hand, XuanYuan integrates AI techniques into databases to provide self-configuring, self-optimizing, self-monitoring, self-diagnosis, self-healing, self-security and self-assembling capabilities for databases, which can improve the database's availability, performance and stability, and reduce the burden of intensive human involvement. On the other hand, XuanYuan enables databases to provide AI capabilities using declarative languages, in order to lower the barrier of using AI. Moreover, XuanYuan also fully utilizes diversified computing power to support data analysis and machine learning.

An AI-native database can be divided into five stages. The first is AI-advised database, which takes an AI engine as a plug-in service and provides offline database suggestions, e.g., offline index advisor, offline knob tuning. The second stage is AI-assisted database, which takes an AI engine as a built-in service and provides online monitoring and suggestions, e.g., online statistics collection, online database state monitoring, and online diagnosis. The third is AI-enhanced database. One one hand, it provides AI based database components, e.g., learned index, learned optimizer, learned cost estimation, learned storage layout. On the other hand, it provides in-database AI algorithms and accelerators. The fourth is AI-assembled database, which provides multiple data models (e.g., relational model, graph model, tensor model) and fully utilizes the new hardware to support heterogeneous computing. It can provide multiple options for each component, e.g., learned optimizer, cost-based optimizer, and rule-based optimizer, and thus can automatically assemble the components to form a database in

[1] https://aws.amazon.com/cn/rds/aurora/

[2] https://e.huawei.com/en/solutions/cloud-computing/big-data/gaussdb-distributed-database

Table 3: Five levels of AI-native database

Level	Feature	Description	Example
1	AI-advised	Plug-in AI engine	○ Workload Management (e.g., workload scheduling) ○ SQL Optimization (e.g., SQL rewriter, index/view advisor) ○ Database Monitor (e.g., knob tuner, system statistics) ○ Database Security (e.g., autonomous auditing/masking)
2	AI-assisted	Built-in AI engine	• Self-configuring (e.g., online knob tuning) • Self-optimizing (e.g., SQL optimization, data storage) • Self-healing (e.g., fault recovery, live migration) • Self-diagnosis (e.g., hardware/software error) • Self-monitoring (e.g., monitor workload/system state) • Self-security (e.g., tractable, encryption, anti-tamper)
3	AI-enhanced	Hybrid DB&AI engine	○ Learning-based Database Component • Learning-based rewriter • Learning-based cost estimator • Learning-based optimizer • Learning-based executor • Learning-based storage engine • Learning-based index ○ Declarative AI (UDF; view; model-free; problem-free)
4	AI-assembled	Heterogeneous processing	○ Self-assembling ○ Support new hardware (e.g., ARM, GPU, NPU)
5	AI-designed	The life cycle is AI-based	Design, coding, evaluation, monitor, and maintenance

order to achieve the best performance for different scenarios. This is similar to AlphaGO, which can explore more optimization spaces than humans. The fifth is AI-designed database, which integrates AI into the life cycle of database design, development, evaluation, and maintenance, which provides the best performance for every scenario.

In this paper, we first present the details of AI-native databases and then provide the research challenges and opportunities for designing an AI-native database.

2 AI-Native Database

We present the design of AI-native databases and Figure 1 shows the architecture. Next we discuss the five levels of AI-native databases as shown in Table 3.

2.1 Level 1: AI-Advised Database

The first level, AI-advised databases, provides offline optimization of the database through automatic suggestions [1, 9, 13]. The plugged-in AI engine is loosely coupled with databases. Limited by available resources, the AI engine mainly provides auxiliary tools from four aspects.

