Front. Comput. Sci., 2025, 19(9): 199608
https://doi.org/10.1007/s11704-024-40624-2

RESEARCH ARTICLE

GaussDB-AISQL: a composable cloud-native SQL system with AI capabilities

Cheng CHEN[1], Wenlong MA[2], Congli GAO[2], Wenliang ZHANG[2], Kai ZENG[2], Tao YE[2], Yueguo CHEN (✉)[3], Xiaoyong DU[1]

1 School of Information, Renmin University of China, Beijing 100872, China
2 Huawei Cloud, Beijing 100085, China
3 Key Laboratory of Data Engineering and Knowledge Engineering (Ministry of Education), Renmin University of China, Beijing 100872, China

Abstract Cloud-native data warehouses have revolutionized data analysis by enabling elasticity, high availability and lower costs. And the increasing popularity of artificial intelligence (AI) drives data warehouses to provide predictive analytics besides the existing descriptive analytics. Consequently, more vendors start to support training and inference of AI models in data warehouses, exploiting the benefits of near-data processing for fast model development and deployment. However, most of the existing solutions are limited by a complex syntax or slow data transportation across engines.

In this paper, we present GaussDB-AISQL, a composable SQL system with AI capabilities. GaussDB-AISQL adopts a composable system design that decouples computing, storage, caching, DB engine and AI engine. Our system offers all the functionality needed by end-to-end model training and inference during the model lifecycle. It also enjoys the simplicity and efficiency by providing a SQL-like syntax and removes the burden of manual model management. When training an AI model, GaussDB-AISQL benefits from highly parallel data transportation by concurrent data pulling from the distributed shared memory. The feature selection algorithms in GaussDB-AISQL make the training more data-efficient. When running model inference, GaussDB-AISQL registers the trained model object in the local data warehouse as a user-defined-function, which avoids moving inference data out of the data warehouse to an external AI engine. Experiments show that GaussDB-AISQL is up to 19× faster than baseline approaches.

Keywords database system, data management, OLAP, cloud computing, AI, machine learning

1 Introduction

Data warehouses have become a popular choice for online analytical processing. More and more cloud computing vendors are providing their data warehouses with elasticity, high availability and lower costs, e.g., Amazon Redshift, Google BigQuery, and Snowflake. Fueled by the growing need to utilize AI, predictive in-warehouse analytics emerge to empower data warehouse practitioners with machine learning capabilities. However, AI technology stack generally requires users to decide the model types, hyper-parameters, input features, etc. The training and inference data are manually moved back and forth between data warehouses and AI engines. In contrast, SQL is much simpler to perform declarative data analysis. Incorporating SQL with AI presents a paradigm shift with management of AI models in data warehouses, which empowers a broader audience of data warehouse practitioners to leverage AI capabilities. This shift has empowered data warehouse businesses to move beyond descriptive analytics, where past data is summarized, towards predictive analytics, where unseen labels are predicted. By leveraging the capabilities of these data warehouses, organizations can now harness the power of their data to anticipate trends, forecast outcomes, and make data-driven decisions that unlock new opportunities.

In addition to the democratization of AI for SQL analysts, SQL + AI technology minimizes data movement, ensures data integrity and streamlines the AI workflow with the following benefits.

- Reducing ETL process. AI application involves data extraction from the data warehouse and transferring to a dedicated AI engine, and then manually load the AI predictions back into the warehouse. This cumbersome process introduces performance bottlenecks and security concerns. SQL + AI eliminates this overhead by bringing the AI algorithms directly to the data within the data warehouse.
- Unlocking scalability and optimization. SQL + AI improves performance by processing data within the data warehouse, which is often highly optimized for storage and computing. This fosters enhanced scalability for handling large datasets compared to

Received June 23, 2024; accepted August 22, 2024

E-mail: chenyueguo@ruc.edu.cn

traditional standalone AI engines, reuses expertise in data warehouse management, enabling seamless integration with existing data pipelines.

- Ensuring data integrity and access control. Most AI frameworks just read and write data from file system, which can violate data integrity or consistency and leads to inaccurate results, operational problems, and security concerns. While data residing in data warehouses enjoys the integrity and consistency guaranteed by data warehouses. Keeping the training and inference data inside a data warehouse to interact with AI modules helps to avoid these problems.

With all these benefits provided by SQL + AI, data warehouses become the new platform to bridge the gap between AI and data storage system, which empowers the widespread adoption of AI techniques. Major cloud data warehouse vendors have all paved the path towards the integration of ML in the warehouse, including but not limited to Google BigQuery [1], Amazon Redshift [2], and Azure Synapse / Raven [3]. BigQuery ML [1] allows training inside the warehouse for some simpler models. Raven [3] pushes inference deeply in the warehouse. Redshift [2] also allows inference through SQL in a loosely integrated fashion. However, existing SQL + AI solutions still face the following key challenges.

1. **Unoptimized or monolithic integration of components.** Some systems [2,4] directly connect SQL and AI components through a simple glue layer. The layer delegates the processing of AI commands embedded within SQL queries to an external AI module. But it is unoptimized w.r.t. resource allocation and data transportation. Other works deeply integrate AI operators from linear algebra [5,6] to more complex neural network operators [7] into SQL systems. Their performance is usually better than the previous approach. However, monolithic integration suffers from limited reuse between systems, leading to reinvention of the wheels for developers.
2. **A steep learning curve caused by complicated syntax and configuration.** The complexity of SQL syntax tailored for AI computations poses challenges for non-technical users. Choosing a suitable model type and tuning with optimal parameters require deep understanding of AI models. Some systems [8,9] demand manual feature engineering before training, which is not only labor-intensive but also susceptible to human errors [10].
3. **Expensive data transfer in training and inference.** They often demand the conversion of data across various formats, leading to inefficiencies. A typical training process for machine learning models involves reading all the training data many times [11], which leaves great potential for improvement. In addition, sending data out of data warehouses to AI modules for inference [2,4] incurs substantial costs because a large amount of AI workloads are inference [12].

To address these challenges, we present GaussDB-AISQL, a composable SQL system with the capabilities of training AI models and running inference on AI models. GaussDB-AISQL provides a SQL-like interface for data analysts that combines descriptive and predictive analysis. It extends the parser, the optimizer, and the executor of the data warehouse to support predictive analysis. GaussDB-AISQL is not only user-friendly and automated for predictive analytics, but also characterized by enhanced processing efficiency and scalability. Figure 1 shows an example application of our system. The Iris dataset describes three species of Iris flowers (setosa, virginica, and versicolor), characterized by four features: the lengths and widths of the sepals and petals. Fig. 1Ⓐ shows how to create the data table for Iris with our transparent model management mechanism. We simply add `PREDICT KEY` as the keyword to specify the target column of Iris. Fig. 1Ⓑ shows how to run model inference to get the predicted value of the target `species` column. The SQL statement in Fig. 1Ⓒ shows a more complex case where a user queries Iris records which have sepal lengths larger than five and the predicted species to be setosa. The detailed mechanism will be revealed in the next section

The main contributions of GaussDB-AISQL are summarized as follows.

1. **A composable system design and job-level resource management.** GaussDB-AISQL has job-level resource management and features in a composable system design. It is an end-to-end solution in data warehouse for the training and inference of AI models. It also decouples from the existing code implementations to improve maintainability and testability. The coordinator of GaussDB-AISQL framework extends that of the data warehouse parser. We only need to extend the SQL parser part to adapt GaussDB-AISQL to other data warehouses systems. The AI-extened SQL statement is converted to the common Substrait [13] intermediate representation (IR) format to fit many data warehouse engines. The coordinator optimizes the IR to get an efficient job-level execution plan.
2. **A complete set of model management functionalities with extremely simple-to-use triggers.** GaussDB-

```
— Transparent model training
Ⓐ CREATE TABLE iris (
    sepal_length FLOAT, sepal_width FLOAT,
    petal_length FLOAT, petal_width FLOAT,
    species char(20) PREDICT KEY);

Ⓑ — Transparent model inference
— Predicted column naming: <col_name>_p
SELECT species_p FROM iris;

— Using the predicted column
Ⓒ SELECT sepal_length, species_p FROM iris WHERE
sepal_length > 5 AND species_p = 'setosa';
```

Fig. 1 AISQL's model-transparent predictive analytics over the Iris example. Users don't need to manually create the model so it is extremely easy-to-use. To the best of our knowledge, AISQL is the first system to provide such kind of transparent SQL interface for running AI in SQL

AISQL implicitly manages the entire model lifecycle and is transparent to users. It minimizes user coding effort by requiring only minimal parameter configuration and automating the entire model training, saving, registration, inference, and version management process. GaussDB-AISQL reuses existing data warehouse syntax, workflow, and frameworks as much as possible to reduce user learning and migration costs. The training process of an AI model is implicitly triggered by just specifying the target column when creating a table (Fig. 1Ⓐ). The model predictions and existing labels can work together in a seamless way. Users have full control over the trained models, including the classical CRUD (create / read / update / delete) operations.

3. **Efficient, data-centric model training and inference.** GaussDB-AISQL enjoys a high speed for model training and inference by efficient data selection and parallel data transportation. GaussDB-AISQL selects only a necessary subset of the training features and data instances to achieve more efficient training. GaussDB-AISQL supports parallel access to training data, parallel execution of SQL and AI pipelines, and parallel inference via embedded user-defined functions (UDFs). The model inference runs inside of data warehouses, reducing the overhead of data transfer. GaussDB-AISQL utilizes the data warehouse's scale-out capability and UDF vectorization for batch inference acceleration, and achieves efficient data transfer between DB engine and AI engine to improve system performance. Experiment results show that GaussDB-AISQL is up to 19× faster than the baseline approaches.

In the next section, we describe the system overview of GaussDB-AISQL and its core components. Section 3 introduces the model management mechanism in the model lifecycle. Section 4 describes the data management for model training and inference, followed by evaluation results in Section 5. We review the related work in Section 6 and conclude in Section 7.

