Demonstration of OpenDBML, a Framework for Democratizing In-Database Machine Learning

Mahdi Ghorbani
University of Edinburgh
Edinburgh, United Kingdom
mahdi.ghorbani@ed.ac.uk

Amir Shaikhha
University of Edinburgh
Edinburgh, United Kingdom
amir.shaikhha@ed.ac.uk

ABSTRACT

Machine learning over relational data has been used in several applications. The traditional approach of joining relations first and then training a model on the joined table is time-consuming and requires a significant amount of memory. Recent research has focused on in-database machine learning (in-DB ML) to address this issue; these methods train the models over relations without joining, resulting in a more efficient process. However, such systems have ad-hoc user interfaces and specific data formats, making them challenging to use. To address this problem, this paper presents OpenDBML, a framework for democratizing in-DB ML. OpenDBML offers a Python interface for multiple in-DB ML systems, a set of commonly used datasets, and the ability to add new datasets and in-DB ML systems via both Python and web interfaces. The paper also presents comprehensive demonstration scenarios to illustrate how to use OpenDBML effectively.

PVLDB Reference Format:
Mahdi Ghorbani and Amir Shaikhha. Demonstration of OpenDBML, a Framework for Democratizing In-Database Machine Learning. PVLDB, 16(12): 3970 - 3973, 2023.
doi:10.14778/3611540.3611598

1 INTRODUCTION

Relational data is used for many real-world applications, such as E-commerce, finances, and healthcare. In several applications, training a machine learning model over relational data can be needed and beneficial (e.g., recommendation systems). The traditional approach to train a machine learning model over relational data is to join all relations first and then use a framework such as scikit-learn [12], TensorFlow [1], or PyTorch [11] to train a machine learning model over the joined table. Although the approach is simple to implement, joining several relations results in significant redundancy and consumes excessive amounts of time and memory.

Several in-database machine learning (in-DB ML) systems [2, 5–8, 10, 13–20] have been proposed to overcome these issues. These systems cast the in-DB ML pipeline as a linear algebra problem [2, 7, 8, 13, 20], an aggregate-join problem [5, 10, 14], or a program in a new intermediate language [15–19]. This way, the ML operators are pushed down to relations and execute ML algorithms over relational data without joining them.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611598

Despite the numerous advantages, the existing in-DB ML systems are not widely adopted for various reasons. One of the key reasons is the lack of a Python API in the majority of these systems. Since Python is the most popular language for machine learning, supporting a Python API is essential for increasing system usefulness. Another major drawback is that each system has its own data format requirements, making it more inconvenient to use. Thus, comparing the run time and accuracy of machine learning algorithms in different systems is challenging.

For instance, to use the most efficient version of LibFM [13] on relational data, the user needs to provide the data in a specialized format, called *block-structured format.* LMFAO [14] needs the data to be written in CSV files with two config files containing information about features, relations, relationships across them, and the join order. Morpheus [2] requires the data to be provided in the *normalized matrix format* and additionally expects all features of the relations to be categorical; for continuous features, the user is responsible to use binning to categorize them. For data scientists converting datasets from Pandas DataFrame [9] to these data formats is tedious and error-prone.

In this paper, we introduce OpenDBML, a framework designed to democratize in-database machine learning by making it more accessible and practical. OpenDBML provides a Python API enabling users to register new (relational and nested) datasets and in-DB ML systems to the framework. It also allows users to transform the data to the expected format by the underlying systems, train models, and measure their accuracy and run time.

OpenDBML plugs in various in-DB ML systems such as LMFAO [14], LibFM [13], or Morpheus [2]. Moreover, OpenDBML supports commonly used datasets for in-DB ML use cases (e.g., MovieLens [3], BookCrossing [21], and Walmart [4]). OpenDBML transforms all these datasets into the data format expected by each of the in-DB ML systems and allows the user to train the supported ML model by each of these systems.

The paper is organized as follows. Section 2 describes a high-level overview of the workflow of OpenDBML and its components. Section 3 explains the OpenDBML's Python API implementation, the approach to integrate new datasets and systems, and how to use OpenDBML. Our system also features a user-friendly web-based interface that connects to the Python backend and simplifies the usage for the user. Details on how to use the GUI are also included in Section 3. Finally, we illustrate several comprehensive demonstration scenarios of OpenDBML in Section 4.

2 OVERVIEW

Figure 1 depicts the high-level workflow of OpenDBML. OpenDBML reads the input dataset and performs data wrangling; it normalizes the nested data, applies feature engineering methods, and transforms the data into the expected format for the underlying in-database machine learning systems. Once the data is prepared, the framework calls these systems to train a model on the processed data. Next, we present detailed descriptions of each component.