Workload Management. AI-based models can be used to control the workload from three aspects. First, AI-based models can benefit workload modeling. Directly modeling a workload with independent features (e.g., tables, columns, predicates) may lead to great information loss, such as the reference correlations among different tables. So instead we use an encoder-decoder model to learn an abstract representation of user workloads, which can reflect the correlation among the basic features. Second, AI-based models can be used for workload scheduling. Considering hybrid OLAP and OLTP workloads, AI-based models can estimate the required resources (e.g., CPU,

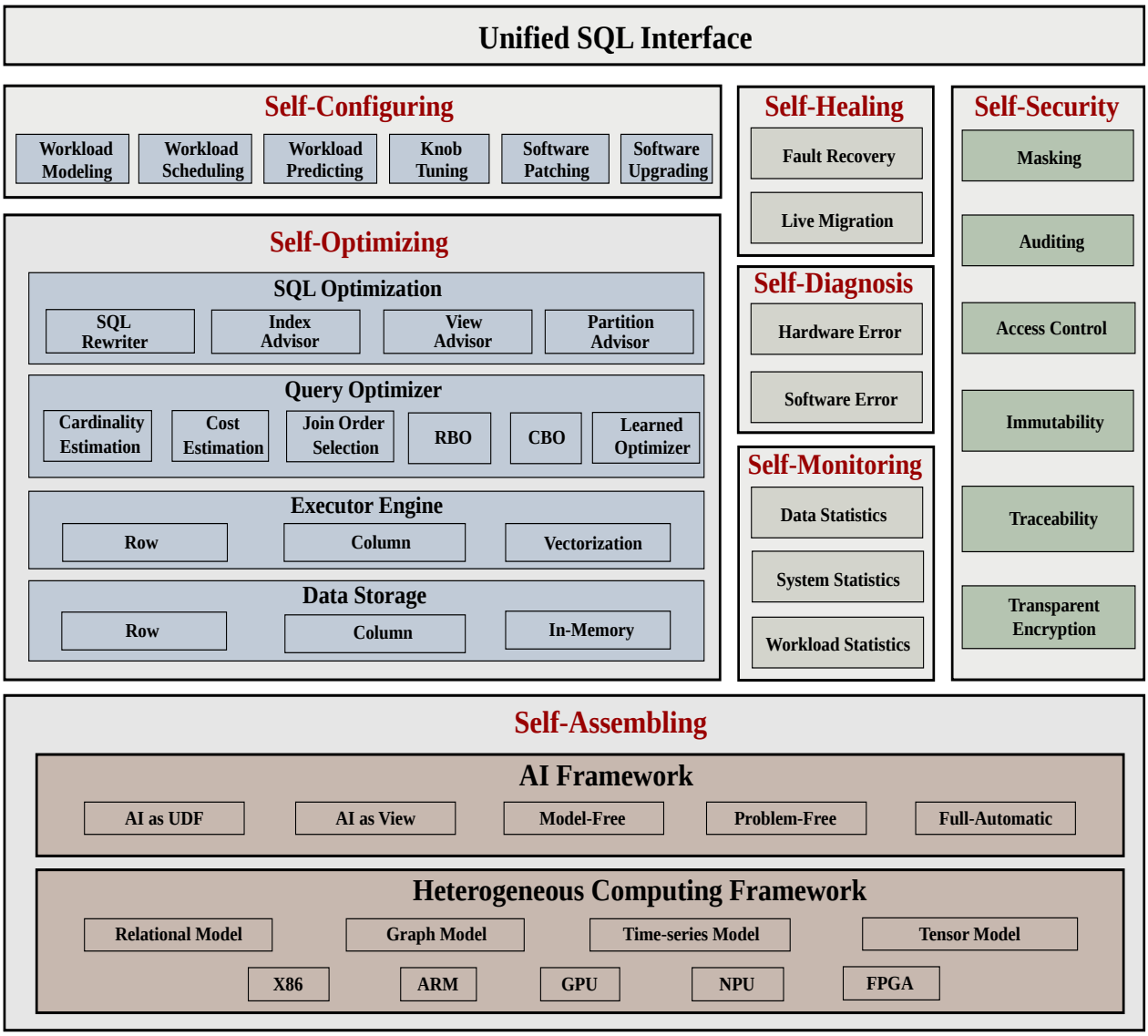

Figure 1: The architecture of AI-Native databases

RAM, DISK) and running time of each query. Then AI-based models can prioritize the workload, assign a high priority for the mission-critical queries, and defer the execution of queries that require heavy resources. Moreover, resource scheduling depends heavily on some parameters related to resource control, e.g., maximum concurrent IO throughput. But these parameters are static and need to be manually configured. Reinforcement learning can be used to learn relations between database (physical/logical) states, workload and database resources, and provides a reasonable and robust scheduling mechanism. Third, AI-based models can be used for workload prediction. We can predict the possible workloads in order to adapt to future workloads. Traditional workload prediction methods rely on database experts based on statistical data, which can not guarantee high accuracy. Instead, machine learning methods for workload forecasting [11] have better adaptability to different workloads.

SQL Optimization. It optimizes SQL queries from the following aspects. First, *SQL rewriter* can rewrite the poor SQL queries issued by ordinary users into well-formulated queries in order to improve the performance. Traditional methods rely on DBAs, which are impractical for heavy workloads. So AI-based methods can provide a rewriting tool to learn the principles of SQL writing (e.g., avoiding full table scanning, selecting indexed

columns for joins) and optimize the SQL structure. Second, *index advisor* can be optimized by AI-based methods. Database indexes are very important to improve the efficiency of SQL queries on complex data sets [4]. However, the traditional methods build the indexes based on DBAs' experiences, which cannot scale to thousands of tables. AI-based models can learn the benefit and cost to build an index given a query workload, and then automatically recommend the index based on the learned information. Third, *view advisor* can be optimized by AI-based models. Given a set of queries, we can first extract the equivalent sub-queries, select the sub-queries with high frequency, and learn the benefit and cost of building views on the sub-queries. Then materialized views can be automatically recommended by AI-based models. Fourth, *partition advisor* can be optimized by AI-based models. Traditionally the partitions are generated based on some primary keys or manually specified keys by DBAs. However, these methods may not find the best partition strategy. Instead, the AI-based models can learn the possible distribution of the partitions, estimate the benefits for different workloads, and recommend high-quality partitions.

