AQUA: Automatic Collaborative Query Processing in Analytical Database

Yuchen Peng
Zhejiang University
zjupengyc@zju.edu.cn

Ke Chen*
Zhejiang University
chenk@zju.edu.cn

Lidan Shou
Zhejiang University
should@zju.edu.cn

Dawei Jiang
Zhejiang University
jiangdw@zju.edu.cn

Gang Chen
Zhejiang University
cg@zju.edu.cn

ABSTRACT

Data analysts nowadays are keen to have analytical capabilities involving deep learning (DL). Collaborative queries, which employ relational operations to process structured data and DL models to process unstructured data, provide a powerful facility for DL-based in-database analysis. The classical approach to support collaborative queries in relational databases is to integrate DL models with user-defined functions (UDFs) in a general-purpose language (e.g., C++) to process unstructured data. This approach suffers from sub-optimal performance as the opaque UDFs preclude the generation of an optimal query plan. A recent work, DL2SQL, addresses the problem of collaborative query optimization by first converting DL computations into SQL subqueries and then using a classical relational query optimizer to optimize the entire collaborative query. However, the DL2SQL approach compromises usability by requiring data analysts to manually manage DL-related data and tune query performance. To this end, this paper introduces AQUA, an analytical database designed for efficient collaborative query processing. Built on DL2SQL, AQUA automates translations from collaborative queries into SQL queries. To enhance usability, AQUA introduces two techniques: 1) a declarative scheme for DL-related data management, and 2) DL-specific optimizations for collaborative query processing, eliminating the burden of manual data management and performance tuning from the data analysts. We demonstrate the key contributions of AQUA via a web APP that allows the audience to perform collaborative queries on the CIFAR-10 dataset.

PVLDB Reference Format:
Yuchen Peng, Ke Chen, Lidan Shou, Dawei Jiang, and Gang Chen. AQUA: Automatic Collaborative Query Processing in Analytical Database. PVLDB, 16(12): 4006 - 4009, 2023.
doi:10.14778/3611540.3611607

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/dilab-zju/aqua.

*Ke Chen is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611607

1 INTRODUCTION

Due to the popularity of deep learning (DL) in many applications such as image classification and natural language processing [6], there is an increasing interest in the database community to develop collaborative query processing techniques that extend the query processing capabilities of relational databases by integrating DL models to process unstructured data. We define a collaborative query as a query that employs relational operations to process structured data and DL models to process unstructured data. The classical collaborative query processing technique, known as the loose-integration approach [3–5], adopts user-defined functions (UDFs) to integrate DL models. Suppose we have a CIFAR-10 dataset where each record is an image of a single object (e.g., bird, cat). Using the loose-integration approach, we store the dataset in a CIFAR table where structured information such as image creation time is stored in a standard column and the raw binary image data is stored in a special BLOB (Binary Large Object) column. To derive the count of each red object in the image created in 2022, the data analyst may submit the following collaborative query Q_1:

```
SELECT Object(C.image) as o, count(C.id) as c
FROM CIFAR as C
WHERE C.createDate >= `2022-01-01`
    and C.createDate <= `2022-12-31`
    and Color(C.image) = `red`
GROUP BY o
```

In Q_1, `Object()` and `Color()` are UDFs implemented in C++, which employ pre-trained DL models for the corresponding classification tasks. The loose-integration approach is easy to use. However, its query performance tends to be sub-optimal. This is because C++ UDFs are opaque to the database engine. As a result, the query optimizer cannot estimate the execution cost of these UDFs and generate optimized query plans for the collaborative query.

DL2SQL[6] is a recent work to address the optimization challenges introduced by opaque UDFs. The main idea is that, instead of storing raw image data in BLOB columns and using opaque UDFs to integrate DL models, DL2SQL retrieves raw image data from the CIFAR master table, transforms these raw data into features that can be processed by the DL model, and stores the features in a separate relational table (called a feature table). Similarly, DL2SQL stores the DL model's parameters in a set of parameter tables. With the feature and parameter tables in hand, the computation of the DL model (e.g., convolution) can be translated into standard subqueries on these tables. Finally, DL2SQL replaces the UDFs with the

resulting sub-queries and converts the entire collaborative query into a standard SQL query without UDFs for query optimization.