2.1 Normalizing Nested Data

In order to support a wider range of datasets, OpenDBML converts nested datasets into relational ones. Normalization is the expansion of columns that contain nested data and converting the dataset to normal form, i.e., a flat relational format. OpenDBML implements three normalization behaviors to deal with various nesting patterns:
One-Many Normalization. When a column's data is a dictionary-like in each cell, this method expands nested rows into sub-relations.
Explode Normalization. Converts a nested column that lists items to a sub-relation with an n-hot encoded representation.
Link Normalization. Expands the nested column into a new table with its own generated primary key. Then creates an intermediate table that records the primary keys of the main table and links them to the relevant primary keys of the new sub-relation.

2.2 Feature Engineering

OpenDBML allows feature engineering over data at various levels.
String to Integer Encoding. To use string values as features in a machine learning system, they must first be encoded as integers. Moreover, since a string usually needs more space compared to an integer, storing them as integers needs less storage. OpenDBML enables the string to integer encoding.
Feature Selection. In machine learning algorithms, one might want to use only a subset of all attributes for training. OpenDBML allows users to specify their preferred features.
Categorization. Certain ML applications may require continuous values in the dataset to be categorized. This requirement can be also enforced by the underlying ML system (e.g., Morpheus). To handle this, OpenDBML provides a histogram binning method.

2.3 Data Transformation

Finally, OpenDBML transforms the data to the desired output format. Different ML systems use different inputs ranging from general design matrices to very specific formats (e.g., libFM's [13] block structure). Currently, OpenDBML supports a variety of formats: (1) Block-structured format used by libFM [13], (2) Config files and dataset format supported by LMFAO [14], (3) Normalized matrix format supported by Morpheus [2], (4) Individual CSV files for each relation, and (5) Join of relations of the dataset as a single CSV file.

For instance, to transform the dataset to the normalized matrix format, OpenDBML gets the unnested and binned data from previous stages. Then all features are one-hot encoded and stored in a sparse format. Next, the target feature is stored separately from other features. After that, a series of mappings from the number of lines in the target feature file to all other relations are created that represent how the features should be joined. Finally, all computed values are stored in files, which they can be fed to Morpheus [2].

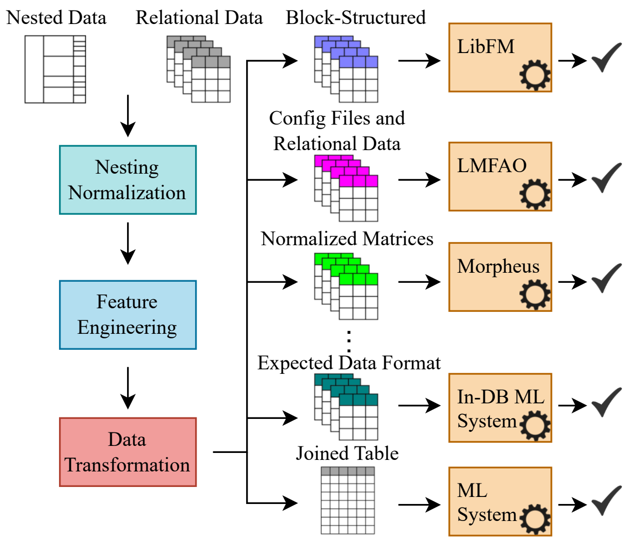


Figure 1: OpenDBML workflow overview.

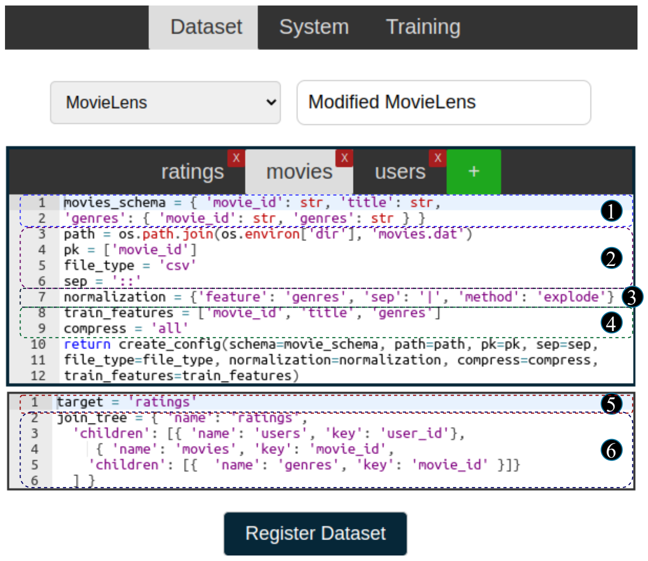
3 IMPLEMENTATION

In this section, we walk through the implementation and the web-based user interface of OpenDBML. Our GUI connects and simplifies the underlying Python API for the user. We implemented the GUI using Vue.js framework[1] and Ace Python IDE[2] for the user's convenience. Next, we describe all stages of the process, including dataset preparation, system registration, and model training.