2 System overview

In this section, we discuss how to architect AISQL[1)] to achieve composable and efficient training and inference of AI or machine learning (ML)[2)] models in a cloud-native data warehouse. First Section 2.1 provides an overview of all the components. Then Section 2.2 describes the core query engine, including the parser, the optimizer, the scheduler, and the executor. The other subsections introduce the data warehouse, ML module, and storage components.

2.1 Architecture

Figure 2 gives an overview of all the components for computation and storage in AISQL. The parser translates the received AI-extended SQL into an IR in the composable

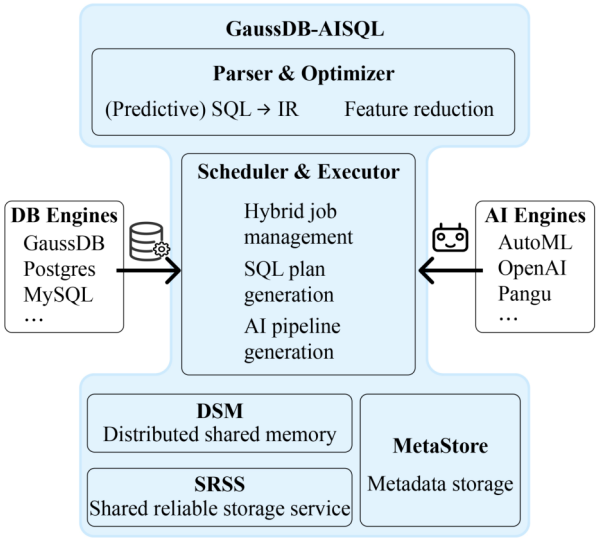


Fig. 2 The architecture of AISQL

Substrait [13] format. After the optimizer transforms the IR into a directed acyclic graph (DAG), the scheduler dispatches the SQL jobs and AI jobs to their separate engines. The executors locally execute query fragments of the DAG. The composable design of AISQL allows it to connect to many popular data warehouses or database engines and AI engines with only minor modification. Currently, we implement Huawei's cloud-native GaussDB data warehouse service (DWS) to plug into AISQL as a first prototype. Despite AISQL's flexibility, it also enjoys high efficiency for AI model training and inference due to the job-level resource scheduling and optimized data flow.

2.2 Query engine

2.2.1 Syntax & extended parser

We take the classical Iris dataset in Fig. 1 as an example to illustrate the working scenario of AISQL. After creation of the Iris table (Fig. 1Ⓐ), AISQL automatically triggers the model training process when loading data. AISQL automatically retrains the model every time when the growth of data volume exceeds a threshold or the distribution of the data shifts. The distribution shift is determined based on the same statistics used in data warehouse query optimization. AISQL tracks the cardinality for nominal data columns and the histograms for numerical data columns. Other indirect indicators are also considered, including changes of query execution time, query plan, and query selectivity. A sudden increase in query execution time or selectivity might be a symptom of outdated statistics, or increase in data volume. The generation of a query plan tree is based on more comprehensive statistics of the data other than cardinality or histogram. Hence we also compare the query plan tree to determine the re-training of AI models. In this way, we relieve the burden of manually re-training models from novice users and always provide freshly trained models. When running inference (Fig. 1Ⓑ), the predicted column is set as a hidden column in the Iris table.

1) We call GaussDB-AISQL as AISQL for conciseness.
2) Hereafter, we use the terms AI and ML interchangeably.

The naming rule for this predicted hidden column is appending _p to the original name of the target column. In the Iris case, it is `species_p`. The predicted column `species_p` contains the predicted value by the latest model. If the model is re-trained, then the values in `species_p` will also be updated. The model inference process is triggered implicitly by the `SELECT` clause over predicted column. AISQL fills the result of inference by the newest model into the predicted column. If the training is not completed, then AISQL will report the ongoing training status and abort the inference query. AISQL automatically infers the task type of classification or regression based on the cardinality of distinct values in the target column.

Table 1 shows three types of syntax for applying AI models in AISQL: Ⓐ Model-Oriented as the common way; Ⓑ Column-Oriented and Ⓒ Model-Transparent as our new way. Table 1 is based on the Iris example without loss of generality. The full set of model management functions are listed in the first column of Table 1. The third row in Table 1Ⓐ shows how to run model inference by calling the model name (`m1`) as a normal SQL function. Row four to row six show the version management of the models, including updating, activating, and deleting the model `m1`. In AISQL, users are NOT required to write these explicit SQL statements listed in Ⓐ Model-Oriented to run their predictive analytics. Unless they have special needs, such as activating an older version of the model, or deleting a model. This complicated set of SQL commands is often seen by prior SQL + AI systems, in which case users need to worry about updating the model according to data changes explicitly. If users want a model trained on the latest data regardless of our auto updating, they can also ALTER TABLE to add a new predicted column as shown in the first cell of Table 1Ⓑ. The training status (training / succeeded / failed) of the model can be checked by the statement in the second cell of Table 1Ⓑ. Users do not need to activate a version of the model in the column-oriented syntax because they achieve that by just specifying the name of the predicted column. When they have an already trained model outside of the data warehouse, they can register the model in PMML [14] or ONNX [15] format directly by just specifying the model path. For example, `REGISTER MODEL m2 LIKE iris WITH(TARGET = 'species', MODELPATH = 'm2.onnx');`.

Our extended parser deeply integrates with the data warehouse and shares operators with the data warehouse optimizer. The parser translates SQL statements into a unified intermediate representation (IR). We support join queries across databases due to the inter-operable Substrait [13] format of IR. AISQL executes the entire pipeline end-to-end using the statement-level IR, treating all sub-statements as part of a single transaction. AISQL optimizes the IR through scheduling, forming a DAG. The scheduler of AISQL then splits the DAG into jobs in the data warehouse engine and the AI engine for execution.

2.2.2 The IR optimizer

AISQL generates a parallelized statement-level DAG to support parallel scheduling. Based on the DAG, the collaborative optimizer divides the job into fragments to ensure the correct execution order of AI jobs and SQL jobs. Each fragment is efficiently processed by parallel threads, which improves the performance of the entire DAG task.

We introduce a new systematic method to reduce data transportation in AISQL. The core idea is to select only important features for training or inference. We define feature importance as how much each feature contributes to a model's predictions. The IR optimizer identifies the optimal subset of features for use in a machine learning model. This process involves removing irrelevant or redundant features to reduce the number of features, thereby improving the model's accuracy and running time. AISQL leverages feature selection-based optimization to reduce internal data scans, data transmission between modules, and the data volume required for training. However, feature selection and feature importance methods integrated into data warehouse systems may be inaccurate, potentially leading to decreased model accuracy. To address this issue, AISQL incorporates optimizer statistics and trains on a small coreset [16] of data. Based on the feedback from the AI system, it determines the required feature columns and filters out useless or low-importance feature columns. Our AutoML module performs supervised feature selection, using χ^2 test, F-test, and mutual information. These tests identify and select features that have higher correlation with the target. This optimization is transparent to

Table 1 Three types of syntax for applying AI models in AISQL. The example SQL statements are based on the Iris table definition in Fig. 1. The example function name for UDF in Ⓐ Model-Oriented syntax is `m1`. The example predicted column name in Ⓑ Column-Oriented syntax is `p1`. Its updated column name is set as `p2`. When running inference, users can freely choose the older column `p1` or the newer `p2`. The default predicted column in Ⓒ Model-Transparent is `species_p` for Iris. And the automatic updating (re-training) of the model in Ⓒ does not change the name of `species_p`

Function	Ⓐ Model-Oriented	Ⓑ Column-Oriented	Ⓒ Model-Transparent
1. Train the model	`CREATE MODEL m1 WITH (TARGET = 'species') AS SELECT * FROM iris;`	`ALTER TABLE iris ADD PREDICT COLUMN p1;`	Auto
2. Check the training status	`SHOW MODEL STATUS m1 VERSION 1;`	`SHOW TABLE iris PREDICT p1;`	`SHOW TABLE iris PREDICT species_p;`
3. Inference of the model	`SELECT *, m1(sepal_length, sepal_width, petal_length, petal_width) FROM iris;`	`SELECT p1 FROM iris;`	`SELECT species_p FROM iris;`
4. Update (re-train) the model	`RETRAIN MODEL m1;`	`ALTER TABLE iris ADD PREDICT COLUMN p2;`	Auto (*the new name of the predicted column is still* `species_p`)
5. Activate a version of the model	`ACTIVATE MODEL m1 VERSION 1;`	*Specify by the name of the predicted column*.	Auto
6. Delete a model	`DROP MODEL m1 VERSION 1;`	`ALTER TABLE iris DROP PREDICT COLUMN p1;`	Auto

users. Users input the complete set of features for model training or inference, as shown in the first and third rows in Table 1Ⓐ. But the physical columns of the input features are reduced by our optimizer training in AutoML for inference in DWS[3)].

2.2.3 Multi-level scheduler

AISQL supports the asynchronous execution of AI jobs and SQL jobs. Multiple users can submit different model training tasks on different data simultaneously. The training data in one training job may also come from many different tables or even databases [17]. However, it is not trivial for backend execution threads to switch between multiple databases because one thread is bond to one database at the initialization moment. AISQL proposes a multi-level thread model (Fig. 3) to overcome this challenge. The multi-level thread model schedules and dispatches the hybrid AISQL jobs for execution. We maintain a single-thread scheduler and a multi-thread executor. The scheduler manages the computing resources, creates the executor threads and dispatches AI/SQL jobs to executors. The executor runs the AI/SQL jobs on the state machine for creating the model and update the status for the scheduler. The state machine submits the model training job, polls the status of AI jobs, updates the metadata for the model, registers the model and the inference UDF. For example, in Fig. 3Ⓐ, we can see User1 launches an AI/SQL job with DB1, and User2 launches two jobs with DB2 and DB3 separately. The scheduler will coordinate theses tasks in the job list (Fig. 3Ⓒ) for job management and running the state machine. It initializes the model training jobs from different databases, and triggers a new backend execution thread (Fig. 3Ⓓ) by sending the `SIGUSR2` signal to the postmaster process. This backend thread initializes local resources such as the corresponding databases according to the scheduling policies, and then trains the model for the database.

