Integrating Machine Learning into SQL with Exasol

Christoph Großmann[1], Johannes Schildgen[1]

[1]*Faculty of Computer Science and Mathematics, Regensburg University of Applied Sciences*

Abstract

This paper introduces a novel, extendable, no-code framework for integrating machine-learning algorithms into SQL using the Exasol database. The framework combines the strengths of the high-performance, parallel-processing analytical Exasol database with the flexible and sophisticated machine learning algorithms of the Python library Scikit-Learn, while providing a seamless integration into SQL. This paper explores the technical background, the concept, and the implementation of the framework. The CREATE MODEL command for creating a machine learning model and the PREDICT function for prediction using a pre-trained model are discussed in detail. The main contributions of the framework are its seamless integration into SQL, scalability, and leveraging of existing database infrastructure. An overview of related work is also given.

Keywords

Machine Learning, SQL, Database, Exasol

1. Introduction

The rapid growth of the volume of data in recent years has presented both significant challenges and opportunities for organizations across various industries. To extract valuable insights from large datasets, there is an increasing demand for efficient and scalable approaches to data processing and analysis [1, 2]. Traditional relational database management systems have long served as the backbone for data storage and retrieval, with SQL as the most used language for interacting with these systems. However, the complexity and volume of data have spurred the demand for integrating machine-learning (ML) capabilities directly into SQL. This enables advanced analytics and predictive modeling while the complexities of data transfers and ETL processes are handled by the database system [3].

This paper introduces an extendable framework for in-database ML, leveraging the power of Exasol, a high-performance, parallel-processing analytical database [4]. The framework extends the capabilities of Exasol's SQL engine by incorporating the Python ML library Scikit-Learn. Thus, the familiar and user-friendly nature of SQL is combined with the flexibility and sophistication of ML. This bridges the gap between traditional SQL-based analytics and the realm of ML, empowering users to seamlessly develop and deploy ML models directly within the database environment. Furthermore, we propose the `CREATE MODEL` and the `PREDICT` function. The `CREATE MODEL` statement allows us to train ML models in the database, for example, a model predicting the salary of an employee. Let us name the model `model` and use a table called

M. Leyer, J. Wichmann (Eds.): Proceedings of the LWDA 2023 Workshops: BIA, DB, IR, KDML and WM. Marburg, Germany, 09.-11. October 2023, published at http://ceur-ws.org

christoph.grossmann@online.de (C. Großmann); johannes.schildgen@oth-regensburg.de (J. Schildgen)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3630/LWDA2023-paper7.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

employee as our data source. Additionally, we specify the prediction target or label salary and the features position and the birthyear used to determine the label.

```
CREATE MODEL "model" ON employees PREDICT (salary) USING ("position", birthyear);
```

We can use the created model to predict the salaries of employees. We again use the position and birthyear of the employee table as features and predict the missing salary entries. The prediction result is shown in Table 1. This example is elaborated on further in the following.

```
SELECT name, "position", birthyear, PREDICT "model" USING ("position", birthyear)
FROM employees WHERE salary IS NULL;
```

Table 1
Example Result for the Prediction of the Salary of Employees

name	position	birthyear	salary
Emily Wilson	Software Engineer	1989	56951.48
John Anderson	Sales Associate	1992	51762.99
⋮	⋮	⋮	⋮

Our framework facilitates efficient resource utilization by capitalizing on Exasol's parallel processing capabilities and ETL pipeline, enabling scalability to handle large datasets and complex analytical workloads. Furthermore, in-database exploratory data analysis is simplified by the availability of no-code ML functionality. Finally, this integration eliminates the need for data movement between different systems, reducing latency, and enhancing the overall efficiency of the analytics workflow.

This paper provides an overview of the technical background, related work, the concept, and the implementation of the framework. It closes with a discussion and a conclusion.

