**ABSTRACT**

This paper outlines the requirements of our ML2SQL compiler that allows a dedicated machine learning language (MLearn) to be run on different target architectures. The language was designed to cover an end-to-end machine learning process, including initial data curation, with the focus on moving computations inside the core of database systems. To move computations to the data, we explain the architecture of a compiler that translates into target specific user-defined-functions for the PostgreSQL and HyPer database systems. For computations inside user-defined-functions, we explain the necessary tensor datatypes and the corresponding functions. We base the explanations on an accompanying example of linear regression. To face the challenges to database systems arising from array-like data, we propose such solutions as integrating ArrayQL as stored procedures to unify the relational and array perspectives.

**CCS CONCEPTS**

- Information systems → Main memory engines;

**KEYWORDS**

SQL, Declarative Language, Database Scripting Languages

---

**1 INTRODUCTION**

Data for machine learning has to be preprocessed until it is possible to apply linear algebra to features aggregated to matrices. For data curation, text-like data such as CSV files have to be processed and cut until a suitable format is found. Afterwards, such optimisation methods as gradient descent find the optimal weights for a parameterised loss function in order to allow predictions on unlabelled data.

From a systems developer point of view, database systems form the native way of efficiently storing data in index structures. Inside of database systems, SQL, as the declarative language, simplifies data curation because it allows feature extraction as projections and selections of the only relevant tuples by design. In addition, hybrid main-memory database systems such as HyPer [4] combine transactional and analytical workload to avoid separate systems for each domain. However, for machine learning purposes, data still has to be extracted before training a model.

In the last decade, many studies have presented architectures for building end-to-end machine learning systems [1, 3, 9]. To assist computations in database systems, the support of matrix or tensor algebra is essential. Therefore, different systems tackle the challenges of representing arrays natively.
in database systems. Array database systems such as RasDaMan [2] or SciDB [8] replace tables by arrays as the native way of storing data.

As we want to move computations to the data, we have analysed the power of scripting languages within database systems. For database systems that support tensor operations, we have demonstrated the ML2SQL compiler [7] that translates a dedicated machine learning language (MLearn) into domain-specific database language extensions. This paper focuses on the architectural stack for compiling the language (s. Figure 1) and executing the result as stored procedures inside of database systems.

The paper’s main contributions are the description of the architecture behind MLearn with an accompanying example, an extension of PostgreSQL by linear algebra and gradient descent on array datatypes, as well as a look on integrating array query languages.

This work is structured as follows: firstly, the MLearn language is introduced together with the functionality of the ML2SQL compiler for translation into Python, pl/pgSQL or HyPerScript. Therefore, we will provide an example of linear regression using data read from an CSV file to show the modularity of the language. Then we will show how the ML2SQL compiler translates these statements into the target language, here as user-defined functions for PostgreSQL using pl/pgSQL. We therefore explain the technical background of features we have added, in order to run different workloads.

2 THE MLLEARN LANGUAGE

```
readcsv(
    name:taxiData
    file:'/tmp/data/small.csv'
    columns: trip_seconds,trip_miles,pickup_community_area,
    dropoff_community_area,fare,tips
    delimiter: ','
    delete empty entries 
)
```

Listing 1: CSV file loading in MLearn: the building block requires the target name, the source file, the necessary columns and the delimiter; it allows preprocessing options.

Listing 1 shows the building block for loading CSV files. It requires the target name (to create tensors with NumPy or relations in database systems), the name of the source file, the necessary columns and the delimiter, as well as the preprocessing options such as the deletion of certain entries or string replacements. When compiling for SQL, the resulting code creates a new table, copies CSV files with domain-specific commands while the preprocessing options are translated in update or delete queries (s. Listing 2).

```
copy taxiData(trip_seconds float,
    trip_miles float, pickup_community_area float,
    dropoff_community_area float, fare float, tips float);
\COPY taxiData(trip_seconds,trip_miles,
    pickup_community_area,dropoff_community_area,fare,
    tips) FROM '/tmp/data/small.csv' WITH DELIMITER ',';
```

Listing 2: The compiled code for loading CSV files in PostgreSQL: create, update and delete SQL statements are used, as well as the copy command.