Overall, the AISQL thread model consists of four layers.

1. Client-side thread (Fig. 3Ⓐ): After a user launches the AI-extended SQL, this thread initiates a model training job (similar to model inference).
2. Server-side session thread (Fig. 3Ⓑ): This thread parses, rewrites and extends SQL to generate statement-level IR. It optimizes the DAG and executes part of the IR, including updating the model metadata, writing training data to an external table, and requesting AI/SQL jobs.
3. Server-side backend scheduler thread (Fig. 3Ⓓ): It manages AISQL jobs and executor resources. It also triggers the creation of executor threads.
4. Server-side backend executor thread (Fig. 3Ⓔ): it oversees the model state machine, including submitting the model training AI job to the AI engine, registering the model, updating the model metadata, polling the model training status, and synchronizing the scheduler status.

The scheduler supports parallel dispatch and creation of executor thread groups by a combination of databases and the user. Asynchronous model training enables efficient asynchronous scheduling execution AISQL assigns a priority for each job with the same combination of databases and the user based on the cost estimation. It configures the thread pool resources of the executor for different combination of databases and users.

2.2.4 Collaborative executor

AISQL extends the data warehouse parser to generate a statement-level IR. Fig. 4Ⓐ illustrates the complete processes from launching the AISQL statement to scheduling the DB engine and the AI engine. The IR is then converted into a

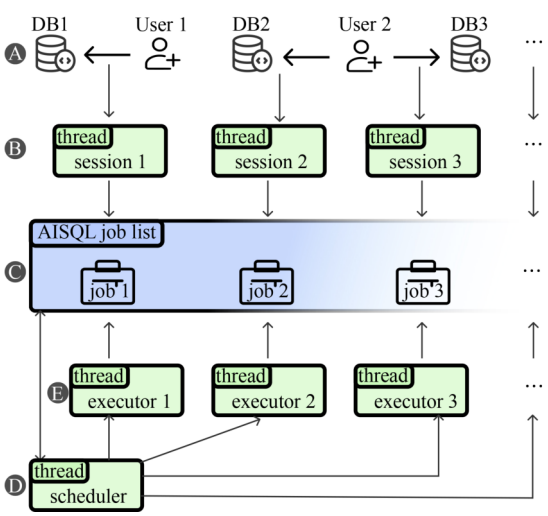


Fig. 3 The multi-level threading mechanism for our scheduler

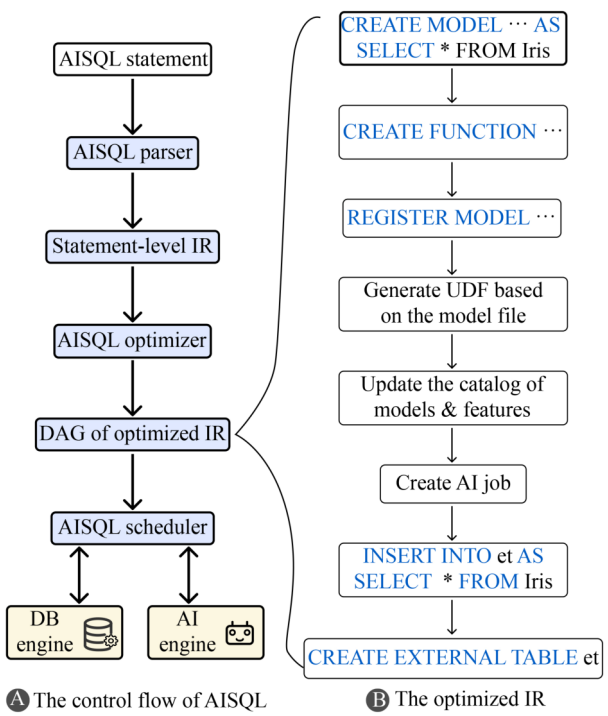


Fig. 4 The executor workflow and the DAG for creating a model

3) We use the short name DWS for GaussDB (DWS) for conciseness.

DAG that the AISQL scheduler can use to generate jobs. For clarity, we show an example DAG with only one path in Fig. 4Ⓑ. If the DAG contains standard SQL statements, then the scheduler will send the SQL job to the data node for execution and collect the results. If the DAG contains AI jobs, then the scheduler will send the job to the AI engine for execution. The scheduler keeps polling for the job status of training and retrieves the output when the training is complete. The pipelined execution in AISQL makes the training more efficient, especially when dealing with large training datasets or when the training data comes from complex SQL query. The executor pipelines 1) training data query, 2) training data scanning within the data warehouse directly with the 3) training data supply for AI engines. This reduces wait times for the AI engine, because the AI engine can begin processing data as soon as part of the training data becomes available.

This design offers several advantages: AISQL possesses an independent optimizer and scheduler, complemented by an embedded parser extension. This enables job-level parallel optimization based on DAG dependency relationships, which improves performance. It can also be more easily extended to multiple SQL engines, in which case we only need to implement a lightweight parser extension. The disadvantages of this design are that it does not support operator-level query plan optimization and performance tuning. However, job-level optimization can compensate for the lack of operator-level performance optimization.

In an alternative design where AISQL and DWS are tightly coupled together, AISQL does not have its own parser and optimizer. Instead, it extends the existing DWS components to support the parsing and execution of AI-extended SQL. SQL operators and AI operators are both in the same query planning tree. The executor parses the AI operators and schedule the jobs based on the dependency relationships among the operators. The SQL query plan can be sent directly to the data nodes to execute, and does not require the coordinator nodes to re-parse it as in other schemes. However, this approach faces challenges in supporting multiple SQL engines and dealing with all the SQL dialects in Huawei Cloud. For each type of SQL engine, this approach would need to extend all the dialect-related components, including parsers, optimizers, executors, etc.

Alternatively, we could abstract the extended parser as a separate AI + SQL module, making the parser, optimizer, and scheduler completely decoupled. AISQL can be extended to any SQL engine dialect by using the external database's parser. The third design separates the extended SQL statements into standard SQL and AI-related SQL based on its own splitter, and generates statement-level IR and optimized IR as a DAG. The advantages of this design are similar to those of our first design, offering the best extensibility for external SQL engines. However, it cannot achieve fine-grained optimization, and there exists duplicate parsing and deparsing on the coordinator nodes in DWS. The AI component and SQL component are loosely connected, lacking the collaborative job-level resources management as in our current approach.

2.2.5 AutoML

We implement an automated machine learning (AutoML) module for tabular data, offered as a distributed service running on Huawei Cloud. AutoML automatically performs data pre-processing, model selection and optimization. After training, the output is a pipeline of fitted components that can be used for inference on new data. Our AutoML module currently supports classification and regression problems. The backend algorithms do not yet include neural networks because in the enterprise, traditional ML models (i.e., linear regression and tree-based models) are still the most widely used [18,19] and usually perform better than neural networks on typical tabular data [20]. To plug other AI engines to AISQL, we only need to modify the input and output interfaces to take Apache Arrow [21] format for connecting our storage services.

2.2.6 DWS

We plug DWS into AISQL as an initial verification. Huawei DWS is a cloud-native distributed data warehouse system featuring a massively parallel processing engine. It comprises multiple logical nodes, each equipped with its own independent CPU, memory, and storage resources. In this system architecture, business data is distributed across multiple nodes. Data analysis tasks are pushed to the nodes where the data resides for local execution. This enables parallel processing of large-scale data and fast response to data requests. DWS can be integrated with ETL tools, BI tools, data mining and analysis tools through standard interfaces. If users want to apply ML models to their data in DWS without AISQL, they will need to export the data to some external ML framework first. A DWS cluster consists of multiple nodes with the same configuration in the same subnet. The cluster is designed to be highly scalable and can store up to 10PB of data. The coordinator node splits and schedules tasks for parallel execution on data nodes. The multi-layer parallel computing of DWS ensures rapid processing and can meet very high SLA requirements. DWS has been battle-tested by thousands of corporations over the last decade and proven to be highly stable and efficient. Despite the achieved success, traditionally we could not help customers to apply AI models to their existing data stored in DWS. Training models outside of DWS requires a high level of expertise in AI models. AISQL democratizes the application of AI models in DWS for predictive analytics.

Thanks to the composable design of AISQL, we only need to extend the parser to support other popular data warehouses or databases, e.g., ClickHouse [22], Apache Druid [23], and MySQL [24]. The query engine's IR optimizer and multi-level scheduler are pluggable modules, seamlessly integrating with data warehouses without any change. Because the parser of AISQL is based on the inter-operable Substrait [13] format of IR, the different warehouses can even inter-operate with each other under the umbrella of AISQL. For example, some companies may have their data fragmented across numerous data warehouses, they can call a simple SQL `join` operator to integrate them and feed into the AI pipeline. For the storage engines we introduce later, they only need to have Apache Arrow [21] format to plug into our query engine.

2.3 Persistent storage and fast cache

2.3.1 SRSS

AISQL puts the training data and model in the shared reliable storage service (SRSS) [25,26] at Huawei Cloud for persistent storage. SRSS is a cutting-edge distributed storage service designed for high performance and strong consistency across data centers and availability zones. It operates as a log-structured, append-only system where data is stored in persistent logs (PLogs). Each PLog is a contiguous, fixed-size chunk in SSD that disallows in-place updates. SRSS achieves reliability through three-way data replication across storage nodes. It ensures write operations are swiftly redirected to healthy nodes in case of failures and promots uninterrupted service availability. The service maintains an organized collection of data PLogs for database logs, with metadata stored in separate PLogs for efficient management and access. This systematic data and metadata handling further exemplifies SRSS's commitment to providing a high-performance, reliable distributed storage service.