2. Technical Background

Exasol is a proprietary distributed relational analytical database management system. It runs on the Linux-based operating system (OS) *ExaCluster OS*, which provides a runtime environment and a storage layer for the database. Exasol being a cluster of nodes allows it to execute queries in parallel and makes it cloud-ready. Furthermore, Exasol uses column-oriented storage and in-memory processing. For data unfit to be stored in the database, Exasol provides the file system *BucketFS* [4, 5]. We chose Exasol since it provides the necessary tools for extending the database and SQL for ML, and opportunities for improving ML processes using database features. The tools needed for our framework are a query rewriter and a way to execute code written in a scripting language, preferably Python, inside the SQL pipeline. The opportunities for improving ML processes are massively parallel processing using the parallel processing infrastructure of Exasol clusters and optimization through automatic query optimization.

The features of the Exasol database our framework relies upon are introduced in the following. The aforementioned *BucketFS* is a plain file system and can be accessed using a *HTTPS* interface. Data stored in BucketFS is replicated over all nodes of a cluster. Eventual consistency is guaranteed [6, 7]. Our framework uses BucketFS for ML model storage. *Script-language container* are the basis for extending the database for ML. They are Docker containers and contain a complete Linux installation with all packages required to execute code in scripting languages like Python, R, or Java. A set of pre-built containers is distributed by Exasol. Nevertheless, it is possible to build a custom container. Script-language containers are stored in BucketFS [8, 9]. *User-defined function (UDF) scripts* provide the interface for extending the SQL pipeline with the script languages provided by script-language containers [10]. These scripts are executed through SQL, pass their input data to a program written in another language and executed in an instance of the currently active script-language container, and then pass the results back to the database. Since UDF scripts are executed within the SQL pipeline, they can make use of database parallelization [11]. UDF scripts already make it possible to extend the Exasol database with ML. *Scripting programs* combine SQL with the scripting language Lua. Thus, they can execute multiple successive SQL statements and provide control structures [12]. *Preprocessor scripts* are query rewriters that analyze and rewrite all SQL statements before they are processed. Thus, they can convert unsupported SQL constructs into statements supported by the SQL parser. They can be seen as specialized scripting programs [13].

We chose Python since it is a popular language for ML providing many popular libraries like Scikit-Learn, PyTorch, and TensorFlow [14, 15]. This decision does not limit our framework to Python. Support for ML libraries written in other script languages can be added in the future. The framework currently integrates ML algorithms of the Scikit-Learn library due to its ease of use, performance, and standardized API [16, 16]. Exasol provides Python libraries for accessing the database [17] and BucketFS [18, 19].

3. Related Work

Exasol's developers and community provide information and many examples for creating UDF scripts for ML and data analysis tasks [20, 21, 22]. These scripts each only handle one specific use case, while our framework provides a generic solution. Furthermore, Exasol provides an extension to use pre-trained ML models via the Transformers API [23].

There are several other approaches to integrate ML into database systems. Among these, our framework stands out through its focus on smooth SQL integration. The approach closest to our framework is the *Apache MADlib* analytics library, which uses user-defined functions and aggregates to implement in-database ML algorithms [24, 25]. Many well-known database vendors have solutions for integrating ML into the database like Oracle [26] and IBM [27]. But there are also many different approaches by the scientific community. Schule et al. propose a complete ML pipeline using recursive tables while training models on GPUs. [28] Makrynioti et al. introduce *sql4ml*, a framework for translating objective functions written in SQL into an equivalent TensorFlow graph [29]. Dolmatova et al. introduce *relational matrix algebra (RMA)*, which seamlessly integrates linear-algebra operations into the relational model [30]. Kersten et al. propose *SciQL*, a SQL-based query language with both tables and arrays as first-class citizens [31, 32]. Apart from these approaches, other approaches that start with a high-level statistical programming language and aim to build a parallel processing infrastructure using database systems exist [33, 34, 35, 36].