Database Monitor. It monitors the database states, tunes database configuration and avoids database fail-overs. First, for *data statistics*, it automatically monitors the access frequency on table columns, data updates, and data distribution among different shards. Second, for *system statistics*, it automatically monitors the status of the database system (e.g., the number of batch requests per second, the number of user connections, network transmission efficiency). Then it analyzes those indicators using machine learning algorithms, tunes system parameters and provides early warnings for abnormal events. Third, for *workload statistics*, it monitors the performance of user workload and profiles on how workload varies. It can also predict future workloads.

DB Security. It combines security and cryptography technology with AI techniques to enhance the security of databases. First, *autonomous masking* aims to hide privacy data such as ID number. Autonomous masking judiciously selects which columns to be masked based on historical data and user-access patterns. Second, *autonomous auditing* optimizes the auditing from two aspects: data pre-processing and dynamic analysis. Traditional auditing often requires auditors to obtain a large number of business data, which is hard to obtain. Autonomous auditing not only saves manpower cost, but also helps auditors to make better decisions by providing useful information from massive data. Third, *autonomous access control* can automatically detect system vulnerabilities. Existing autonomous detecting methods are mainly to retrieve through security scanning, but cannot detect unknown security vulnerabilities. AI-based models can discover security vulnerabilities [22], which not only detect the most known vulnerabilities in a vulnerability database, but also predict and evaluate potential vulnerabilities.

2.2 Level 2: AI-Assisted Database

The second level, AI-assisted databases, integrates an AI engine into the database kernel for run-time optimization. AI components (e.g., tuning model, workload scheduling, view advisor) can be merged into the corresponding database components [10]. In this way, AI capabilities are integrated into the working procedure of the database. For example, if embedding the tuning model into the query optimizer, we can first conduct query tuning for each query (e.g., tuning user-level parameters to better adapt to the query features), and then normally generate and execute the query plan. The advantage of AI-assisted databases is that 1) it can provide more fine-grained optimization; 2) it can reduce more overhead by embedding an AI engine into the kernel.