While DL2SQL tackles the query optimization challenge inherent to the loose-integration approach, it compromises on *usability*, a key strength of the loose-integration approach. First, DL2SQL requires the data analysts to manage by hand the relational feature tables and parameter tables. For instance, a single pre-trained ResNet model may require 152 tables to be created and manipulated, not considering the additional model variants fine-tuned toward the task-specific datasets. Although scripts may aid in generating these tables, it is difficult, if not impossible, to completely eliminate the burden of managing DL-related data without the support of a database engine. Second, to achieve the desired query performance, manual performance tuning is still essential, as current database optimizers are not designed specifically for DL computations. As an example, operator fusion, a common technique employed in machine learning (ML) to speed up inference by merging consecutive computations (e.g., convolution and its subsequent activation computation), is not implemented by any open-source databases we explored. Therefore, data analysts must tune SQL queries to implement such optimizations, which is a tedious task.

This paper presents AQUA, a high-performance analytical database for efficient collaborative query processing. AQUA builds upon DL2SQL and improves its usability via two extensions. First, it introduces a declarative scheme for DL-related data management. We represent a dataset consisting of structured and unstructured data as an instance of a collaborative relation, where structured data is stored under normal relational columns and unstructured data is stored in *virtual* columns. Data loaded into the *virtual* columns are automatically converted into features and stored in a separate table. Similarly, to handle unstructured data, the data analyst declares a user-defined function (called an inference function) to wrap a DL model. Unlike DL2SQL, our system stores features and model parameters as arrays, allowing tensor computations directly in the database, which is more efficient in terms of both storage and computation. Our novel DL management scheme completely eliminates the need to manually manage DL-related data.

Furthermore, we incorporate specific optimizations for collaborative query processing to eliminate the requirement for manual performance tuning. Our optimization consists of two parts. In the offline part, we accelerate feature data access by tensor compression and optimize the DL computational graph by operator fusion. During the online phase, we utilize a DL-aware optimizer that uses a novel cost model to generate optimized query plans.

In the rest of the paper, we present the design and architecture in Section 2, leaving the implementation details and empirical studies for a separate research paper. We demonstrate the main contributions of AQUA, including declarative DL data management and collaborative query processing, in Section 3.

2 SYSTEM OVERVIEW

This section provides an overview of AQUA in the following aspects: the foundations of collaborative query processing (Section 2.1), the system architecture and workflow (Section 2.2), and the core components of AQUA (Section 2.3).

2.1 Foundations

We now present our data model behind AQUA, and formulate the concept of inference function and collaborative query.

Data model. We model a collection of records where each record contains both structured and unstructured information as an instance of a collaborative relation. We will also call such a relation as a collaborative table (or cTable for short).

A collaborative relation D is a relation with $n > 1$ attributes $A_1, \cdots, A_n$, where each attribute A_i is a standard relational attribute or an unstructured attribute. We further require D to satisfy two additional requirements: 1) a collaborative relation must have at least one relational attribute, and 2) key constraints can only be imposed on relational attributes. That is, unstructured attributes cannot be part of the key or primary key of a collaborative relation. Listing 1 shows how to create the cTable CIFAR in AQUA. Note that the column *image* with data type jpeg is declared as virtual, meaning that the *image* column is an unstructured column that will be stored separately. For unstructured attributes, we currently only support images. Future versions of AQUA will support other unstructured data types, such as audio and text.

Listing 1: A cTable Example

```
CREATE cTable CIFAR(
    id int,
    createDate date,
    image jpeg virtual
    ...
) ;
```

Inference Function. Operations on collaborative tables are standard relational operations (e.g., projection, selection, and join). We introduce inference functions to enable these relational operations to process unstructured columns. An inference function (called nUDF) is a scalar function $f : x \xrightarrow{M} y$ that takes as input a value x in an unstructured column and applies a DL model M on x to produce a result string y as output (called label of x). That is, an inference function is a user-defined function that wraps a DL model to process unstructured data. In AQUA, the DL model M, supported in ONNX format, is produced by an external machine learning system. Listing 2 shows how to create a nUDF `Color()` from a DL model serialized in a file. A nUDF can be plugged into any relational operator that accepts a scalar value as an argument and enables that operator to handle unstructured columns.