4. Concept

An important part of our framework is the convenient, well-integrated syntax for handling ML models. ML models are handled as database objects stored in system tables and with support for DDL commands. These commands include `CREATE`, `RENAME`, `DROP`, `ALTER`, and `REPLACE`. The `CREATE` command creates a new model and trains it. The `RENAME` command renames an existing model and all associated files. The `DROP` command deletes an existing model and all associated files. The `ALTER` command allows for changing the parameters of an existing model. The `REPLACE` command replaces an existing model with the newly trained one. In addition to these commands, we introduce three commands unique to ML models: `IMPORT`, `RETRAIN`, and `PREDICT`. The `IMPORT` command creates the metadata for an ML model that already exists in BucketFS. The `RETRAIN` command retrains the specified ML model with the updated data in the source table or view. An error is thrown, if the source table or view is missing columns needed for the training of the model. The `PREDICT` function uses a previously trained ML model and the specified input data to predict values.

In the following, we present the syntax of the `CREATE` and the `PREDICT` command. Additionally, examples are given for better understanding. These examples use the "employees" table shown in Table 2. The table contains the name, position, year of birth, and salary of different employees. Some of the employee salaries are `NULL` and thus unknown. We will create an ML model to predict these values.

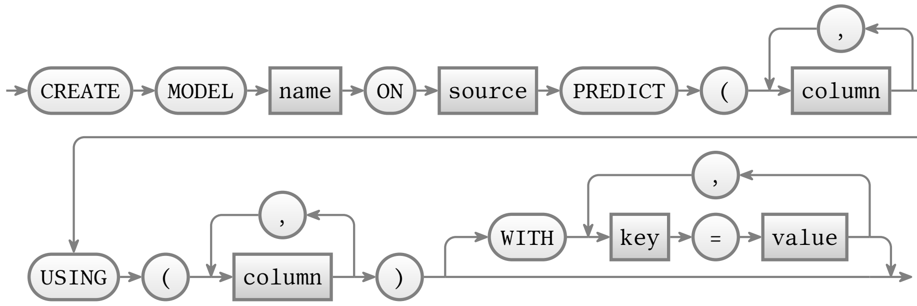
Table 2
Employee Table containing the Name, Position, Year of Birth, and Salary of Employees

name	position	birthyear	salary
Jacob Taylor	Software Engineer	1995	48446.32
Emma Anderson	Software Engineer	1988	57854.25
Daniel Young	Sales Associate	1992	50888.03
Ava Thompson	Sales Associate	1993	50106.14
Emily Wilson	Software Engineer	1989	NULL
John Anderson	Sales Associate	1992	NULL
⋮	⋮	⋮	⋮

4.1. Model Creation

The syntax for creating an ML model using our framework is shown in Figure 1. The `name` determines the unique object identifier of the model. This identifier is needed for all further interactions with the model, like using it for predictions. The `source` identifier determines which table or view is used as the input for training the model. Thus, the table or view has to exist and preferably contain data. If the table or view is empty, the model has to be retrained after the data is inserted. The `column` specifiers in the `PREDICT` clause determine which columns of the source table or view are the labels of the model and thus contain the values to be predicted. The `column` specifiers in the `USING` clause determine which columns of the source table or view are the features of the model and thus contain the values that can be used to predict the labels. The WITH clause allows for setting additional parameters using key-value pairs. These parameters can be used to determine the output type of the model, to specify the ML algorithm to be used, and to pass additional settings to the algorithm. Examples of output types are classification and regression. In case no output type is determined using the WITH clause, regression is assumed if all labels are of the data type DOUBLE PRECISION (or its aliases DOUBLE, FLOAT, NUMBER, and REAL). Otherwise, classification is assumed as the output type.

Figure 1: SQL Syntax for Machine-Learning Model Creation



As an example, we create a model "sal", which uses the employee table as its source. The label to predict is the salary and the features are the position and the birthyear of employees. Furthermore, we specify the model to use the 'DecisionTreeRegressor' function, which determines the output to be a regression. Additionally, we specify a maximum depth of 64 for the created decision tree. The query to create the specified model is the following:

```
CREATE MODEL "sal" ON employees PREDICT (salary) USING ("position", birthyear)
WITH 'Function' = 'DecisionTreeRegressor', 'max_depth' = 64;
```

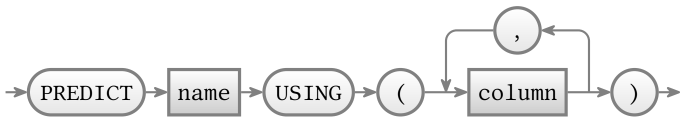
The source table for this model contains some NULL values in the salary column. For training, only tuples without NULL values in labels are used. After the training, the model is stored in BucketFS for future use.