Moreover, built-in AI engine can provide self-configuring, self-optimizing, self-monitoring, self-healing, self-diagnosis, and self-security services.

Self-configuring. Database can automatically tune their own configuration to adapt to changes in workload and environment. First, the workload can be self-configured. Database can execute queries in different granularities in parallel, each of which has various requirement of system resources and performance. Database can configure the workload based on the workload features, which are obtained by conducting workload modeling, scheduling and predicting. Second, databases include several configuration mechanisms, such as knob tuning, software patching, software upgrading and etc. For example, all databases have hundreds of tunable knobs, which are

vital to nearly every aspect of database maintenance, such as performance, availability, robustness, etc. However, these configurations require to be tuned manually, which not only is time consuming but also cannot find optimal configurations. AI based methods, e.g., deep reinforcement learning, can automatically tune the database knobs. Moreover, other configurations (e.g., software bugs, partition scheme) can also be optimized by AI-based methods.
Self-optimizing. First, we can design a learning-based query optimizer in cost/cardinality estimation, join order selection and data structures. 1) AI techniques can optimize cost/cardinality estimation, which is vital to query plan selection. However, database mainly estimates cardinality based on raw statistics (e.g., number of distinct values, histograms) and is poor in estimating the resulting row number of each query operator (e.g., hash join, aggregate, filter), by using histograms. AI-based methods, e.g., Tree-LSTM, can learn data distribution in depth and provide more accurate cost/cardinality estimation. 2) AI techniques can optimize join order selection. Different join schemes have a great impact on query performance and finding the best plan is an NP-hard problem. With static algorithms (e.g., dynamic programming, heuristics algorithm), the performance of join order selection in databases is limited by the quality of the estimator. AI-based methods can better choose between different join order plans by taking one-step join as the short-term reward and the execution time as the long-term reward. 3) Data partitions, indexes, and views can also be online recommended by built-in AI models. Second, we can utilize learning-based models to optimize executor engine and data storage. For executor engine, we consider two aspects. 1) We provide hybrid query executing methods such as row-based and column-based. Here AI can serve different data applications (e.g., row-based for OLTP, column-based for OLAP). 2) We provide tensor processing engine to execute AI models. To enhance AI techniques to optimize database components, databases can execute AI models natively with a vectorization engine.
Self-monitoring. Database can automatically monitor database states (e.g., read/write blocks, concurrency state, working transactions) and detect operation rules, e.g., root cause analysis rules. It can monitor database states (e.g., data consistency, DB health) in the life-cycle. The monitored information can be used by self-configuring, self-optimizing, self-diagnosis, and self-healing. Note that it may have some overhead to monitor the database states and we need to minimize the overhead for monitoring database states.
Self-diagnosis. Self-diagnosis includes a set of strategies to diagnose and correct abnormal conditions in databases, which are mostly caused by errors in hardware (e.g., I/O error, CPU error) and software (e.g., bugs, exceptions). Self-diagnosis helps to guarantee services even if some database nodes work unexpectedly. For example, in case of data-access error, memory overflow or violation of some integrity restrictions, database can automatically detect the root causes using the monitored states and thus cancel the corresponding transactions in time.
Self-healing. Databases can automatically detect and recover from database problems (e.g., poor performance, hardware/software fail-overs). First, it isolates different sessions or users and avoids affecting others users when encountering errors. Second, it adopts AI-advised database tools to reduce the recovery time and saves humans from the failure-recovery loop. Third, it can kill some abnormal queries that take too many resources.
Self-security. Self-security includes several features. First, the data in the life-cycle (including storage, memory and CPU) are always encrypted, which should be unreadable for the third party. Second, the data access records should be tractable in order to get the access history of the data. Third, the data should be tamper-proofed in order to prevent malicious modification of data. Fourth, AI-based models can be used to automatically learn the attacking rules and prevent the unauthorized access and attack patterns. Fifth, it can automatically detect sensitive data using AI-based models.