Listing 2: A nUDF Example

```
CREATE INFERENCE FUNCTION Color(JPEG image)
FROM MODEL in file "cnn.onnx"
```

Collaborative Query. Utilizing inference functions, we formulate a collaborative query (CQ) as a relational expression, in which at least one of the relational operators takes inference functions within its arguments. Our CQ formulation is consistent with the loose-integration approach. That is, query composition using our approach provides the same usability as that approach. The query Q_1 in Section 1 is an example of a CQ query that is accepted by AQUA. To process CQs, the query processing engine of AQUA automatically translates a CQ into an SQL query and optimizes it for execution. Therefore, our approach obviates the need for manual query translation and optimization.

2.2 System Architecture

Figure 1 depicts the system architecture of the AQUA. To support declarative DL data management and inference functions, we extend a relational database with three components: a *model loader*, a *data loader*, and a *nUDF Processor*. We further enhance the traditional relational query processing module with two additional components: an offline optimizer and an online optimizer for collaborative query optimization.

- *Model Loader* is a component for loading model parameters into the array storage.
- *Data Loader* converts unstructured data into features and stores the generated features in the array store.
- *nUDF Processor* handles the DL model of the nUDF and transforms the computational graph of the DL model into a series of relational algebraic expressions for further optimizations.
- *Offline Optimizer* applies techniques such as tensor compression and operator fusion to optimize the storage of DL-related data and the process of tensor computations generated by the nUDF processor.
- *Online Optimizer* employs a novel cost model for online query processing.

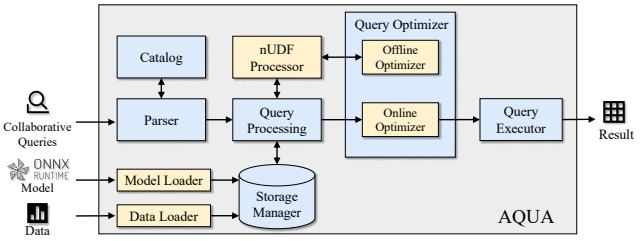


Figure 1: The Architecture of AQUA

2.3 Components

1) Model Loader. Acting as the entrance to manage DL models, the Model Loader enables users to upload arbitrary pre-trained models. We have integrated the ONNX Runtime[2] into AQUA to load DL models expressed in ONNX (or any format convertible to ONNX [1]). A serialized DL model comprises two parts: a computational graph that orchestrates the inference computation and the parameters in the model. We store the computational graph of the DL model as metadata, while the parameters of the model are maintained as tensor tables in an array store. We implement a compression scheme for extremely sparse parameters and store the compressed parameters within the tensor table.

2) Data Loader. The Data Loader manages DL-related data. With this component, users can insert records containing unstructured data efficiently. The Data Loader transforms unstructured data into features and stores the resulting features in a separate array-store table, preserving the original shape of the features. Compared to DL2SQL's approach of storing features in relational tables, our array storage approach saves more than 10 times the storage space and enables data transformation during query execution.

3) nUDF Processor. The nUDF Processor converts the DL computational graph in an inference function into a series of relational algebraic operations, including tensor computations implemented by relational operators. To do so, the nUDF processor traverses the computational graph of the DL model in topological order, which is typically a directed acyclic graph (DAG). Throughout the traversal, operations natively supported by the array store (e.g., tensor multiplication) are transformed into tensor computation functions. Unsupported operations are turned into relational algebraic expressions. To speed up certain tensor computations, We further implement specific data transformation functions, such as an *im2col()* function before convolution. This function extracts patches of the input tensor and rearranges these patches into a new tensor for efficient Generalised Matrix Multiplication (GeMM), providing a 6-fold improvement in processing performance compared to the trivial joins of tables.