4.2. Prediction

The syntax for the variadic function PREDICT is shown in Figure 2. The name corresponds to the identifier of an already existing ML model. The output of the prediction is one set of labels for each input row. These labels correspond to the trained labels, having the same name and a compatible data type. The column parameter list determines which columns serve as the features of the model. The number and position of these features have to match the number and position of the features used in the training step. The data types of the features have to be compatible with the features used for training, while the name of the features is of no importance. Furthermore, a prediction does not have to use the same table that was used for training.

As an example, we use the previously trained ML model to predict the salary of employees, for whom this information is missing. We again use the employee table as source as well as

Figure 2: SQL Syntax for Machine-Learning Model Prediction



`position` and `birthyear` as features. Since salary is a currency value, we format it to have two decimal places by casting it as a `DECIMAL(14,2)`. Furthermore, we rename the result of the prediction to `pred_salary` to avoid duplicate column names. The query for this prediction is the following:

```
SELECT name, "position", birthyear, salary AS original_salary,
PREDICT "sal" USING ("position", birthyear) FROM employees;
```

The result of the query is shown in Table 3. For each employee, this information is predicted based on the data of employees with valid salary information. To persist the prediction, `INSERT`, `CREATE TABLE AS`, or `UPDATE` queries can be used.

Table 3
Example Result for the Prediction of the Salary of Employees

name	position	birthyear	original_salary	salary
Jacob Taylor	Software Engineer	1995	48446.32	48436.18
Emma Anderson	Software Engineer	1988	57854.25	58103.16
Daniel Young	Sales Associate	1992	50888.03	51762.99
Ava Thompson	Sales Associate	1993	50106.14	49971.14
Emily Wilson	Software Engineer	1989	NULL	56951.48
John Anderson	Sales Associate	1992	NULL	51762.99
⋮	⋮	⋮	⋮	⋮

5. Implementation

The implementation of the framework works with both the single-node "Community Version" [37] as well as proprietary cluster versions. To avoid a library version mismatch within the default script-language container, the Exasol script-language container version 8.0.0 is used.

5.1. Available Algorithms

Our framework currently supports five algorithms of the Python library Scikit-Learn. These supported algorithms are listed in Table 4. All parameters of the algorithms are supported by our framework. In the future, other algorithms of the Scikit-Learn framework and other frameworks, even ones written in other programming languages, will be supported by our framework.

Table 4
Machine-Learning Algorithms of the Scikit-Learn Library currently supported by the Framework

Namespace	Algorithm	Output Type
`ensemble`	`RandomForestClassifier`	Classification
`linear.model`	`LinearRegression`	Regression
`svm`	`SVR`	Regression
`tree`	`DecisionTreeClassifier`	Classification
`tree`	`DecisionTreeRegressor`	Regression

5.2. Framework Layers

To process commands interacting with ML models, our framework employs several layers. These layers are visualized in Figure 3. The first layer processing incoming queries consists of preprocessor scripts. This layer converts the custom SQL syntax of the framework to scripting programs, which are the second layer of the framework. The scripting programs handle the metadata of the model and call the UDF scripts, which are the third, final layer. UDF scripts handle the calling of the actual ML functionality provided by the Python library Scikit-Learn. Furthermore, the UDF scripts handle the storage and loading of models to and from BucketFS. In the following, the implementation is discussed in further detail.

