SQLFlow: An Extensible Toolkit Integrating DB and AI

Jun Zhou\textsuperscript{1,2} \quad Ke Zhang\textsuperscript{2} \quad Lin Wang\textsuperscript{2} \quad Hua Wu\textsuperscript{2} \quad Yi Wang\textsuperscript{2} \quad ChaoChao Chen\textsuperscript{1,*}

\textsuperscript{1}College of Computer Science and Technology, Zhejiang University, Hangzhou, China
\textsuperscript{2}Ant Group, Beijing, China
\textsuperscript{*}Corresponding Author

Abstract

Integrating AI algorithms into databases is an ongoing effort in both academia and industry. We introduce SQLFlow, a toolkit seamlessly combining data manipulations and AI operations that can be run locally or remotely. SQLFlow extends SQL syntax to support typical AI tasks including model training, inference, interpretation, and mathematical optimization. It is compatible with a variety of database management systems (DBMS) and AI engines, including MySQL, TiDB, MaxCompute, and Hive, as well as TensorFlow, scikit-learn, and XGBoost. Documentations and case studies are available at https://sqlflow.org. The source code and additional details can be found at https://github.com/sql-machine-learning/sqlflow.

Keywords: Machine Learning, AI Tasks, Model Training

1. Introduction

Nowadays, typical data science workflows consist of separate steps of processing data in a DBMS, exporting the processed data, and developing models in an Artificial Intelligence (AI) platform. This de facto standard workflow has several shortcomings: the data transfer incurs overhead and makes it hard to trace data, which is essential for compliance; development is slowed by imperative details in languages such as R and Python. From a practitioner point of view, the ideal system should efficiently execute AI algorithms in-place, be user-friendly and preserve the usability of exiting algorithms. And the integration of DBMS and AI algorithms provides the solution to meet the desired attributes (Makrynioti and Vassalos, 2019; Zhou et al., 2020; Villarroya and Baumann, 2020).

The integration of DBMS and AI algorithms could be realized by utilizing or extending DBMS to support AI algorithms. This could be done by direct SQL implementation (Ordonez, 2004; Ordonez and Pitchaimalai, 2009; Nagarjuna and Reddy, 2012; Marten and Heuer, 2017; Du, 2020; Giesser et al., 2021; Kaufmann et al., 2021; Blacher et al., 2022) or by extending DBMS (Luo et al., 2018; Karanasos et al., 2019; Jankov et al., 2020; Schüle...
et al., 2019; Dolmatova et al., 2020; Schüle et al., 2022). The progresses on implementation abstraction (Yuan et al., 2020), programming model (Jasny et al., 2020), and execution engine (Schleich et al., 2019) could significantly outperform existing AI frameworks. In addition, typical AI workloads such as model selections (Zhang et al., 2021) and even the whole AI pipeline (Schule et al., 2021) could be expressed and executed in SQL. This approach is user-friendly, performs AI tasks in the DBMS which avoids the data transfer and makes it easy to meet compliance requirement. Nevertheless, it is hard to reuse existing AI tools such as TensorFlow (Abadi et al., 2016), scikit-learn, and XGBoost (Chen and Guestrin, 2016).

An alternative approach for the integration of DBMS and AI algorithms is implementing AI algorithms in DBMS, mostly via user-defined functions (UDFs) (Hirn and Grust, 2021). Apache MADlib (Hellerstein et al., 2012) is the representative open-source example of this approach. Oracle Machine Learning (Oracle, 2022), Microsoft SQL Server AI Services (Microsoft, 2022), and Google BigQuery (Google, 2022) are notable commercial platforms adopting this approach. While this approach performs data processing in the DBMS and preserves the usability of existing AI tools, it is not as user-friendly since users still need to handle low-level details in UDFs.

Works above are contrasting paradigms of SQLFlow and similar platforms such as IntegratedAI (De Boe et al., 2020), Vertica-AI (Fard et al., 2020), Amazon RedShift AI (Amazon, 2022) to integrate DBMS and AI algorithms and each paradigm represents different trade-offs (Kumar et al., 2017). SQLFlow is also very user-friendly and preserves the usability of existing AI tools, at the cost of transferring data from DBMS to external platforms. While data transfer is needed in SQLFlow, SQLFlow could support data tracing since it builds an end-to-end workflow and is able to manage data throughout the workflow.

2. Overview of SQLFlow

SQLFlow is developed so practitioners could build end-to-end AI pipelines with just a few lines of SQL code. To this end, SQLFlow extends SQL syntax to support AI tasks including training, prediction, model evaluation, model explanation, custom jobs, and mathematical
SQLFlow: An Extensible Toolkit Integrating DB and AI

It also offers data processing and feature engineering capabilities aside from AI functionalities. SQLFlow supports various database systems like MySQL, MariaDB, TiDB (Huang et al., 2020), Hive, MaxCompute (Alibaba, 2023) and many machine learning toolkits like TensorFlow, Keras, XGBoost. SQLFlow has been adopted in practical applications by DiDi (Wang et al., 2020), Alibaba, and Ant Group over the years. For example, Xie et al. (2019) developed a SQLFlow-based deep neural network (DNN) classification model and applied it to taxi coupon distribution.