2.3.2 MetaStore

The MetaStore service at Huawei Cloud is a key infrastructure for digital integration. It empowers customers to unify the management, permission control, and transaction of metadata from multiple engines. Traditional methods only support the definition, modification, and query of metadata through SQL statements. A complete authorization activity requires two authorization operations to be performed on the computing engine and object storage, which is inconvenient for customers. MetaStore solves these problems by providing a unified API with visual interface for the definition and authorization of data lake metadata. It supports convenient operation and rapid construction by customers. AISQL utilizes MetaStore to manage the metadata of AI models, training data, and inference data.

2.3.3 DSM

To further speed up the parallel training data supply, we introduce a distributed shared memory (DSM) [27] layer above SRSS to connect the computation components. DSM is a novel elastic and adaptive memory-disaggregated caching system designed for our AI + SQL workload in the cloud environment. In-memory caching systems are crucial for reducing service latency and improving throughput in cloud services But many of them face limitations due to the tightly coupled CPU and memory on conventional servers. DSM proposes to leverage the disaggregated memory architecture to separate compute and memory resources. In this way, DSM allows more flexible and independent resource allocation. DSM introduces a client-centric caching framework that efficiently executes various caching algorithms over the compute pool in a disaggregated memory environment, utilizing only remote memory accesses. DSM offers a flexible and efficient execution of various caching algorithms on disaggregated memory, a sample-friendly hash table and frequency counter cache for improved performance. Its distributed adaptive caching mechanism especially suits our needs for training data supply because of its optimal algorithm selection based on dynamic conditions.

3 Model management

This section presents the model management mechanism in model lifecycle.

3.1 Model operations

As discussed in Section 2.2.1, we can use the `CREATE` or `REGISTER` keyword to introduce a new model in AISQL. Figure 5 shows the activity diagram among different components for creating a model.

The key workflow for `CREATE MODEL` is divided into two parts: front-end synchronous execution and back-end asynchronous execution. (1) *Front-end synchronous execution*. The `CREATE MODEL` DDL (data definition language) is executed in the cluster. A tuple of the model metadata is inserted into the table for models in MetaStore, generating a globally unique model identifier. A state machine for a specific version of model training is also created. AISQL then queries the training data. DWS saves the metadata for the training data to MetaStore and writes the training data to the distributed shared memory. A job for model training is submitted to the AISQL scheduler. The scheduler then distributes the job to an execution thread for execution. (2) *Back-end asynchronous execution*. The AISQL scheduler selects a job according to the scheduling policy. It then starts a thread to execute the state machine for model training. It submits a job to AutoML to obtain a job ID. The execution thread then periodically queries the status of the job until the model training is successful or fails. Once the model is trained

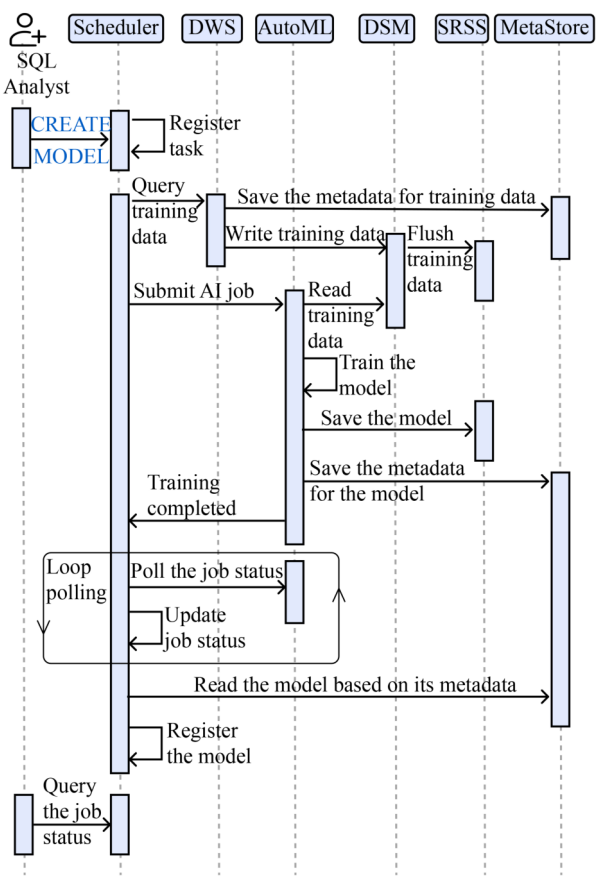


Fig. 5 The activity diagram for `CREATE MODEL`

successfully, AutoML writes the model file to DSM. The state machine for model training loads the model file and registers the model inference UDF. The inference UDF will then call the model for inference. The state machine for model training then updates the model system table. If the model training is successful, the newly trained version will be marked as the active version. If the model training fails, the system table will be updated and the reason for the training failure will be recorded.

As for `REGISTER MODEL`, the workflow is completed in the front-end synchronous thread. The `REGISTER MODEL` DDL is executed on the cluster. Just like what happens in `CREATE MODEL`, a tuple of the model metadata is inserted into the model table in MetaStore, generating a globally unique model object identifier. The model file is copied from a user-specified external address to a unique internal model file path. The inference UDF will then call the model for inference after the UDF is registered.

3.2 Model metadata

The model metadata is stored in three tables in MetaStore. (1) The overall model metadata table. It includes several crucial attributes: a unique identifier for each model (the primary key), model names, the model task type (classification / regression), the target column and its type, the active version of the model, and the data source for the model, specified by the query statement used during the creation of the model. (2) The feature column table. It includes the unique identifier for each model (part of a composite primary key), a feature column index (the second part of the composite primary key), the name and type of the feature columns, and the importance of the feature column for training and inference. (3) Lastly, the model version table includes the unique identifier for each model and a version number (both forming a composite primary key); the status of the model version which reflects the state recorded by the model version's training state machine; the start and end times of model training; the save path for the model file; a unique identifier for the model version's training task; reasons for training failure, if any; and the model version's registered path. This structure facilitates the comprehensive management of AI models within AISQL.

4 Data flow

This section describes the data management for model training and inference (Fig. 6).

4.1 Training data supply

Figure 6 shows how the data flows in AISQL for training and inference. The control flow mainly happens in top-level control components except the lower storage components. All dashed arrows indicate the training data flow, except for ❶. The dashed arrow ❶ represents the flow of the trained model parameters from DSM to the data warehouse engine for in-warehouse inference. And the model parameters are usually very small compared with real-world industrial data. The data flow for model training starts from the dashed arrow Ⓐ where the training data selected by the optimizer of AISQL is written by the DB engines to DSM. Then the AutoML module reads the training data from DSM as indicated by the dashed arrow Ⓑ.

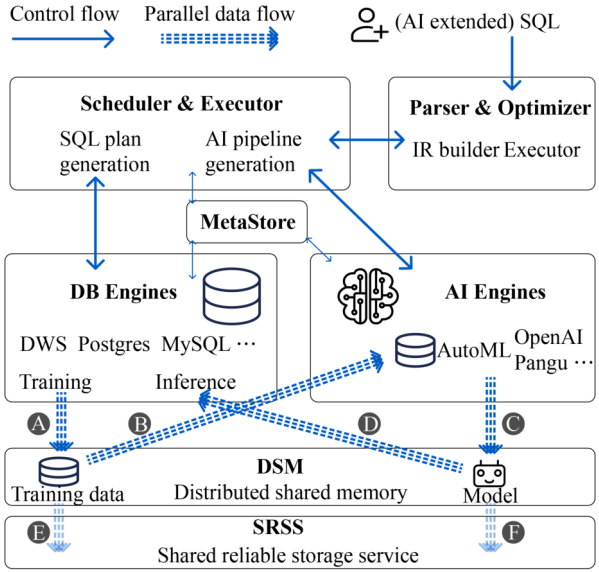


Fig. 6 The data flow in AISQL. The solid arrows indicate the control flow. The dashed arrows indicate the data flow (dashed arrow ❶ is for model parameters)

AISQL initiates model training either following a CREATE MODEL SQL statement or in response to an automatically detected shift in data distribution. As introduced in Section 2.2.2, the optimizer will identify the less important features and abandon them for further training. The less important feature columns stay in DWS and will not be sent to AutoML. Once DWS begins to scan the needed training data, it instantly saves the metadata to MetaStore so that the AI engine can begin model training even with part of scanned training data. Small data sets are directly transmitted to AutoML for training, reducing the overhead of network communication between engines. Large data sets are parallelly written to DSM (Fig. 6Ⓐ), and AI engines parallelly pull data from the fast cache provided by DSM (Fig. 6Ⓑ). Each data node writes to DSM in the Apache Arrow [21] format in a pipeline of batches. Writing operators are pipelined with upstream operators like scanning to reduce latency. Writing in batches reduces the overhead of address lookup. Finally, the scheduler in DWS calls the AutoML module to start the training job. The AutoML module then reads the reduced training data from DSM layer (Fig. 6Ⓑ) and starts the training process. Our AutoML also takes data in Apache Arrow [21] format as input to avoid format conversion. After the training is succeeds, AutoML parallelly writes the model to DSM (Fig. 6Ⓒ). DSM automatically flushes data to SRSS based on its caching policy for persistent storage and high availability. The light parallel arrows in Fig. 6Ⓔ and Ⓕ indicate the persistence of training data and model parameters respectively. Normally, data integration can be achieved by a `join` operation. If complex data cleaning is needed, users can also write their own UDFs in the `CREATE MODEL` statement to transform the columns.

4.2 Inference data

AISQL achieves predictive analytics by running embedded or remote model inference. In embedded inference, the model file is loaded locally from DSM (Fig. 6❶). DSM will retrieve

the model from SRSS in case of cache-missing. The serving process is embedded in the DB engine. The model file can be internally trained one or registered externally from other sources. The supported model formats are ONNX [15] and PMML [14], which are converted to C++ shared object for fast execution. The execution of model inference UDFs leverages vectorization. The vectorization processes data in batches for faster inference. Users can also register remote inference services for advanced AI tasks like language understanding, e.g., OpenAI GPT-4 [28] or Huawei Pangu [29]. In remote inference, AISQL sends the inference data to a remote inference service. The remote inference service generates predictions and returns them to AISQL to write into DWS. The predictions are written in DWS as the predicted column introduced in Section 2.2.1.