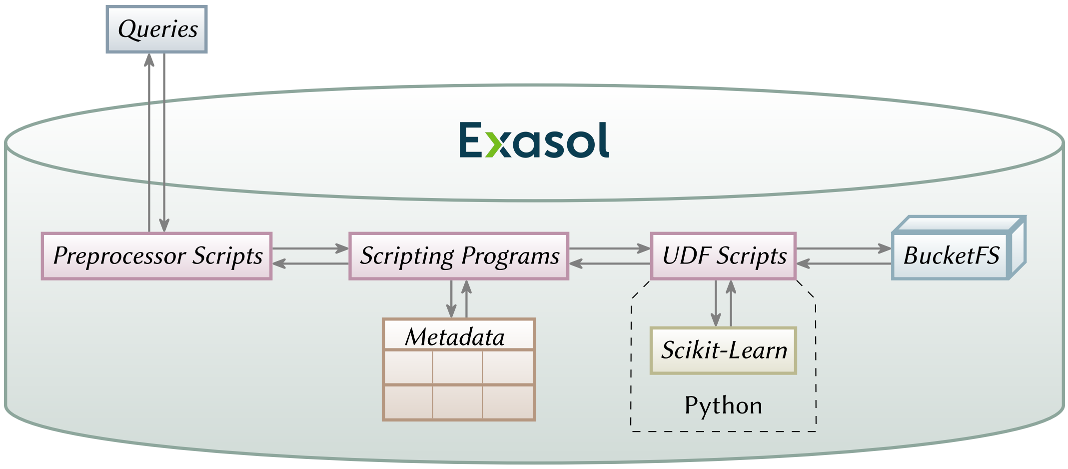


Figure 3: Layers and Elements of the Machine-Learning Framework

For each algorithm supported by the framework, two UDF scripts need to be created. The first UDF script handles model creation and training. For this purpose, the script takes the name of the model, settings for the algorithm, a list of features, and a label. The number of labels is only restricted in the current implementation of the framework. The script passes the settings, features, and labels to the algorithm, starts the training of the model, and finally stores the model in BucketFS. Since the currently implemented algorithms cannot process character strings as input or output, it is necessary to map these to integers. This is handled by the UDF scripts and a mapping dictionary. As an example, let us assume the key `'Software Engineer'` is mapped to the integer value `1`. On prediction, each input instance of `'Software Engineer'` would also be mapped to `1`. In the case of classification, each output instance of `1` would be mapped to `'Software Engineer'`. Mapping dictionaries are created before model creation and stored alongside the model in BucketFS. In a future version of the framework, this mapping functionality will be replaced by in-database mapping tables.

As an example for creating a model using UDF scripts, we use the statement created by the preprocessor when processing the following `CREATE MODEL` statement.

```
CREATE MODEL "sal" ON employees PREDICT (salary) USING ("position", birthyear)
WITH 'Function' = 'DecisionTreeRegressor', 'max_depth' = 64;
```

Since UDF scripts are ML-function-specific, the function to be used has to be determined before the execution. In our case, the function parameter set to `'DecisionTreeRegressor'` means that the decision-tree-regressor function of the Scikit-Learn library is selected. The preprocessor script rewrites the `CREATE MODEL` statement into the following statement.

```
SELECT ML.sklearn_tree_DecisionTreeRegressor_train
    ('sal', '{"model_params":{"max_depth":64}}', "position", birthyear, salary)
FROM employees WHERE salary IS NOT NULL ORDER BY RANDOM();
```

The second UDF script handles prediction. The parameters of the script are the name of the model, settings, the row identifier, and the features used for predicting labels. The features passed to the prediction script have to match the number, position, and data type of the features which were used to create the model. The script loads the model and all associated mapping dictionaries from BucketFS and passes the features to the model for prediction. The predicted labels are then combined with the internal row identifiers by position and the set is returned. An important restriction of Exasol is that no other expression can be present in the `SELECT` clause when calling a UDF script that emits a table. To solve this problem, we use common table expressions.