**Key Design Considerations**

SQLFlow adheres to the following design principles: 1) Rather than being limited to specific SQL engines, it should be compatible with a wide range of DBMS. 2) Rather than only prototyping, it should democratize ML by letting SQL practitioners build models using existing AI tools and SQL skills. 3) Rather than embedding Python or R code in SQL statements, it should allow users to construct and execute ML workflows using SQL queries. 4) Rather than being imperative and expressing how the computation is to be achieved, it should use the declarative paradigm to describe what the outcome of the computation will be.

**SQLFlow Architecture**

SQLFlow turns lines of SQL code into a complete AI pipeline by invoking Extended Syntax Parse (ESP), Code Generator (CG) and Submitter, shown in Figure 1. First, the ESP decodes the input SQL program using a collaborative parsing algorithm. For standard SQL statements, ESP forwards them to corresponding DB engines. For extended SQL statements supporting AI functionalities, such as TRAIN, PREDICT, EXPLAIN, and MAXIMIZE, ESP uses SQLFlow extension parser to handle them. As a result, SQLFlow will understand “normal statements” of SQL dialects, as well as those SELECT statements followed by SQLFlow extension clauses. After the input SQL statements are parsed, the CG verifies data schema with the help of SQL engine and produces the Python program, which includes information required by AI engines such as IO, dataset, and model structure. Finally, the Submitter acts as a pluggable translator, generating Job Descriptions that are tailored to various AI engines. The AI job is either run locally or submitted to a distributed computing service through it.

SQLFlow provides an interface for user interaction using a client-server architecture. The interaction with the end user is handled in the client by the “MagicCommand” module implemented in Go language. All modules in Figure 1 are included in “SQLFlow Server”. “MagicCommand” and “SQLFlow Server” communicate via gRPC. The detailed architecture is provided here (Contributors, 2019).

3. SQLFlow Use Case and Performance

**Classify Iris Dataset Using DNNClassifier**

There are four features and one label in the Iris dataset (Fisher, 1936). Each feature is represented by a float number. The label is an integer with the values 0, 1, and 2. The SQLFlow training statement is shown as Program 1. The trained DNN model is saved into table models.my\_dnn\_model. The following results are obtained: \{’accuracy’: 0.72, ’loss’: 0.5695601\}.

```bash
-- Program 1: Train a DNNClassifier on the Iris dataset --
SELECT * FROM iris.train
TO TRAIN DNNClassifier
```
Performance Comparisons We also compare the performance of NumPy, HyPer (Kemper and Neumann, 2011; Neumann et al., 2015), PostgreSQL, and SQLFlow (with MySQL) for batch gradient descent-based logistic regression (LR) using identical tests as (Blacher et al., 2022). We use a machine with an Intel Xeon E5-2650 32-core (64 threads) processor, 131 GB of RAM, and Ubuntu 20.04.4 LTS. We measure performance in terms of the average time per iteration when running the LR, and verify that all implementations achieve a training accuracy of 0.99. Figure 2(a) shows that SQLFlow, HyPer (database-friendly implementation), and NumPy achieve top three rankings on each of the three different synthetic datasets when only training time is considered, while the performance on PostgreSQL is low compared to others (out of memory when sample size $\geq 1\,000\,000$). This demonstrates the AI engine’s good training performance. Figure 2(b) shows that when data IO is considered, HyPer’s in-memory database feature and database-friendly SQL mappings make it the least time consuming in terms of IO and help it achieve the highest performance. SQLFlow and HyPer COO (coordinate format) are comparable. NumPy spends more than 90% of time parsing input texts, negatively impacting its performance. This shows that optimizing data IO is still a promising topic in data systems.

We also test the multi-threaded (32 threads) performance with 100 features and 1 000 000 examples and we observe that HyPer outperforms SQLFlow by a margin of 25%. NumPy and SQLFlow have comparably performance and both outperform HyPer COO by 15%. Using SQL for ML algorithms puts computations near to the data and can thus achieve high performance, which echoes the findings of (Blacher et al., 2022).

4. Conclusion

In this paper, we introduced SQLFlow, an extensible toolkit integrating DB and AI so practitioners could build end-to-end AI pipelines with just a few lines of SQL code. As an open-source software, it contributes to the overall effort of integrating DB and AI. Future directions include accelerating data IO, supporting more frameworks and more functionalities such as privacy computing (Zhou et al., 2022; Fang et al., 2020).
Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No.72192823 and No.62172362).
References


Joe Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, et al. The madlib analytics library or mad skills, the SQL. *arXiv preprint arXiv:1208.4165*, 2012.