5 Evaluation

We benchmark the performance of AISQL against other state-of-the-art open-source and proprietary systems on nine ML benchmark datasets. AISQL's training and inference are up to 19× faster, while its accuracy is always at least as good as the competitors on eight out of the nine datasets.

5.1 Experimental setup

5.1.1 Evaluation metrics

We report wall-clock times in seconds by running each system once and then reporting the average of four subsequent runs with warm cache, following [30]. We evaluate the classification tasks by balanced accuracy (BA) and regression tasks by R^2 score on the hold out test set, following recent works [31−33] about ML tasks on tabular data. BA is defined as the average recall obtained for each class. R^2 is a statistical measure of how well the regression line approximates the actual data.

5.1.2 Hardware and software configuration

All the experiments on SQL+AI systems are performed on a 3-node cluster. The operating system for our experiments is CentOS 7.5. Each node has 8 vCPUs, 16 GB of RAM, and is connected to a 10 Gbps intranet. Our AutoML can also run on GPU servers without modification. But some of the baselines and algorithms don't support GPU environment so we conduct all the experiments on the CPU servers. For testing the traditional AI workflow, we run auto-sklearn [34] on a cloud server with 48 GB free memory and 24 vCPUs, following prior studies [35,36]. We set sklearn[4)] to train parallelly on the 24 vCPUs for a fair comparison with AISQL.

We use the latest DWS 9.0.0 for the experiments. We include B1, MindsDB [4], and H2O [37] as the baseline systems. B1 (name hidden for legal reasons) is the SQL + AI product provided by one of the most popular cloud computing vendors. MindsDB is a flexible open-source middleware connecting data systems with AI engines. We choose raw file with the default AutoML implementation in MindsDB version 24.2.3.0. H2O is a scalable open-source AutoML platform. We test on the latest H2O 3.44.0 installed with Spark 3.2.2. We shutdown all the other systems at the time of running experiments on one system.

5.1.3 Tasks and datasets

We use nine ML benchmark datasets in the experiments. We choose the relatively large IP dataset for testing AISQL's own scalability against the traditional ML workflow. And other eight standard ML benchmark datasets for comparison with other SQL + AI or distributed ML systems.

IP dataset for testing scalability. This dataset was collected in a network section from Colombia by performing packet captures at different hours. Its goal is to predict the application name from one of the 78 categories. The total file size in CSV (Comma Separated Values) format is 1.77 GB. We randomly selected 30% of the data as a test set, then we randomly sampled from the remaining data to generate train sets with sizes of 0.5 GB and 1 GB. In a real-world production environment, it is common to see gigabytes of training data for ML tasks. But we didn't find popular open source tabular datasets larger than 5 GB. So we use the SDV [38] synthesizer to fit the IP dataset and generate 3 GB to 1024 GB of training data to test our system performance. We didn't compare the IP dataset for accuracy against other SQL + AI systems because B1 is too expensive to run such a large dataset.

Other datasets for testing accuracy and time. We also include popular classification and regression datasets across a wide range of domains. For the original datasets, we randomly sample 70% for training and 30% for testing. Table 2 shows the detailed statistics of all the datasets for the training part.

5.1.4 Experiment protocol for traditional AI workflow

We conduct interviews with many of our clients to figure out the real-world traditional industrial AI workflow without AISQL. We find out that most companies have their data engineers and ML engineers operate independently in separate cloud environments. Data engineers have their own cluster for managing datasets, while ML engineers use a separate cluster for model training. Hence we keep the above setting as the traditional ML workflow in our experiments. We assume the two clusters of data engineers and ML engineers have the same hardware configuration. Both DWS with data exportation scripts and the AutoML training and inference

Table 2 Statistics of the training datasets. The $\overline{Acc}$ column shows the mean of balanced accuracy of all the systems for classification tasks, or the mean of R^2 of all the systems for regression tasks. Cls denotes classification and Reg denotes regression. #Fts denotes number of features

Dataset	Task	#Rows	#Fts	$\overline{Acc}$
IP [39]	Cls	3,577,296	87	0.9241
Census [40]	Cls	32,561	15	0.7941
Airline [41]	Cls	377,568	8	0.6506
Connect4 [42]	Cls	47,290	43	0.6916
Bank [43]	Cls	31,648	17	0.7172
Friday [44]	Reg	116,774	10	0.5211
Diamonds [45]	Reg	37,758	17	0.9752
Taxi [46]	Reg	407,284	15	0.2610
Mercedes [47]	Reg	2,946	377	0.6132

4) We call auto-sklearn as sklearn from now on for conciseness.

scripts are pre-deployed on the cloud. In this way, we omit the time spent on developing the program for model training and inference by ML engineers.

The data engineers execute the data exportation script on DWS to save the training data in CSV format to SRSS. The data engineers then download the training data from SRSS and share it with the ML engineers. The ML engineers upload the data to a separate SRSS bucket and run the pre-deployed scripts to train the model. The speed of download and upload in our experimental environment is about 2 MB/s, which is the typical network speed of our customers. The total time taken is the sum of the time taken to export data from DWS to SRSS, download from and upload to SRSS, and model training by sklearn.

In the Beta test phase, we conduct a qualitative survey with SQL experts to evaluate how AISQL facilitates domain experts for applying AI in SQL (results in Section 5.4).

5.2 Results

5.2.1 Compare model training time with traditional ML workflow

Section 5.1.4 describes the experiment protocol for traditional ML workflow. For AISQL we log the time between when the `CREATE MODEL` SQL statement is launched and when the model training is complete. Table 3 shows the training time costs of AISQL and sklearn baseline across a wide range of training dataset sizes. The speedup factor is calculated as the time cost of sklearn divided by the time cost of AISQL. The baseline could not finish training due to out-of-memory errors when the dataset size exceeds 5 GB. In such cases, we provide linearly estimated time costs for sklearn (marked with *) in Table 3 assuming linearity to be one. Table 3 shows that AISQL consistently outperforms the baseline sklearn approach in terms of model training, data flow, and the total time cost. AISQL is four to seven times faster than the traditional sklearn baseline. Our advantage is more obvious when dealing with larger datasets. One highlight in Table 3 is that our data flow speed is 26× to 43× faster than the traditional way. The data flow in AISQL includes scanning the training data in DWS and writing data to DSM before the actual training (updating model parameters) happens. Writing to DSM takes very little time and is pipelined with the prior in-DB scan operation. Training in AISQL is also in the pipeline. AISQL begins training when half of the data is written to DSM. Thus, the total training time for AISQL is *less* than the simple sum of computation and data flow in Table 3. In contrast, the data flow in sklearn involves the cumbersome process of moving from data engineers' environment to ML engineers' environment, as discussed in Section 5.1.4. The data flow in AISQL and sklearn both include the time cost for saving the trained model. Our time cost of the training part is also consistently faster than the baseline.

5.2.2 Compare model inference time with traditional ML workflow

Table 4 shows that AISQL significantly reduces the time required for model inference compared to the baseline sklearn. For all experiments on varying sizes of inference data, we

Table 3 Total training time (seconds) of AISQL and sklearn on different sizes of IP dataset. The training part logs the time actually spent on learning model parameters when the dataset is ready in place. The data flow part logs the time for exporting data out of DWS and saving the trained models

IP dataset size/GB	Training			Data flow			Total		
	AISQL	sklearn	Speedup	AISQL	sklearn	Speedup	AISQL	sklearn	Speedup
0.5	209	401	1.9	11	481	43.7	220	882	4.0
1	306	899	2.9	35	908	26.0	341	1,808	5.3
1.5	367	916	2.5	46	1,877	40.8	413	2,793	6.8
2	472	1,053	2.2	71	1,927	27.1	543	2,980	5.5
3	634	1,384	2.2	68	2,783	40.7	702	4,167	5.9
5	811	2,277	2.8	187	4,928	26.4	998	7,205	7.2
10	1,756	*4,152	*2.4	339	*9,699	*28.6	2,095	*13,851	*6.6
100	15,386	*38,587	*2.5	2,968	*96,417	*32.5	18,354	*135,004	*7.4
1024	150,502	*392,121	*2.6	22,561	*986,725	*43.7	173,063	*1,378,846	*8.0

The baseline could not finish training due to out-of-memory errors when the dataset size exceeds 5GB. In such cases, we provide linearly estimated time costs for sklearn (marked with *)

Table 4 Total inference time (seconds) of AISQL and auto-sklearn on different sizes of IP dataset. The inference part logs the time actually spent on running inference when the dataset is ready in place. For sklearn, the data flow part logs the time for exporting data out of DWS and loading the trained models. For AISQL, the data flow only involves loading model parameters from DSM to DWS

IP dataset size/GB	Inference			Data flow			Total		
	AISQL	sklearn	Speedup	AISQL	sklearn	Speedup	AISQL	sklearn	Speedup
0.5	41	52	1.3	7	501	70.6	41	553	13.5
1	67	99	1.5	9	955	102.7	67	1,054	15.7
1.5	92	145	1.6	8	1,811	229.2	92	1,956	21.3
2	110	191	1.7	6	1,866	291.6	110	2,057	18.7
3	168	283	1.7	5	2,776	523.8	168	3,059	18.2
5	270	468	1.7	4	4,596	1,121.0	270	5,064	18.8
10	525	930	1.8	6	9,147	1,524.5	525	10,077	19.2
100	5,102	9,248	1.8	8	91,053	11,381.6	5,102	100,301	19.7
1024	51,695	94,738	1.8	8	931,859	116,482.4	51,695	1,026,597	19.9

keep the models fixed for AISQL and sklearn respectively, and only changes the sizes of the inference data. The one model used for testing AISQL is trained on the original 1.77 GB IP dataset by AISQL. And the one model used for testing sklearn is trained on the same dataset by sklearn. For inference in sklearn, the total time taken is calculated as the sum of the time taken to export data from DWS to SRSS and the time taken for sklearn to perform inference. In contrast, the inference data flow in AISQL does not involve exporting data because we register the trained model as an in-warehouse UDF. For AISQL, the data flow is much simpler. AISQL has a compute and storage separation design for scalability. Hence it stores the trained model in DSM and automatically flushes to SRSS. The inference data flow in AISQL only involves loading model from DSM to DWS (Fig. 6❶). Since the model size is independent of the inference dataset size, AISQL's time costs for data flow do not change much (all less than 10 seconds). The model size is very small compared with our large dataset, so AISQL can be 10K× faster than the traditional data flow. Beyond fast data flow, our actual inference speed also surpasses sklearn due to the distributed ability of DWS.