As an example for prediction with a pre-trained model using UDF scripts, we use the statement created by the preprocessor when processing the following statement using the `PREDICT` function.

```
SELECT name, "position", birthyear, salary AS original_salary,
PREDICT "sal" USING ("position", birthyear) FROM employees;
```

The prediction UDF script is determined using the stored model metadata. The preprocessor script rewrites the previous statement into the following statement.

```
WITH pred AS (SELECT ML.sklearn_tree_DecisionTreeRegressor_predict
    ('sal', '', ROWID, "position", birthyear) FROM employees)
SELECT e.name, e."position", e.birthyear, e.salary AS original_salary, p.label
AS salary FROM employees e JOIN pred p ON e.ROWID = p.identifier GROUP BY IPROC();
```

As already discussed, scripting programs combine the handling of metadata with the execution of ML functionality. The metadata of the framework could theoretically also be managed inside of UDF scripts, but this would necessitate the use of a database connector. This would defeat the purpose of executing ML in-database since an external connection to the database is needed. Scripting programs take the parameters extracted by the preprocessor scripts, as input. The preprocessor script takes incoming queries containing the custom syntax of our framework, splits them into tokens, and extracts the parameters of clauses of statements. In case a new ML model is created, the scripting programs choose the ML function to be used according to the settings the user provided in the `WITH` clause. If multiple ML functions fit the given settings, the function with the lowest priority value is selected. When training an ML model, all settings relevant to the model are passed to the UDF script. In case an existing ML model is needed, the scripting programs determine the function used to create the model through the metadata of the model. When executing predictions, the scripting programs use either Exasol's internal row ID of the source table `ROWID` or the `ROW_NUMBER` function as the row identifier for data passed to the prediction UDF script. The `GROUP BY IPROC()` clause groups the rows by the node they are stored on. Thus, each row is processed locally on the node it is stored on and only the results are transmitted over the network.

5.3. Tracked Metadata

The metadata of the framework is stored in two tables. The `ML.Algorithm` table contains information about the algorithms integrated into the framework, The information contained about algorithms includes their algorithm type, their output type, the module or library it is contained in, and the function it references to. The `ML.Model` table contains information about all created ML models created by the user. The information about models includes the algorithm used to train the model, its name, the source table or view, the features used during training, the labels used during training, and the settings used during training. This information is used for `PREDICT` or `RETRAIN` statements, for example.

6. Discussion

Our framework is an extendable, no-code integration of ML into SQL while employing Exasol's distributed, parallel processing capabilities and ETL pipeline in addition to Scikit-Learn's flexible and sophisticated ML algorithms. The main contributions of the proposed framework lie in its seamless integration into SQL, scalability, and leveraging of existing database infrastructure. The integration of ML into SQL also benefits users familiar with SQL, such as data scientists, analysts, and database administrators. The framework enables them to leverage their existing SQL skills, making the transition to advanced analytics and ML more accessible. Furthermore, it is also possible to export and import models, since all ML models of the framework are stored in BucketFS.

However, certain restrictions need to be addressed. Firstly, the prediction phase currently uses UDF scripts. In future work, the prediction step will be changed to use preprocessor scripts. Other future work includes replacing the mapping directories with mapping tables in the database. Furthermore, enabling more than one possible label is also future work. The `WITH` clause is currently restricted to exclusively textual values. Moreover, when using a model the version of the used libraries has to match the versions of the libraries used for creating the model. This can be achieved by using the same script-language container that was used for the model creation. Currently, the user of the framework has to activate the correct script-language container for each model. The automation of this process is also future work. Future work also includes the extension of the framework with additional algorithms of the Scikit-Learn library and other ML libraries written in Python or other programming languages. Future directions for our framework include distributed training, incremental model training, sample weights, model statistics, explainability functions, and data preparation.

In comparison to other approaches, our framework stands out through its smooth SQL integration. Furthermore, our framework has the advantage of employing the well-established ML library Scikit-Learn. However, by employing third-party libraries, our framework has a disadvantage compared to approaches implementing ML algorithms directly in SQL. Examples of approaches like this are Apache MADlib [24] and Oracle Machine Learning [26]. No efficiency and speed comparisons between these approaches and our framework have been done yet.

In comparison to traditional ML approaches involving data movement between databases and separate analytics platforms, the proposed framework offers advantages in terms of reduced data transfer, improved performance, and enhanced scalability. These advantages are all achieved by using the database as the singular platform for data storage and analysis.