3.1 Extending Datasets

To register a new relational dataset, its schema, primary keys, how the data should be read, and normalization and feature engineering parameters are required. As an example, we illustrate the step-by-step process of registering the MovieLens [3] dataset into our system. Figure 2 shows the dataset registration user interface. The top part of the figure is configurations for `movies` relation, and the bottom part is the general configurations for the whole dataset.
❶ *Relation's Schema.* The first step is preparing the schema of each relation in a Python dictionary format. The schema comprises a mapping of all attributes of the relation to their corresponding Python type, in the same order as the relation's columns.
❷ *Relation's Configurations.* Path to relation, type of the underlying file for the relation (e.g., CSV, XML, JSON), primary keys, and separator are provided to the user interface.
❸ *Normalization Parameters.* In the web interface, it is also necessary to specify the normalization method and the separator for any nested columns in the relation.
❹ *Feature Engineering Parameters.* The specification of string features that are transformed to an integer, the set of training features for each relation, and any other desired feature engineering parameters are shown in the user interface. Users may input `'all'` instead of explicitly listing all applicable features to apply these methods universally.

[1]https://vuejs.org/
[2]https://ace.c9.io/



```
movies_schema = { 'movie_id': str, 'title': str,
'genres': { 'movie_id': str, 'genres': str } }
path = os.path.join(os.environ['dir'], 'movies.dat')
pk = ['movie_id']
file_type = 'csv'
sep = '::'
normalization = {'feature': 'genres', 'sep': '|', 'method': 'explode'}
train_features = ['movie_id', 'title', 'genres']
compress = 'all'
return create_config(schema=movie_schema, path=path, pk=pk, sep=sep,
file_type=file_type, normalization=normalization, compress=compress,
train_features=train_features)
```

```
target = 'ratings'
join_tree = { 'name': 'ratings',
  'children': [{ 'name': 'users', 'key': 'user_id'},
     { 'name': 'movies', 'key': 'movie_id',
    'children': [{  'name': 'genres', 'key': 'movie_id' }]}
   ] }
```

Figure 2: The GUI for registering a new dataset to OpenDBML.

❺ *Target Value.* The target value for the machine learning prediction is provided in the general configuration section.
❻ *Join Order.* The general configurations panel includes a separate variable for the join tree. Join order may have `name`, `key`, `foreign-key`, `children`, and `output_name` as keys. The name of the relation is specified by `name`, and the primary key of the relation by `key`. In addition, `key` is used as the join key when the foreign key in the top relation has the same name as the primary key of the sub-relation. If their names are different, then `foreign-key` will be provided in the sub-relation. The `output_name` is used when the result of the join of the top and sub-relation needs to be stored in a file with the specified name. Finally, `children` provides all the information about sub-relations and how they should be joined with the top relation.
Specific System's Configurations. Configurations related to the specific system can be provided in the individual configuration sections or the general dataset configuration section, depending on their application. For example, if the dataset is being prepared for parallel execution using LMFAO [14], the number of threads for each relation must be specified. Otherwise, the code will run sequentially by default. For example, the dataset configuration section might include the following:

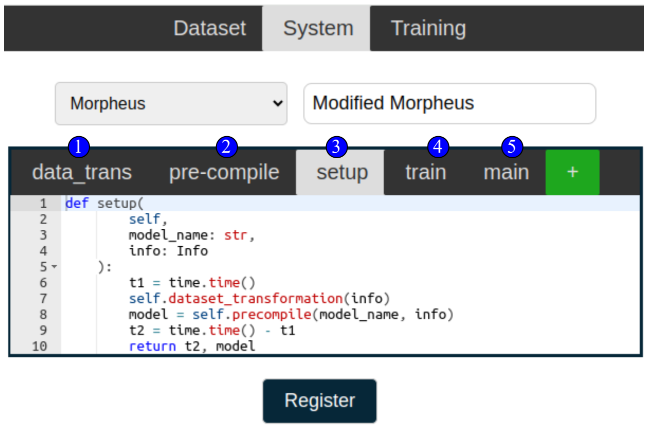
```
number_of_threads = { 'ratings': 8, 'users': 4,
                      'movies': 4, 'genres': 2 }
```