5.2.3 Compare with other SQL AI systems

Figure 7 shows the training time and test accuracy for the classification and regression tasks. The metric for test accuracy in classification is balanced accuracy, and R^2 for regression, as introduced in Section 5.1.1. The last column $\overline{BA}$ / R^2 in Table 2 displays the arithmetic mean of the four systems compared against the eight datasets. AISQL beats all the other competitors by a significant margin w.r.t. the training time. The training time of AISQL is often less than half of other competitors. Furthermore, AISQL outperforms in five out of eight ML tasks w.r.t. balanced accuracy or R^2. For the datasets Friday and Census, AISQL ties with the best proprietary system B1. The balanced accuracy of AISQL is only slightly behind (by less than 1%) B1 on the Connect4 dataset. Notably, on the challenging Taxi dataset, where the average R^2 score is 0.2610, AISQL leads by a large margin

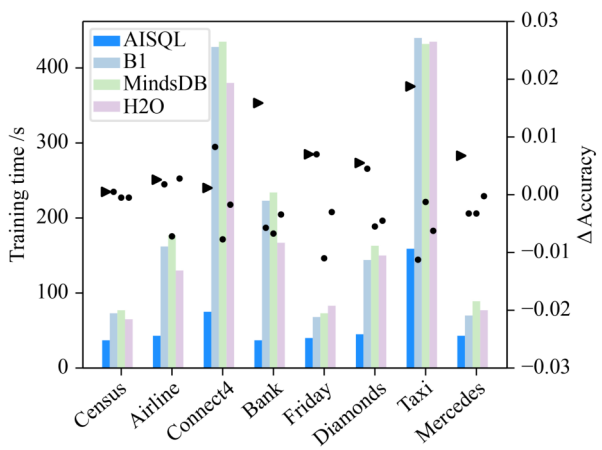


Fig. 7 The training time (seconds) and accuracy results for ML tasks. The Δ of accuracy is calculated as the actual accuracy value minus the mean accuracy over a dataset. The heights of the colored bars indicate the training times. The heights of the black triangles and dots indicate the Δ of balanced accuracies

(almost 2%). H2O is the second fastest method in classification, owing to Spark's rapid in-memory computing. However, H2O requires sizing the cluster memory to be about four times the size of training data for best performance. H2O reports out-of-memory errors for datasets larger than 16 GB. On the other hand, AISQL can support training sets of any size that fit within data warehouse disk (e.g., 1024 GB) due to its use of disaggregated shared memory.

We believe the accuracy advantage mainly comes from our optimized data transfer, because all the baseline systems can not finish training when AISQL has finished. MindsDB does not have the interface to control training time. So we test on B1 and H2O, setting the maximal running time to the double of their default running time. In that case, B1 and H2O have a negligible accuracy gap (less than 3.1‰, Table 5) to AISQL.

5.3 Ablation study

We study the contributions of the main modules in AISQL by evaluating different configurations: (C1) Turn off the feature selection (FS) in the IR optimizer for training. We test C1 on the eight ML datasets because FS highly depends on the dataset characteristics. (C2) Turn off the pipelined execution of training data supply for model training. The model training only starts after all of the training data has been written to DSM. We test C2 on the IP dataset because C2's performance highly depends on dataset sizes.

(C1) Figure 8 illustrates the training time consisting of computation and data flow w/ and w/o FS. Except for Airline, Connect4, and Diamonds datasets, FS significantly reduces the training time in all other five datasets. The maximum speedup factor is 1.3× achieved on the Mercedes dataset, which has the largest feature set (377 features) While in industrial scenarios, it is common to encounter datasets with more than thousands of features. Removing irrelevant or redundant features reduces both computational and network transportation costs while avoiding degradation of learning performance. Interestingly, the total training time w/o FS for Airline and Connect4 is slightly faster than the original AISQL w/ FS. This is due to a small overhead associated with running FS before the actual training process starts. Moreover, all features are critical in these two datasets, so no feature is removed during training. Figure 8 does not include the IP dataset because it is not on the same scale with other datasets and can not fit in the same figure. We observe a similar pattern on the 1.77 GB original IP dataset, where the FS speedup is 1.2×. Overall, FS proves to be effective in reducing training time.

Table 5 Accuracy values of B1 and H2O minus the accuracy of AISQL. The training time is the double of the default training time for B1 and H2O

	AISQL	ΔB1×2	ΔH2O×2
Census	0.7946	2.6‰	−0.3‰
Airline	0.6532	1.6‰	−1.5‰
Connect4	0.6899	0.9‰	2.0‰
Bank	0.7331	0.0‰	0.1‰
Friday	0.5343	−3.1‰	−1.1‰
Diamonds	0.9810	0.1‰	0.7‰
Taxi	0.2811	1.2‰	0.4‰
Mercedes	0.6201	2.8‰	−1.5‰

Table 6 Ablation study results on pipelined training. The last column lists the total training time with pipeline turned off divided by the time with pipeline turned on (speedup factor)

IP dataset size/GB	Computation		Data flow		Total		Speedup
	on	off	on	off	on	off	
0.5	209	205	11	21	216	226	1.05
1	306	303	35	67	326	370	1.13
1.5	367	362	46	89	393	451	1.15
2	472	466	71	137	513	603	1.18
3	634	627	68	132	664	759	1.14
5	811	803	187	355	903	1,158	1.28
10	1,756	1,743	339	648	1,920	2,391	1.25
100	15,386	15,371	2,968	5,699	16,867	21,070	1.25

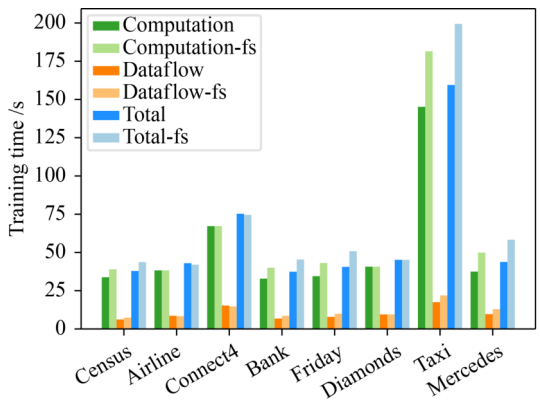


Fig. 8 Ablation study results on feature selection for model training. The suffix "-fs" indicates that feature selection is turned off. We use paired color scheme and the lighter colors represent the condition with feature selection turned off

(C2) Table 6 presents the training time with pipelined execution of data supply for model training turned on and off. The data flow becomes about 2× slower when the pipelined execution is turned off. This hurts the performance especially when the training set is large. Although the computation part of training is slightly faster than pipelined execution, because AutoML does not interact with DSM, the total time is still 5% to 25% slower when the pipeline is turned off.

5.4 Qualitative evaluation

We invited 155 working experts in SQL analysis to conduct the survey on the usability of AISQL. All of the participants have been tested on composing the right SQL queries for 4 questions, and only the 146 experts who passed the 4 SQL questions were allowed to continue the survey. Participants were between 28 and 45 years of age ($age_{mean} = 34.07$, $age_{std} = 6.81$), with an average of over five years of experience in SQL analysis. All of the participants have been using DWS with AISQL installed for at least two months. Participants self-reported their familiarity with normal SQL with a rating of 6.11 + 0.75, where 1 represents "No Experience" and 7 represents "Expert". Moreover, they were often assigned the task of applying AI models to their data in the warehouse, with a frequency of 10 to 25 times per week.

Figure 9 shows the results of the qualitative evaluation. The detailed questions are listed in the caption of Fig. 9. First, results on Q3 show that the overall quality of AISQL w.r.t. response speed and ease of use is fairly good. In Q3, only two experts gave a neutral rate on the simplicity of AISQL. All the other experts rated at least 4 points for the overall usage experience of AISQL. They highly appreciated the seamless integration between AI and SQL, as well as the near-data training and inference. As one of the experts mentioned, "Without AISQL, we have to export the data out of our warehouse and send to another AI team to run predictive analysis. AISQL just made our life much easier." Q1 and Q2 show the detailed ratings on the novel model-transparent features of AISQL: hidden column as predicted column and automatic model re-training. Still, over 80% of the experts strongly agree that these two features are useful and effective. For Q1, most experts liked our idea of hidden column because they could more easily adapt their existing SQL code to run AI inference by simply adding a suffix _p to the predicted column. However, we do get around 10% of 4-point scores

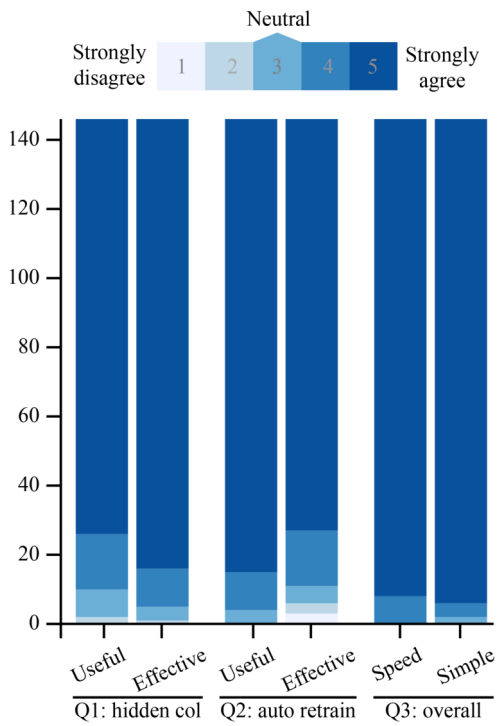


Fig. 9 The results of the qualitative evaluation. Y-axis denotes cumulative population. Q1: What's your usage experience about the hidden column as predicted column in AISQL? Please rate this feature's usefulness and effectiveness. Q2: What's your usage experience about the automatic model re-training during the usage of AISQL? Please rate this feature's usefulness and effectiveness. (Usefulness is a metric for how users want this feature. And effectiveness is a metric for how we have implemented.) Q3: Overall, is AISQL fast (response speed) and simple for you to use?

w.r.t. the usefulness of hidden column and effectiveness of auto re-training. Because some of the SQL experts are just used to manage the models manually to get a more precise control. To sum up, the qualitative evaluation shows that AISQL is easy to use, solving the challenge 2 in Section 1.