4) Offline Optimizer. The Offline Optimizer applies three techniques (i.e., operator fusion, tensor fusion, and tensor compression) to optimize the DL-related data storage and relational expressions generated by the nUDF Processor. These techniques are briefly described below:

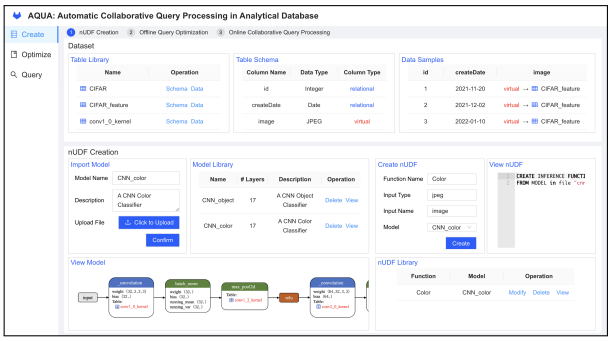
Operator Fusion. We implement operator fusion in AQUA, a key optimization in many state-of-the-art DL systems [7]. Operator fusion merges two or more DL operators into a single operator, allowing these operators to be executed together without any additional data transmission. We apply pre-defined fusion patterns (e.g. merging an activation operator into its preceding operator) to the DL computational graph and rewrite the relational algebra expressions for the subgraphs that match our fusion patterns.

Tensor Fusion. We adopt Tensor Fusion to combine a sequence of tensor operations into a single one for improving data locality. For example, in the DL model loaded in Listing 2, tensor computation with several consecutive multiplications on the input tensor $input\_fm$ is formulated as $input\_fm \times T_1 \times T_2 \times T_3$, where T_i is a kernel tensor. The multiplication of the parameter tensors can be fused into a single tensor T_{fused}, thereby reducing intermediate results and improving performance.

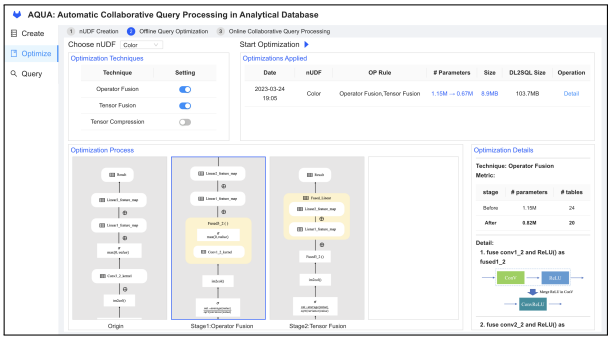
Tensor Compression. To save storage space and speed up the computation of sparse tensors whose proportion of non-zero elements is less than our pre-defined threshold, we perform tensor compression on these sparse tensors by storing only the non-zero elements and corresponding indices in compressed sparse row (CSR) format in the array storage.

5) Online Optimizer. The online optimizer leverages DL knowledge to generate an optimal execution plan for complex CQs, streamlining the previously cumbersome tuning process. We employ a cost-based optimization scheme to produce the optimized query plan of a CQ for online query processing. The details of the proposed cost model will be presented in a separate research paper. Briefly, we perform parallelized execution for certain tensor operations in a nUDF. We also combine relational operations in a nUDF with relational operations outside the nUDF in the CQ for optimization whenever possible. We further define *collaborative optimization hints* considering the dependency between the nUDFs and non-nUDF parts in a CQ. For example, if a nUDF is plugged in the select operator, the corresponding wrapped operations will be evaluated at last. After all the cost-based query transformations

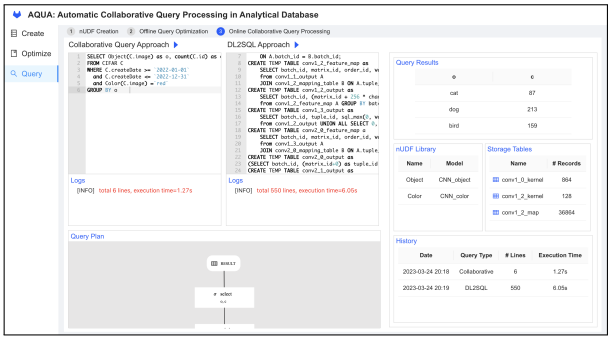
have been performed, we finally generate the optimized query plan for execution.