2.3 Level 3: AI-Enhanced Database

The third level, AI-enhanced database, not only uses AI techniques to improve the database design but also provides in-database AI capabilities.

2.3.1 Learning-based Database Components

Most of database core components are designed by humans based on their experiences, e.g., optimizer, cost estimation, index. However, we find that many components can be designed by AI-based models. First, the traditional indexes, e.g., B-tree, R-tree, can be designed by AI-based models. For example, learned indexes are verified that they can reduce the index size and improve the query performance [7]. Second, the cost/cardinality estimation can be optimized by deep learning, because the empirical methods cannot capture the correlations among different tables/columns while deep learning can capture more information using deep neural networks. Third, the join order selection problem is an NP-hard problem and traditional heuristics methods cannot find the best plan; while the deep reinforcement learning techniques can learn more information and get better plan. Fourth, query optimizer replies on cost/cardinality estimation, indexes, join order selection, etc, and an end-to-end learning based optimizer is also promising.

Thus many database components can be enhanced by AI-based methods, which can provide alternative strategies beyond traditional empirical techniques.

2.3.2 In-Database AI Capabilities

Although AI can address many real-world problems, there is no widely deployed AI systems that can be used in many different fields, because AI is hard to be used by ordinary users. Thus we can borrow database techniques to lower the barrier of using AI. First, SQL is easy to be used and widely accepted, and we can also extend SQL to support AI. Second, we can utilize database optimization techniques to accelerate AI algorithms, e.g., indexing, incremental computing, and sharing computations.

We categorize the techniques of supporting AI capabilities in database in five levels.

AI Models As UDFs. We embed AI frameworks (e.g., MADlib, TensorFlow, Scikit-learn) in database and provide user-defined functions (UDFs) or stored procedures (SPs) for each algorithm. Then users can call UDFs or SPs from databases to use AI algorithms.

AI Models As Views. If a user wants to use AI algorithms (e.g., random forests) in the first level, the user requires to first train the model and then use the trained model. In the second level, we can take an AI algorithm as a view, which is shared by multiple users. If an algorithm is used by a user, we can materialize the model and other users can directly use the model. The model can also be updated by incrementally training.

Model-free AI. In the first and second levels, the user must specify the concert algorithms (e.g., k-means for clustering). Actually, users may only know which problems should be addressed, e.g., clustering or classification, but do not know which algorithms should be used. In this way, database can automatically recommend the algorithms that best fit the user scenarios.

Problem-free AI. The users even cannot specify the problems that require to be addressed, e.g., classification and clustering. Given the database, the problem-free AI can automatically find which problems can be addressed by AI algorithms and recommend suitable AI algorithms.

Full-automatic. The system can automatically discover AI opportunities, including discovering the problems, the models, the algorithms, relevant data, and training methods.

2.3.3 Hybrid AI and DB Engine

The above methods still use two engines: AI engine and DB engine. It calls for a hybrid AI and DB engine that provides both AI and DB functionalities. Then given a query, the SQL parser parses the SQL query and produces a general-purpose query plan. Based on the operators in the plan, it decides whether it utilizes relational data models or utilizes AI models. For relational data models, the query plan is sent to the database executor; otherwise, it is sent to the AI executor. Moreover, it also calls for new models to support both relational algebra and tensor models, and in this way, we can utilize a unified model to support both AI and database.

2.4 Level 4: AI-Assembled Database

The fourth level, AI-assembled database, not only automatically assembles database components to generate a database that can best fit a given scenario, but also schedules the tasks to diversified hardware.

Self-assembling. First, each database component has multiple options. For example, optimizers include cost-based model, rule-based model, and learned-based model. We first take each component variant as a service. Then we select the best components based on users' requirements. Note that different variants of the same component should adopt the same standard interface such that the components can be assembled.