6 Related work

Composable DBMS. In the field of data management, the move towards composable systems [48] represents a significant shift from traditional monolithic designs. Prior work has explored various facets of composability, including the development of open source technologies like Apache Arrow [21] for cross-language data interoperability and the establishment of standardized APIs to facilitate component reuse across different systems. Projects like Google's ZetaSQL [49] and Meta's CoreSQL [50] have demonstrated the feasibility of creating unified language frontends, while the advent of Substrait [13] offers a unified intermediate representation to bridge language and execution layers seamlessly. Furthermore, initiatives such as Apache Calcite [51] and the Orca [52] optimizer underscore efforts towards creating reusable components for query optimization. On the execution front, Velox [53] exemplifies the potential for a unified execution engine that can serve a wide array of data management systems. This shift towards a composable future in data management encourages collaboration and aims to accelerate the pace of innovation by building on shared foundations. AISQL highlights the important design decisions that were rarely discussed in the research literature for practitioners to build a production-ready composable SQL + AI system in a cloud environment.

External integration. One direct solution for integrating machine learning into existing SQL databases is through a glue layer like MindsDB [4] to stitch these two worlds together. This approach connects SQL systems with external machine learning platforms or libraries. Enabling data to be extracted for model training and results to be fed back into the database for influence. Industrial products like Microsoft SQL Server Machine Learning Services [54] gives the ability to run Python and R scripts with relational data in the database. Raven [3,55] optimizes the inference queries but does not touch model training. This approach offers flexibility in model choice and tooling, but ignores the opportunities of performance optimization in reduced data transfer, larger than-RAM data sets, etc. While in AISQL, our hybrid job management and efficient data flow help to mitigate these issues.

In-DB machine learning. This category of methods augment SQL systems by integrating machine learning algorithms and operators, facilitating the analysis of data and the training of models directly within the database environment. Many DBMS vendors provide algorithms for in-database model training and prediction using SQL queries, including Amazon Redshift ML [2], Oracle ML [9], IBM Db2 Machine Learning [56], PolarDB [57], SAP HANA Predictive Analysis Library (PAL) [58], Google BigQuery ML [1], etc. Many existing efforts in the research community enhance database capabilities through UDF extensions. For example, MADlib [59] offers in-database analysis with SQL-based algorithms for machine learning, data mining, and statistics. MASQ [60] converts selected scikit-learn classifiers and regressors to plain SQL to execute in most SQL systems. Schule et al. [61] also try to express the machine learning pipeline via SQL with recursive tables. Dan [62] overviews ML over relational data as a database problem, justified by the need for feature extraction and group-by aggregation. Gandhi et al. [63] claim that achieving a truly AI-centric database requires moving the DBMS engine from a relational to a tensor abstraction. OpenDBML [64] demonstrates efficient in-DB ML by providing a user-friendly Python interface. To the best of our knowledge, AISQL is the first system to support the fully automate, transparent-to-user AI model training and inference.

Model management. ModelHub [65] manages the entire deep learning model lifecycle. This end-to-end system facilitates storing, versioning, snapshotting, querying, and reusing models alongside their associated data artifacts like hyperparameters and trained snapshots. ModelHub utilizes three key components: model versioning system, domain-specific language module, and hosted deep learning model sharing. ML pipeline debugging and model diagnosis tools tackle different stages of the machine learning workflow to identify and rectify issues. Pipeline debugging starts with data verification [66], ensuring its integrity before delving into model behavior. Model diagnosis tools like [67] focus on understanding why models underperform. They analyze both model parameters and data artifacts during development, which poses data management challenges due to the large volume. AISQL manages the full lifecycle of AI models via the metadata described in Section 3.2.

7 Conclusion

Bringing the training and inference of AI models to SQL in data warehouse has many advantages, such as reduced data transportation overhead, consistent user-experience, better scalability, data integrity and access control. However, current implementations still suffer from limited reusability, steep learning curve and expensive data transfer. GaussDB-AISQL is a composable cloud-native SQL system with AI capabilities to solve these challenges. GaussDB-AISQL efficiently manages AI and SQL jobs in a holistic manner. GaussDB-AISQL includes an AutoML module for model training, as well as its coordination mechanism with the workflow scheduling framework. Its SQL-like syntax for predictive analytics is model-transparent for users and extremely easy to use. GaussDB-AISQL features in an efficient data flow mechanism between components such as data warehouses, AI engines, metadata storage and shared reliable storage service. The pipelined training data supply and feature reduction significantly speedup the training process. GaussDB-AISQL embeds model inference within DWS to benefit from the scalability provided by DWS and avoids unnecessary data exporting to the AI engine. Currently, GaussDB-AISQL only supports classification and regression on tabular data and lacks the ability to understand text or image data. However, this limitation can be addressed by enhancing the AutoML module

or using remote inference, thanks to our composable architecture. GaussDB-AISQL has already been deployed in Huawei Cloud DWS and served many industrial clients.

Evaluation results show that GaussDB-AISQL provides up to 8× faster performance for model training and up to 19× faster for model inference compared to the traditional ML workflow. When compared to other SQL + AI systems, GaussDB-AISQL still achieves less than half the training time and higher accuracy.

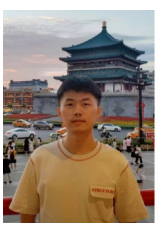
Acknowledgements We thank the reviewers for their constructive feedback. This work was supported by the fund for building world-class universities (disciplines) of Renmin University of China.

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

References

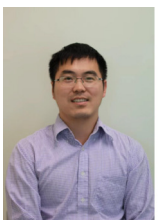
1. Marrandino, Alessandro. Machine Learning with BigQuery ML: Create, execute, and improve machine learning models in BigQuery using standard SQL queries . Packt Publishing Ltd, 2021.
2. Amazon Web Services, Inc. Amazon redshift machine learning. See docs.aws.amazoncom/redshift/latest/dg/machine_learning website, 2024
3. Park K, Saur K, Banda D, Sen R, Interlandi M, Karanasos K. End-to-end optimization of machine learning prediction queries. In: Proceedings of 2022 International Conference on Management of Data, SIGMOD '22. 2022, 587−601
4. MindsDB. MindsDB. See mariadbcom/about-us/partners/mindsdb/ website, 2024
5. Huang B, Babu S, Yang J. Cumulon: optimizing statistical data analysis in the cloud. In: Proceedings of 2013 ACM SIGMOD International Conference on Management of Data. 2013, 1−12
6. Cohen J, Dolan B, Dunlap M, Hellerstein J M, Welton C. MAD skills: new analysis practices for big data. Proceedings of the VLDB Endowment, 2009, 2(2): 1481−1492
7. Lin Q, Wu S, Zhao J, Dai J, Li F, Chen G. A comparative study of in-database inference approaches. In: Proceedings of the 38th IEEE International Conference on Data Engineering (ICDE). 2022, 1794−1807
8. Wang Y, Yang Y, Zhu W, Wu Y, Yan X, Liu Y, Wang Y, Xie L, Gao Z, Zhu W, Chen X, Yan W, Tang M, Tang Y. SQLFLow: a bridge between SQL and machine learning. 2020, arXiv preprint arXiv: 2001.06846
9. Oracle Corporation. Oracle machine learning. See Docs.oracle.com/en/database/oracle/machine-learning/ website, 2024
10. Wang D, Andres J, Weisz J D, Oduor E, Dugan C. AutoDS: towards human-centered automation of data science. In: Proceedings of 2021 CHI Conference on Human Factors in Computing Systems. 2021, 79
11. Jordan M I, Mitchell T M. Machine learning: trends, perspectives, and prospects. Science, 2015, 349(6245): 255−260
12. Paganelli M, Sottovia P, Park K, Interlandi M, Guerra F. Pushing ML predictions into DBMSs. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(10): 10295−10308
13. Substrait. See Github.com/substrait-io website, 2024
14. Group T D M. The predictive model markup language. See dmg.org/pmml/pmml-v4-4-1.html website, 2024
15. ONNX. See Onnx.ai/ website, 2024
16. Chai C, Wang J, Tang N, Yuan Y, Liu J, Deng Y, Wang G. Efficient coreset selection with cluster-based methods. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2023, 167−178
17. Kumar A, Naughton J, Patel J M. Learning generalized linear models over normalized data. In: Proceedings of 2015 ACM SIGMOD International Conference on Management of Data. 2015, 1969−1984
18. Kaggle. The state of data science. See www.kaggle.com/kaggle-survey-2020 website, 2020
19. Psallidas F, Zhu Y, Karlas B, Interlandi M, Floratou A, Karanasos K, Wu W, Zhang C, Krishnan S, Curino C, Weimer M. Data science through the looking glass and what we found there. 2019, arXiv preprint arXiv: 1912.09536
20. Grinsztajn L, Oyallon E, Varoquaux G. Why do tree-based models still outperform deep learning on typical tabular data? In: Proceedings of the 36th International Conference on Neural Information Processing Systems. 2022, 37
21. The Apache Software Foundation. Apache arrow. See Arrow.apache website, 2016
22. ClickHouse. ClickHouse. See githubcom/ClickHouse/ClickHouse website, 2024
23. Apache Druid. Apache® druid. See druidapache.org/ website, 2024
24. MySQL. See www.mysql.com/ website, 2024
25. Depoutovitch A, Chen C, Chen J, Larson P, Lin S, Ng J, Cui W, Liu Q, Huang W, Xiao Y, He Y. Taurus database: how to be fast, available, and frugal in the cloud. In: Proceedings of 2020 ACM SIGMOD International Conference on Management of Data. 2020, 1463−1478
26. Ma Y, Xie S, Zhong H, Lee L, Lv K. HiEngine: how to architect a cloud-native memory-optimized database engine. In: Proceedings of 2022 International Conference on Management of Data. 2022, 2177−2190
27. Shen J, Zuo P, Luo X, Su Y, Gu J, Feng H, Zhou Y, Lyu M R. Ditto: an elastic and adaptive memory-disaggregated caching system. In: Proceedings of the 29th Symposium on Operating Systems Principles. 2023, 675−691
28. Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, et al. GPT-4 technical report. 2023, arXiv preprint arXiv: 2303.08774
29. Ren X, Zhou P, Meng X, Huang X, Wang Y, Wang W, Li P, Zhang X, Podolskiy A, Arshinov G, Bout A, Piontkovskaya I, Wei J, Jiang X, Su T, Liu Q, Yao J. PanGu-Σ: Towards trillion parameter language model with sparse heterogeneous computing. 2023, arXiv preprint arXiv: 2303.10845
30. Rojas J S. IP network traffic flows labeled with 75 apps. See Kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps website, 2018
31. Kohavi R. Census income-UCI Machine Learning Repository. See Archive.ics.uci.edu/dataset/20/census+income website, 1996
32. Bifet A, Ikonomovska E. The airlines dataset. See www.openml.org/d/1169 website, 2009
33. Tromp J. Connect-4- UCI Machine Learning Repository. See Archive.ics.uci.edu/dataset/26/connect+4 website, 1995
34. Moro S, Rita P, Cortez P. Bank marketing- UCI Machine Learning Repository. See Archive.ics.uci.edu/dataset/222/bank+marketing website, 2012
35. Raabe M. The black Friday dataset. See www.openml.org website, 2019
36. Mueller A. The diamonds dataset. See www.openml.org/data/download/21792853/dataset website, 2019
37. Taxi N Y C. New York city taxi tip prediction. See www.openml.org/d/44065 website, 2016
38. Group Mercedes Benz. Mercedes-Benz greener manufacturing. See Github.com/MezbanS/Mercedes-Benz-Greener-Manufacturing website, 2017
39. Khamis M A, Ngo H Q, Nguyen X, Olteanu D, Schleich M. Learning models over relational data using sparse tensors and functional