Actually, MLearn is designed as a meta-language to facilitate the use of database scripting languages or Python frameworks. The ML2SQL compiler\(^1\) (s. Figure 2) produces Python or SQL code runnable in PostgreSQL or HyPer. The

\(^1\)https://gitlab.db.in.tum.de/ml2sql/ml2sql
In order to allow machine-learning-related computations within database systems, they have to provide tensors and functionalities for training a model. HyPer has already extended its array datatype to serve as tensors by allowing algebra on those types. Inside select clauses of query statements, linear algebra on array datatypes allows various computations. To reach a broader audience for our declarative machine learning language, we also provide some matrix algebra functionalities for PostgreSQL online. The operations are first precompiled as a shared library (s. Listing 4), then loaded by SQL statements (s. Listing 5). Table 1 shows the list of matrix operations needed that we implemented in PostgreSQL to allow pl/pgSQL procedures with matrix operation produced by the ML2SQL compiler.

![Table 1: Implemented matrix operations for PostgreSQL needed by MLearn.](https://gitlab.db.in.tum.de/ml2sql/psql-matrix-extension)

In addition to matrix operations, a gradient descent optimizer is essential for training models inside database systems. When compiling MLearn to SQL, we make use of our gradient descent operator supporting automatic differentiation with lambda expressions already integrated in HyPer. Lambda expressions inject user-defined code into hard-coded database operators, for example distance metrics in clustering algorithms. Here, they allow arbitrary loss functions to
be specified as shown in Listing 6. Inside, we can combine attributes from a training dataset with weights, either individually by hand or aggregated to matrices.

```sql
CREATE OR REPLACE FUNCTION regression(X float[], y float[], d float) returns float[] AS $$
begin
    Xt := matrix_transpose(X);
    return matrix_power((Xt*X),(-1)::integer)*Xt*y;
end; $$ LANGUAGE plpgsql;
```

Listing 6: Gradient descent table function using lambda expressions: inside the expression, each parameter can be combined separately or using matrix algebra.

In addition to matrix operations, we adapted our gradient descent framework to run with PostgreSQL. However, as the dimensions of PostgreSQL arrays are assigned dynamically, the PostgreSQL gradient descent operator expects the dimensions as further arguments inside the lambda expression for the loss function. Together with these extensions, we are now able to translate the code in Listing 3 to pl/pgSQL to run in PostgreSQL. Listing 7 shows an extract of the resulting code.

```sql
SELECT * FROM gradient_descent(
    λ(d,w) (w.a*d.x+w.b*d.x*c+d.y)^2,
    (select x1,x2,y from trainingdata d),
    (select a,b,c from weights w), 0.05, 100);
```

Listing 7: Resulting pl/pgSQL procedures for linear regression and increasing the matrix by the bias.

4 CONCLUSION AND ONGOING WORK

This paper has introduced the architectural details behind a declarative machine learning language (ML2SQL) with the aim of shifting computations inside of the core of database systems. It has shown how the ML2SQL compiler treats preprocessor statements to allow the inclusion of code snippets and libraries. It has provided a basic example of how to compute linear regression and referred to the necessary technical implementations such as gradient descent in PostgreSQL and HyPer. For this purpose, we have specified the required matrix operations in PostgreSQL with one coding example.

To sum up, we have discovered out that array processing represents the major building block for tasks related to machine learning. These tasks would strongly benefit from SQL especially for data preprocessing. In addition, when integrating the advantages of array database into hybrid OLTP and OLAP database systems, no domain specific systems would be required. We shall therefore work on applying matrix algebra to tables using stored procedures that are written in ArrayQL.

ACKNOWLEDGEMENT

This research has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 725286).

REFERENCES