7. Conclusion

This paper introduced a novel, extendable, no-code framework to integrate ML into SQL with Exasol. This framework bridges the gap between traditional SQL-based analytics and ML, empowering users to perform advanced analytical tasks directly within the database environment. We introduced Exasol, a high-performance, parallel-processing analytical database, and its features relevant to the framework. These features include scripting programs, preprocessor scripts, UDF scripts, script language containers, and the file system BucketFS. Furthermore, we discussed related work in the form of other approaches for ML with Exasol and other frameworks for in-database ML. The concept for the framework was introduced while discussing the syntax of the `CREATE MODEL` command for creating a new ML model and the `PREDICT` function for prediction using a pre-trained model in detail. The implementation of the framework consists of three layers: preprocessor scripts, scripting programs, and UDF scripts. Each layer provides a part of the complete functionality to translate incoming queries and execute the required ML functionality. Additionally, metadata for ML models is tracked in tables. The current restrictions of the framework and solutions were discussed. In comparison to other frameworks, our framework stands out with its seamless integration into SQL but is probably outshone regarding efficiency by frameworks re-implementing ML directly in the database. The main contributions of our framework are its seamless integration into SQL, scalability, and leveraging of existing database infrastructure.

The framework was initially created during the master's thesis "Extending SQL for Machine Learning" [38]. The source code is freely available at https://github.com/christoph-grossmann/Exasol_DB_ML_Framework.

References

[1] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J. S. Rellermeyer, A survey on distributed machine learning, ACM Comput. Surv. 53 (2020). URL: https://doi.org/10.1145/3377454. doi:10.1145/3377454.

[2] I. H. Sarker, Machine learning: Algorithms, real-world applications and research directions, SN Computer Science 2 (2021) 160. URL: https://doi.org/10.1007/s42979-021-00592-x. doi:10.1007/s42979-021-00592-x.

[3] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, K.-L. Tan, Database meets deep learning, ACM SIGMOD Record 45 (2016) 17–22. URL: https://doi.org/10.1145%2F3003665.3003669. doi:10.1145/3003665.3003669.

[4] Exasol, Exasol website, 2023. URL: https://www.exasol.com/.

[5] Exasol, Exasol white papers, 2023. URL: https://www.exasol.com/resource-type/white-papers/.

[6] Exasol, BucketFS client, 2023. URL: https://github.com/exasol/bucketfs-client.

[7] Exasol, BucketFS database access, 2023. URL: https://docs.exasol.com/db/latest/database_concepts/bucketfs/database_access.htm.

[8] Exasol, Exasol script languages, 2023. URL: https://github.com/exasol/script-languages-release.

[9] Exasol, Adding new packages to existing script languages, 2023. URL: https://docs.exasol.com/db/latest/database_concepts/udf_scripts/adding_new_packages_script_languages.htm.

[10] S. Mandl, O. Kozachuk, J. Graupmann, Bring your language to your data with EXASOL, in: Datenbanksysteme für Business, Technologie und Web, 2017.

[11] Exasol, UDF scripts, 2023. URL: https://docs.exasol.com/db/latest/database_concepts/udf_scripts.htm.

[12] Exasol, Scripting programs, 2023. URL: https://docs.exasol.com/db/latest/database_concepts/scripting.htm.

[13] Exasol, SQL preprocessor, 2023. URL: https://docs.exasol.com/db/latest/database_concepts/sql_preprocessor.htm.

[14] A. Nagpal, G. Gabrani, Python for data analytics, scientific and technical applications, 2019 Amity International Conference on Artificial Intelligence (AICAI) (2019). doi:10.1109/AICAI.2019.8701341.

[15] S. Raschka, J. Patterson, C. Nolet, Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence, Information 11 (2020). URL: https://www.mdpi.com/2078-2489/11/4/193. doi:10.3390/info11040193.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-Learn: Machine Learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[17] Exasol, Pyexasol, 2023. URL: https://docs.exasol.com/db/latest/connect_exasol/drivers/python/pyexasol.htm.

[18] Exasol, Exasol BucketFS Utils Python, 2023. URL: https://github.com/exasol/bucketfs-utils-python.