(a) The nUDF Creation Interface



(b) Offline Query Optimization



(c) Online Collaborative Query Processing

Figure 2: AQUA User Interface

3 DEMONSTRATION

We have developed a Web application to demonstrate the main contributions of AQUA. The interface of the Web application is presented in Figure 2. We will invite the audience to create a nUDF, compose collaborative queries, and participate in query optimization. The demonstration is planned as follows.

Environment. The web application and AQUA are hosted on a cloud server with 187GB of RAM, an NVIDIA Quadro P5000 with 16GB of graphics memory, and an Intel(R) Xeon(R) 2.10GHz 32-core CPU. We will use the CIFAR-10 dataset and pre-trained image classification DL models in our demonstration.

nUDF Creation. We demonstrate how to create a nUDF with ease of use (Figure 2(a)). As a nUDF takes a value in an unstructured column as input, we pre-load a cTable *CIFAR* containing image data. By examining the cTable schema and samples, users can gain a clear understanding of how a virtual image column is separately stored. To utilize DL models, users only need to upload their pre-trained models. The system can complete the model transformation, and display the established tables in milliseconds. With existing models, users simply provide a few necessary parameters, allowing the system to complete the remaining work.

Offline Query Optimization. Our interactive system empowers users to quickly conduct offline optimization on a nUDF (Figure 2(b)). A nUDF is selected, and the desired optimization techniques are set to apply in a staged progression. With each optimization step, the query operator graph is updated and presented to the user, where each node represents a physical structure (e.g., parameter table) or a relational operator. In particular, the changes compared to the previous graph are highlighted, implying a simplified relational algebra expression and reduced storage space for better query performance. For a deeper understanding, the user can click on the graph to view detailed information on the optimization process.

Online Collaborative Query Processing. The user can input a collaborative query with only a few lines to complete their analytical task (Figure 2(c)). After online optimization and query execution, the user can view the query plan produced by AQUA to understand how the operators beneath a nUDF interact with other relational operators. To evaluate our proposed collaborative query processing approach, our system allows the user to input hand-written DL2SQL queries in native SQL syntax. The performance metrics, including the execution time and the code length, are also displayed. In our approach, the query execution can be 5 times faster than the manual codes. We demonstrate that our approach provides better query performance, and significantly improves usability in terms of query composition compared to DL2SQL. Overall, our system is both cost-effective and convenient, and improves the productivity of data analysts who need collaborative query processing.

ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China (No.2022YFB3304100) and Fundamental Research Funds for the Central Universities. The authors are supported by the State Key Laboratory of Blockchain and Data Security, and Key Lab of Intelligent Computing Based Big Data of Zhejiang Province.

REFERENCES

[1] 2019. ONNX. http://onnx.ai.
[2] 2019. ONNX Runtime. http://github.com/microsoft/oxxnruntime.
[3] A. Fard, A. Le, G. Larionov, W. Dhillon, and C. Bear. 2020. Vertica-ML: Distributed Machine Learning in Vertica Database. In *SIGMOD*. 755–768.
[4] J. M. Hellerstein, . Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. 2012. The MADlib Analytics Library: Or MAD Skills, the SQL. *Proc. VLDB Endow.* 5, 12 (2012), 1700–1711.
[5] M. Jasny, T. Ziegler, T. Kraska, U. Roehm, and C. Binnig. 2020. DB4ML - An In-Memory Database Kernel with Machine Learning Support. In *SIGMOD*. 159–173.
[6] Q. Lin, S. Wu, J. Zhao, J. Dai, F. Li, and G. Chen. 2022. A Comparative Study of in-Database Inference Approaches. In *ICDE*. 1794–1807.
[7] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren. 2021. DNNFusion: Accelerating Deep Neural Networks Execution with Advanced Operator Fusion. In *SIGPLAN*. 883–898.