Thus we propose a self-assembling module. For different scenarios, we can dynamically select the appropriate components in each layer of service and assemble the appropriate execution path. The execution path can be seen as a natural language sequence (NLS), such as $\langle SQL_i$, parser_pg, optimizer_RBO, storage row, accelerator $\rangle$, in which each position has only discrete token options. So the problem is how to generate NLS in the query level. One possible method is to use reinforcement learning (RL) algorithm. It takes the whole path sequence as an episode and a single action as an epoch. Under each epoch, the agent chooses the next component (action), which executes the query, leading to a state transition (e.g., the query status changes into syntax tree). In this problem, action is discrete, so DDQN algorithm [16] can be used. Compared with other RL algorithms with discrete outputs, DDQN eliminates the problem of overestimation (deviate greatly from the optimal solution) by choosing decoupling actions and calculating the target Q value.

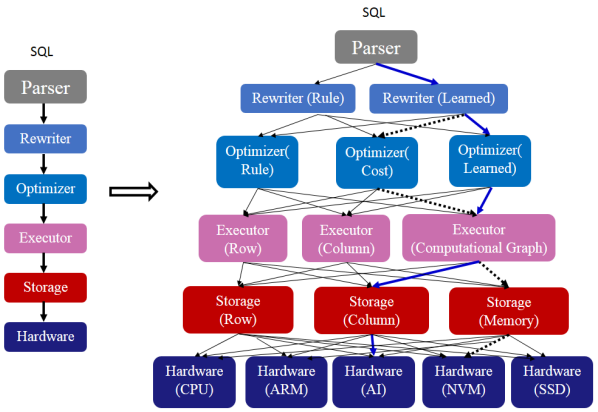

Figure 2: AI-Assembled Database

However, the RL-based routing algorithm has two problems. Firstly, Q network will not generate scores until the last node is generated. It is insensitive to the choice of intermediate nodes. However, for the entire path, it is necessary to give an action a comprehensive score on current and future impacts. Secondly, when training with epoch as a unit, each node of a path is scattered in a training sample, instead of being used as a whole to calculate the gradient.

Generative Adversarial Network (GAN) [21] can better solve those problems of end-to-end path selection. We use G network to generate path vectors based on the workload, database state and component characteristics, and we use D network as the performance model. But the traditional GAN network model is not fully applicable to our problem. Because it is mainly used to generate data with continuous range of values, and it is difficult to generate path vectors with discrete tokens. That is because G-networks need to be fine-tuned based on gradient descent and regress towards the expectation. But when the data is discrete tokens, fine-tuning is often meaningless. So we choose to combine RL algorithm with GAN [19]. Firstly, G network is used as an agent in the RL algorithm: action is the next service node; state includes not only query status, database status, but also generated node information. Each iteration generates a complete path. Secondly, unlike the traditional RL algorithm, each action is scored by D network to guide the generation of the whole path sequence.

Heterogeneous Computing. We need to make full use of diversified computing power. Note that DB and AI usually require different computing power and hardware. For DB, traditional optimizer processes queries with CPU. While AI requires new AI chips to support parallel processing (e.g., GPU and NPU) and self-scheduling. And now many applications need to use both DB and AI techniques, especially in large data analysis scenarios. Thus we need to support multiple models, e.g., relation model, graph model, stream model and tensor model. We can automatically select which models should be used. We also need to be able to switch computing powers based on the models. For example, for the tuning module in the optimizer, when training the tuning model, we

fetch training data into memory with AI chips and conduct backward propagation (training the neural network) with NPU. For the join and selection operations, we may still use the traditional hardware. We also need to study whether a model (e.g., relational model) can be transformed into other models (e.g., tensor model). The ultimate objective is to fully unleash the power of diversified computing, including x86, ARM, GPU, NPU. We aim to continuously push our AI strategy forward and foster a complete computing ecosystem.

2.5 Level 5: AI-Designed Database

The fifth level, AI-designed databases, is fully designed by AI, including design, coding, evaluation, monitoring and maintenance. In this way, AI techniques are integrated into the whole database life cycle so as to achieve the best performance for both DB and AI.

3 Challenges and Opportunities

It brings new research challenges and opportunities to design an AI-native database, which aims to support data management, data analysis, machine learning together in the same system.