[19] Exasol, Exasol BucketFS Python library, 2023. URL: https://github.com/exasol/bucketfs-python.
[20] Exasol, Data science with Exasol, 2023. URL: https://github.com/exasol/data-science-examples.
[21] Exasol, Data science and UDFs examples, 2023. URL: https://docs.exasol.com/db/latest/advanced_analytics/advancedexamples.htm.
[22] Exasol, Machine learning example in R and Python, 2022. URL: https://exasol.my.site.com/s/article/Machine-Learning-Example-in-R-and-Python?language=en_US.
[23] Exasol, Exasol transformers extension, 2023. URL: https://github.com/exasol/transformers-extension.
[24] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, A. Kumar, The MADlib analytics library: Or MAD skills, the SQL, Proceedings of the VLDB Endowment 5 (2012) 1700–1711. URL: https://doi.org/10.14778/2367502.2367510. doi:10.14778/2367502.2367510.
[25] The Apache Software Foundation, Apache MADlib community artifacts, 2023. URL: https://github.com/apache/madlib-site/tree/asf-site/community-artifacts.
[26] Oracle, Machine learning in Oracle database, 2023. URL: https://www.oracle.com/artificial-intelligence/database-machine-learning/.
[27] IBM Corporation, In-database machine learning Db2 database, 2023. URL: https://www.ibm.com/docs/en/db2/11.5?topic=content-in-database-machine-learning&mhsrc=ibmsearch_a&mhq=machinelearningdatabase.
[28] M. Schule, H. Lang, M. Springer, A. Kemper, T. Neumann, S. Gunnemann, In-Database Machine Learning with SQL on GPUs, in: 33rd International Conference on Scientific and Statistical Database Management, SSDBM 2021, Association for Computing Machinery, New York, NY, USA, 2021, p. 2536. URL: https://doi.org/10.1145/3468791.3468840. doi:10.1145/3468791.3468840.
[29] N. Makrynioti, R. Ley-Wild, V. Vassalos, Machine Learning in SQL by Translation to Tensorflow, in: Proceedings of the Fifth Workshop on Data Management for End-To-End Machine Learning, DEEM '21, Association for Computing Machinery, New York, NY, USA, 2021. URL: https://doi.org/10.1145/3462462.3468879. doi:10.1145/3462462.3468879.
[30] O. Dolmatova, N. Augsten, M. H. Böhlen, A relational matrix algebra and its implementation in a column store, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, ACM, 2020. URL: https://doi.org/10.1145%2F3318464.3389747. doi:10.1145/3318464.3389747.
[31] M. Kersten, Y. Zhang, M. Ivanova, N. Nes, SciQL, a query language for science applications, in: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, AD '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 1–12. URL: https://doi.org/10.1145/1966895.1966896. doi:10.1145/1966895.1966896.
[32] Y. Zhang, M. Kersten, M. Ivanova, N. Nes, SciQL: Bridging the gap between science and relational DBMS, in: Proceedings of the 15th Symposium on International Database Engineering Applications, IDEAS '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 124–133. URL: https://doi.org/10.1145/2076623.2076639. doi:10.1145/2076623.2076639.
[33] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, M. Schleich, AC/DC: In-Database Learning

Thunderstruck, in: Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, DEEM'18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3209889.3209896. doi:10.1145/3209889.3209896.

[34] J. V. D'silva, F. D. Moor, B. Kemme, AIDA - abstraction for advanced in-database analytics, Proceedings of the VLDB Endowment 11 (2018) 1400–1413.

[35] X. Li, B. Cui, Y. Chen, W. Wu, C. Zhang, Mlog: Towards declarative in-database machine learning, Proceedings of the VLDB Endowment 10 (2017) 1933–1936.

[36] L. Chen, A. Kumar, J. Naughton, J. M. Patel, Towards linear algebra over normalized data, 2017. doi:10.14778/3137628.3137633.

[37] Exasol, Exasol community edition, 2023. URL: https://docs.exasol.com/db/latest/get_started/communityedition.htm.

[38] C. Großmann, Extending SQL for Machine Learning, Master thesis, Ostbayerische Technische Hochschule Regensburg, 2023. doi:https://doi.org/10.35096/othr/pub-6059.