Similarly, when preparing the dataset for Morpheus [2], the number of bins for continuous features is specified in the configuration. If not specified, a default number of bins will be assigned. For instance, the `ratings`'s configuration may contain:

```
number_of_bins = { 'rating': 5 }
```

3.2 Extending Machine Learning Systems

The first step of integrating an in-database machine learning system to OpenDBML is providing an implementation for the Python API of



```
def setup(
        self,
        model_name: str,
        info: Info
    ):
        t1 = time.time()
        self.dataset_transformation(info)
        model = self.precompile(model_name, info)
        t2 = time.time() - t1
        return t2, model
```

Figure 3: The GUI for registering a new in-database machine learning system to OpenDBML.

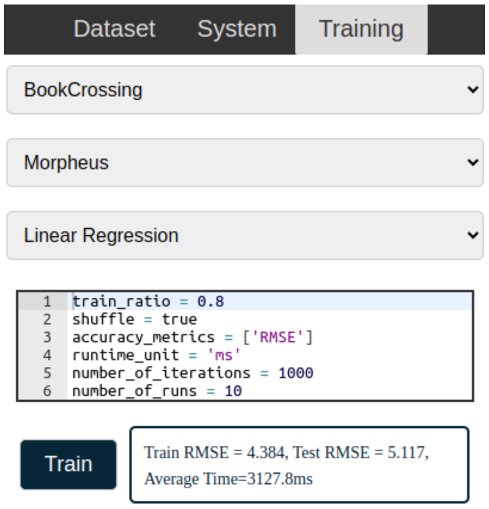
OpenDBML through the user interface. OpenDBML's API consists of four methods and a class that must be implemented. After that, the system can be registered to our framework. Figure 3 represents the user interface for system registration.
① *Data Transformation.* All systems require a function that transforms the input data into their expected format. Additionally, certain systems, such as LMFAO [14], may require configuration files that provide meta-information about the dataset (e.g., information about relations, features, and relationships across them). Implementing this function ensures that any relational dataset registered in OpenDBML can be used by new systems.
② *Pre-compilation.* This function performs pre-compilation and code generation if required. Then prepares the machine learning model and returns the resulting model object.
③ *Setup.* This function glues all the previously provided functions together, thereby carrying out all the necessary steps for setting up a machine learning model given a particular system.
④ *Training.* The underlying method of communicating with the system to train a model over a dataset given the dataset configuration is implemented here. This implementation uses the dataset information and the system's internal arguments to develop the `Model` class. Implementation of this class is necessary to execute machine learning algorithms within the system.
⑤ *Main.* This method is responsible for parsing the input arguments, calling `setup`, and training the model.

3.3 Model Training

The user interface for training a model on a dataset requires the specification of training parameters, including the train-test split ratio, whether to shuffle the dataset, the accuracy metrics, the time measurement unit, the number of training iterations, the number of runs, and any additional parameters (cf. Figure 4). If any of these parameters are not available, OpenDBML will use default values.

4 DEMONSTRATION SCENARIO

We will demonstrate three scenarios that illustrate OpenDBML's capabilities. Next, we describe each scenario.



```
train_ratio = 0.8
shuffle = true
accuracy_metrics = ['RMSE']
runtime_unit = 'ms'
number_of_iterations = 1000
number_of_runs = 10
```

```
Train RMSE = 4.384, Test RMSE = 5.117,
Average Time=3127.8ms
```

Figure 4: The user interface for training a model on a supported dataset using OpenDBML.

4.1 New Dataset Registration

Our first demonstration scenario is registering a new dataset to OpenDBML using the user interface shown in Figure 2. We do not include MovieLens [3] in our supported datasets, to begin with, and then proceed to add it. There are two ways of doing this. One way is selecting `New Dataset` from the drop-down menu, then click on the plus button and provide the name of relations and their configurations. The other way is to select an existing dataset to view its configuration and use it as a template. The tabs in the panel can be renamed, removed, or added if required. Once all the parameters for all relations are provided, we will enter the name of our new dataset and register it to our framework.