dependencies. ACM Transactions on Database Systems, 2020, 45(2): 7
40. Kadra A, Lindauer M, Hutter F, Grabocka J. Well-tuned simple nets excel on tabular datasets. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. 2021, 1832
41. Bej S, Davtyan N, Wolfien M, Nassar M, Wolkenhauer O. LoRAS: an oversampling approach for imbalanced datasets. Machine Learning, 2021, 110(2): 279−301
42. Kotelnikov A, Baranchuk D, Rubachev I, Babenko A. TabDDPM: modelling tabular data with diffusion models. In: Proceedings of the 40th International Conference on Machine Learning. 2023, 725
43. Feurer M, Klein A, Eggensperger K, Springenberg J T, Blum M, Hutter F. Efficient and robust automated machine learning. In: Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015, 2755−2763
44. Yakovlev A, Moghadam H F, Moharrer A, Cai J, Chavoshi N, Varadarajan V, Agrawal S R, Idicula S, Karnagel T, Jinturkar S, Agarwal N. Oracle AutoML: a fast and predictive AutoML pipeline. Proceedings of the VLDB Endowment, 2020, 13(12): 3166−3180
45. Li Y, Shen Y, Zhang W, Zhang C, Cui B. VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. The VLDB Journal, 2023, 32(2): 389−413
46. H2O.ai. Scalable AutoML in H2O-3 open source. See H2o.ai/platform/h2o-automl/ website, 2023
47. Patki N, Wedge R, Veeramachaneni K. The synthetic data vault. In: Proceedings of 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA). 2016, 399−410
48. Pedreira P, Erling O, Karanasos K, Schneider S, McKinney W, Valluri S R, Zait M, Nadeau J. The composable data management system manifesto. Proceedings of the VLDB Endowment, 2023, 16(10): 2679−2685
49. Wilhite D. GoogleSQL: A SQL language as a component. In: Proceedings of the 1st International Workshop on Composable Data Management Systems. 2022
50. Chattopadhyay B, Pedreira P, Agarwal S, Sun Y, Vakharia S, Li P, Liu W, Narayanan S. Shared foundations: modernizing Meta's data lakehouse. In: Proceedings of the 13th Conference on Innovative Data Systems Research. 2023
51. Begoli E, Camacho-Rodríguez J, Hyde J, Mior M J, Lemire D. Apache calcite: a foundational framework for optimized query processing over heterogeneous data sources. In: Proceedings of 2018 International Conference on Management of Data. 2018, 221−230
52. Soliman M A, Antova L, Raghavan V, El-Helw A, Gu Z, Shen E, Caragea G C, Garcia-Alvarado C, Rahman F, Petropoulos M, Waas F, Narayanan S, Krikellas K, Baldwin R. Orca: a modular query optimizer architecture for big data. In: Proceedings of 2014 ACM SIGMOD International Conference on Management of Data. 2014, 337−348
53. Pedreira P, Erling O, Basmanova M, Wilfong K, Sakka L, Pai K, He W, Chattopadhyay B. Velox: Meta's unified execution engine. Proceedings of the VLDB Endowment, 2022, 15(12): 3372−3384
54. Microsoft. Microsoft SQL server machine learning services. website, 2024
55. Karanasos K, Interlandi M, Psallidas F, Sen R, Park K, Popivanov I, Xin D, Nakandal S, Krishnan S, Weimer M, Yu Y, Ramakrishnan R, Curino C. Extending relational query processing with ML inference. In: Proceedings of the 10th Conference on Innovative Data Systems Research (CIDR 2020). 2020
56. Corporation I. IBM db2 machine learning. website, 2024
57. Li F. Modernization of databases in the cloud era: building databases that run like Legos. Proceedings of the VLDB Endowment, 2023, 16(12): 4140−4151
58. AP. SAP HANA predictive analysis library (PAL). See Help.sap.com website, 2024
59. Hellerstein J M, Ré C, Schoppmann F, Wang D Z, Fratkin E, Gorajek A, Ng K S, Welton C, Feng X, Li K, Kumar A. The MADlib analytics library: *or MAD skills, the SQL*. Proceedings of the VLDB Endowment, 2012, 5(12): 1700−1711
60. Del Buono F, Paganelli M, Sottovia P, Interlandi M, Guerra F. Transforming ML predictive pipelines into SQL with MASQ. In: Proceedings of 2021 International Conference on Management of Data. 2021, 2696−2700
61. Schule M, Lang H, Springer M, Kemper A, Neumann T, Gunnemann S. In-database machine learning with SQL on GPUs. In: Proceedings of the 33rd International Conference on Scientific and Statistical Database Management, SSDBM '21. 2021, 25−36
62. Olteanu D. The relational data Borg is learning. Proceedings of the VLDB Endowment, 2020, 13(12): 3502−3515
63. Gandhi A, Asada Y, Fu V, Gemawat A, Zhang L, Sen R, Curino C, Camacho-Rodríguez J, Interlandi M. The tensor data platform: towards an AI-centric database system. In: Proceedings of the 13th Conference on Innovative Data Systems Research. 2023
64. Ghorbani M, Shaikhha A. Demonstration of OpenDBML, a framework for democratizing in-database machine learning. Proceedings of the VLDB Endowment, 2023, 16(12): 3970−3973
65. Miao H, Li A, Davis L S, Deshpande A. Towards unified data and lifecycle management for deep learning. In: Proceedings of the IEEE 33rd International Conference on Data Engineering (ICDE). 2017, 571−582
66. Wang X, Dong X L, Meliou A. Data x-ray: a diagnostic tool for data errors. In: Proceedings of 2015 ACM SIGMOD International Conference on Management of Data. 2015, 1231−1245
67. Vartak M, da Trindade J M F, Madden S, Zaharia M. MISTIQUE: a system to store and query model intermediates for model diagnosis. In: Proceedings of 2018 International Conference on Management of Data. 2018, 1285−1300



Cheng Chen is now a PhD student at Renmin University of China, China. Currently he also works as an intern at the Database Innovation Lab of Huawei Cloud. His research interests are data-centric AI and DB for AI.

Wenlong Ma is a research scientist at the Database Innovation Lab of Huawei Cloud. He received his PhD degree from Institute of Computing Technology, Chinese Academy of Sciences, China. His major research area lies in database systems and AI.



Congli Gao is a research scientist at the Database Innovation Lab of Huawei Cloud, China. His major research area lies in database systems and AI.

Wenliang Zhang is the director of the Database Innovation Lab of Huawei Cloud, China. His major research area lies in big data management systems and cloud computing.

Kai Zeng is the Chief Architect of Huawei Cloud Data Warehouse Service. He also works as an adjunct professor in Yangtze Delta Region Institute, University of Electronic Science and Technology of China, China. His research interest lies in large scale data intensive systems.

Tao Ye is a director at Huawei Cloud Data Warehouse Service. He holds a PhD in Computer Science from Huazhong University of Science and Technology, China. His research interests lie in exploring the fundamental principles and algorithms of database kernels.

Yueguo Chen is a professor at School of Information, Renmin University of China, China. He received his PhD degree from National University of Singapore, Singapore. His research interests lie in database systems and interdisciplinary studies.

Xiaoyong Du is a professor at School of Information, Renmin University of China, China. He is the director of the Key Laboratory of Data Engineering and Knowledge Engineering (Ministry of Education). His research interests lie in database systems, big data analytics, and knowledge engineering.