3.1 From one-size-doesn't-fit-all to one-stack-fits-all

Michael Stonebraker argues that one size does not fit all, due to various applications (e.g., OLTP, OLAP, stream, graph) and diversified hardware (e.g., CPU, ARM, GPU, FPGA, NVM). Note that the database components and their variants are limited, but the number of possible combinations for these components to assemble a database is huge. So the database architects design the database architectures by combining different variants of techniques based on their empirical rules and experience. Thus these human-designed databases may not be optimal because they may fall into a local optimum. It calls for automatically designing a database using AI techniques, which can adapt to different scenarios.

We argue that one stack fits all. The basic idea is to first implement the database components, e.g., indexes, optimizers, storage, where each component has multiple variants/options, then use AI techniques to assemble these components to form some database candidates, and finally select a database that best suits a given scenario. In this way, we can automatically verify different possible databases (i.e., different combinations of components), explore many more possible database designs than human-based deign, and could design more powerful databases. This is similar to AlphaGO, where the learning-based method beats humans, because the machines can explore more unknown spaces.

There are several challenges in one-stack-fits-all. First, each component should provide standard interfaces such that different components can be integrated together. Second, each component should have different variants or implementations, e.g., different indexes, different optimizers. Third, it calls for a learning-based component to assemble different components. Fourth, the assembled database can be evaluated and verified before the database is deployed in real applications. Fifth, each component should be run on different hardware, e.g., learned optimizers should be run on AI chips and traditional cost-based optimizers should be run on general-purpose chips. It calls for effective methods to schedule the tasks.

3.2 Next Generation Analytic Processing: OLAP 2.0

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. We think that hybrid DB and AI online analytic processing on multi-model data should be the next generation OLAP, i.e., *OLAP 2.0*.

There are several challenges in supporting OLAP 2.0. First, different data types use different models, e.g., relational model, graph model, KV model, tensor model, and it calls for a new model to support multi-model data analytics. Second, OLAP 2.0 queries may involve both database operations and AI operations, and it needs to design new optimization model to optimize these heterogeneous operations across different hardware.

3.3 Next Generation Transaction Processing: OLTP 2.0

Traditional OLTP mainly uses general-purpose hardware, e.g., CPU, RAM and Disk, but cannot make full use of new hardware, e.g., AI chips, RDMA, and NVM. Actually, we can utilize new hardware to improve transaction processing. First, based on the characteristics of NVM, including non-volatile, read-write asymmetry speed, and wear-leveling, we need to reconsider the database architecture. For example, we can utilize NVM to replace RAM and replace page-level storage with record-level storage on NVM. Second, we can utilize RDMA to improve the data transmission in databases. Moreover, we can use the programmable feature of intelligent Ethernet card to enable filtering on RDMA and avoid unnecessary processing in RAM and CPU. Third, there are some AI chips which are specially designed hardware, and it is also promising to design database-oriented chips that are specially defined hardware for databases.

There are several challenges in supporting OLTP 2.0. First, it is challenging to fully utilize new hardware to design a new generation database. Second, it is hard to evaluate and verify whether the new hardware can benefit the database architecture. Third, it calls for an effective tool to automatically evaluate a feature or a component (even a database).

3.4 AI4DB

There are several challenges that embed AI capabilities in databases.

Training Samples. Most AI models require large-scale, high-quality, diversified training data to achieve good performance. However, it is rather hard to get training data in databases, because the data either is security critical or relies on DBAs. For example, in database knob tuning, the training samples should be gotten based on DBAs' experiences. Thus it is hard to get a large number of training samples. Moreover, the training data should cover different scenarios, different hardware environments, and different workloads.

Model Selection. There are lots of machine learning algorithms and it is hard to automatically select an appropriate algorithm for different scenarios. Moreover, the model selection is affected by many factors, e.g., quality, training time, adaptability, generalization. For example, deep learning may be a better choice for cost estimation while reinforcement learning may be a better choice for join order selection. The training time may also be important, because some applications are performance critical and cannot tolerate long training time.