4.2 New In-DB ML System Registration

The second demonstration scenario is extending OpenDBML to support more in-database machine learning systems using the user interface shown in Figure 3. We do not include Morpheus [2] in our supported systems, to begin with, then we proceed to add it. Similar to dataset registration, this is possible in two ways. First, the user can select `New System` from the drop-down menu and implement the code in the five provided tabs. Second, the user can use an already existing system as a template for the new system's implementation. If the new system needs more auxiliary functions to be implemented, the user can use the plus button to define new functions. When the implementation is done, the user provides the name of the new system and adds it to OpenDBML.

4.3 Train a Model on a Dataset

For the third scenario, we aim to train multiple machine learning models using the in-database machine learning systems over numerous datasets in OpenDBML, including the newly registered ones. To do so, we will use the user interface shown in Figure 4. First, we will select the dataset, the machine learning system, and the underlying algorithm from the drop-down menus. Then, we will provide the required training parameters. Finally, we will click on the `Train` button to start the training process. Once the training is complete, we can see the run time and accuracy outputs. This information can be used for benchmarking considering any element of the triple (algorithm, system, dataset) be fixed (F) or varying (V). It can be used in many cases, such as 1) (F, V, V) to compare systems, 2) (V, F, F) to compare algorithms, 3) (F, F, V) to benchmark a specific algorithm, 4) (V, V, F) to find the best model for a specific dataset, and 5) (V, F, V) to benchmark a specific system.

ACKNOWLEDGEMENT

The authors thank Austin Pan and Katherine McClorey for their help with the software development and Huawei for their support of the distributed data management and processing laboratory at the University of Edinburgh.

REFERENCES

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[2] Lingjiao Chen, Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. 2017. Towards Linear Algebra over Normalized Data. *Proc. VLDB Endow.* 10, 11 (aug 2017), 1214–1225.
[3] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. https://doi.org/10.1145/2827872
[4] Kaggle. 2014. Walmart Recruiting. https://www.kaggle.com/competitions/walmart-recruiting-store-sales-forecasting/. Accessed: 2022-04-20.
[5] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2020. Learning models over relational data using sparse tensors and functional dependencies. *TODS* 45, 2 (2020), 1–66.
[6] Mijung Kim. 2014. *TensorDB and tensor-relational model (TRM) for efficient tensor-relational operations.* Arizona State University.
[7] Arun Kumar, Jeffrey Naughton, and Jignesh M Patel. 2015. Learning generalized linear models over normalized data. In *SIGMOD.* 1969–1984.
[8] Side Li, Lingjiao Chen, and Arun Kumar. 2019. Enabling and Optimizing Non-Linear Feature Interactions in Factorized Linear Algebra. In *SIGMOD.* New York, NY, USA, 1571–1588.
[9] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.
[10] Dan Olteanu. 2020. The relational data borg is learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3502–3515.
[11] Adam Paszke et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 8024–8035.
[12] Fabian Pedregosa et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.
[13] Steffen Rendle. 2013. Scaling Factorization Machines to Relational Data. *Proc. VLDB Endow.* 6, 5 (mar 2013), 337–348. https://doi.org/10.14778/2535573.2488340
[14] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q Ngo, and XuanLong Nguyen. 2019. A layered aggregate engine for analytics workloads. In *SIGMOD.* 1642–1659.
[15] Hesam Shahrokhi and Amir Shaikhha. 2023. Building a Compiled Query Engine in Python. In *CC'22.* 180–190.
[16] Amir Shaikhha, Mathieu Huot, Jaclyn Smith, and Dan Olteanu. 2022. Functional collection programming with semi-ring dictionaries. *Proceedings of the ACM on Programming Languages* 6, OOPSLA1 (2022), 1–33.
[17] Amir Shaikhha, Marios Kelepeshis, and Mahdi Ghorbani. 2023. Fine-Tuning Data Structures for Query Processing. In *CGO'23.* ACM, 149–161.
[18] Amir Shaikhha, Maximilian Schleich, Alexandru Ghita, and Dan Olteanu. 2020. Multi-layer optimizations for end-to-end data analytics *(CGO'20).* ACM, 145–157.
[19] Amir Shaikhha, Maximilian Schleich, and Dan Olteanu. 2021. An intermediate representation for hybrid database and machine learning workloads. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2831–2834.
[20] Keyu Yang, Yunjun Gao, Lei Liang, Bin Yao, Shiting Wen, and Gang Chen. 2020. Towards factorized svm with gaussian kernels over normalized data. In *2020 IEEE 36th International Conference on Data Engineering (ICDE).* IEEE, 1453–1464.
[21] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving Recommendation Lists through Topic Diversification *(WWW '05).* ACM, New York, NY, USA, 22–32.