Model Convergence. It is very important that whether the model can be converged. If the model cannot be converged, we need to provide alternative ways to avoid making bad decisions. For example, in knob tuning, if the model is not converged, we cannot utilize the model for knob suggestion.

Adaptability. The model should be adapted to different scenarios. For example, if the hardware environments are changed, the model can adapt to the new hardware.

Generalization. The model should adapt to different database settings. For example, if the workloads are changed, the model should support the new workloads. If the data are updated, the model should be generalized to support new data.

3.5 DB4AI

Accelerate AI algorithms using indexing techniques. Most of studies focus on the effectiveness of AI algorithms but do not pay much attention to the efficiency, which is also very important. It calls for utilizing database techniques to improve the performance of AI algorithms. 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.

Discover AI Models. Ordinary users may only know their requirements, e.g., using a classification algorithm to address a problem, but do not know which AI algorithms should be used. Thus it is important to automatically discover AI algorithms. Moreover, it is also challenging to reuse the well-trained AI models by different users.

3.6 Edge Computing Database

Most databases are designed to be deployed on servers. With the development of 5G and IOT devices, it calls for a tiny database embedded in small devices. There are several challenges in designing such a tiny database. The first is database security to protect the data. The second is real-time data processing. The small device has low computing power, and it is rather challenging to provide high performance on such small devices. The third is data migration among different devices. Some devices have small storage and it is challenging to migrate the data across different devices.

4 CONCLUSION

We proposed an AI-native database XuanYuan, which not only utilizes AI techniques to enable self-configuring, self-optimizing, self-monitoring, self-healing, self-diagnosis, self-security, and self-assembling, but also provides in-database AI capabilities to lower the burden of using AI. We categorized AI-native databases into five levels, AI-advised, AI-assisted, AI-enhanced, AI-assembled, and AI-designed. We also discussed the research challenges and provided opportunities in designing an AI-native database.

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*, pages 1009–1024, 2017.

[2] C. J. Date. *An introduction to database systems (7. ed.)*. Addison-Wesley-Longman, 2000.

[3] G. Figueiredo, V. Braganholo, and M. Mattoso. Processing queries over distributed XML databases. *JIDM*, 1(3):455–470, 2010.

[4] A. Gani, A. Siddiqa, S. Shamshirband, and F. H. Nasaruddin. A survey on indexing techniques for big data: taxonomy and performance evaluation. *Knowl. Inf. Syst.*, 46(2):241–284, 2016.

[5] 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.

[6] 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.

[7] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.

[8] S. Krishnan, Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning. *CoRR*, abs/1808.03196, 2018.

[9] G. Li, X. Zhou, B. Gao, and S. Li. Qtune: A query-aware database tuning system with deep reinforcement learning. In *VLDB*, 2019.

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

[11] 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 *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 631–645, 2018.

[12] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 3:1–3:4, 2018.

[13] M. Mitzenmacher. A model for learned bloom filters and related structures. *CoRR*, abs/1802.00884, 2018.

[14] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. In *SIGMOD*, pages 4:1–4:4, 2018.

[15] W. G. Pedrozo, J. C. Nievola, and D. C. Ribeiro. An adaptive approach for index tuning with learning classifier systems on hybrid storage environments. In *HAIS*, pages 716–729, 2018.

[16] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[17] W. Wang, G. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K. Tan, and S. Wang. SINGA: putting deep learning in the hands of multimedia users. In *SIGMM*, pages 25–34, 2015.

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

[19] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[20] J. Zhang, Y. Liu, K. Zhou, and G. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *SIGMOD*, 2019.

[21] Z. Zhou, J. Liang, Y. Song, L. Yu, H. Wang, W. Zhang, Y. Yu, and Z. Zhang. Lipschitz generative adversarial nets. *CoRR*, abs/1902.05687, 2019.

[22] S. Zong, A. Ritter, G. Mueller, and E. Wright. Analyzing the perceived severity of cybersecurity threats reported on social media. *CoRR*, abs/1902.10680, 2019.